Maskeringstechnieken gebruiken bij auteurherkenning

Optimale aantal te maskeren woorden vinden, rekeninghoudend met de prestaties en onderwerprobuustheid van het model

door

A.I. Elias

om de graad Bachelor of Science te behalen

aan de Technische Universiteit Delft,

wordt in het openbaar verdedigd op donderdag 30 januari 2025 om 14.00.

Studentnummer:	5362628	
Projectduur:	1 oktober, 2024 – 30 januari, 2025	
Scriptiecommissie:	Dr. J. Söhl,	TU Delft, begeleider
	Dr. C. Kraaikamp,	TU Delft, beoordelaar

Een elektronische versie van deze scriptie is te vinden op `http://repository.tudelft.nl/`.

TU Delft

Samenvatting

Maskeringstechnieken kunnen nuttig zijn bij auteurherkenning om auteurherkenningsmethoden minder onderwerpafhankelijk te maken. Echter, als er teveel gemaskeerd wordt gaat er relevante informatie verloren waardoor de auteurherkenningsmethode juist minder goed werkt. Hierin moet een zorgvuldige afweging gemaakt worden.
Het doel van dit onderzoek is om een aanbeveling te doen wat betreft het optimale aantal te maskeren woorden voor verschillende datasets. Ook worden er verschillende maskeringstechnieken bij verschillende classificatiemethoden vergeleken. Er wordt gekeken naar maskering met behulp van een algemene woordenlijst (COCA-woordenlijst) en maskering met behulp van een eigen frequentielijst per dataset. Hierbij worden twee classificatiemethoden gebruikt: support vector machines en logistische regressie. Voor drie verschillende datasets: tweets, literaire teksten en brieven wordt gekeken welke manier van maskeren en welke classificatiemethode het beste werkt. Dit wordt gedaan door de prestaties te vergelijken. Ook wordt er daarnaast, vanuit de literatuur, gekeken naar de verschillen in onderwerprobuustheid en op basis daarvan wordt, gecombineerd met de informatie wat betreft de prestaties, een aanbeveling gedaan wat betreft het optimale aantal te maskeren woorden.

A.I. Elias
Delft, januari 2025

Inhoudsopgave

1

Inleiding

”Desinformatie, misinformatie en nepnieuws zijn een probleem. Dit zijn namelijk vormen van misleidende en onjuiste informatie, die tot onrust en onzekerheid in de samenleving kunnen leiden. Daarnaast zijn er steeds meer manieren om desinformatie en nepnieuws te verspreiden. En is het steeds moeilijker te herkennen. Het is daarom belangrijk dat iedereen in Nederland kritisch en nieuwsgierig blijft naar waar nieuws vandaan komt.”

-Rijksoverheid, 2024 [11]

Nepnieuws is het gesprek van de dag. De een maakt zich flink zorgen, een ander gelooft dat die wel kritisch en scherp genoeg is om de juiste informatie te filteren, en weer een ander neemt alles voor waarheid aan wat die voorgeschoteld krijgt. Een ding is in ieder geval duidelijk: de hoeveelheid desinformatie, misinformatie en nepnieuws is de afgelopen jaren flink toegenomen met de opkomst van internet, sociale media en AI. En dat is een zorgelijke zaak wat betreft de Rijksoverheid.

Auteurherkenningsmethoden kunnen hier een uitkomst voor bieden [7]. Met behulp van auteurherkenning kan bepaald worden of bepaalde teksten wel echt geschreven zijn door de auteurs aan wie die toegeschreven worden. Ook kunnen meerdere teksten van dezelfde bron herkend worden, wat kan helpen bij bijvoorbeeld het opsporen van geautomatiseerde botnetwerken.

Dit is pas slechts één van de vele toepassingen van auteurherkenning. Auteurherkenning heeft een ontzettend brede en interdisciplinaire belangstelling en het speelt een rol in veel verschillende vakgebieden [7]. Daarom is dit een zeer nuttig onderwerp om te onderzoeken en om te kijken op welke manieren auteurherkenning het beste werkt.

In dit onderzoek wordt er specifiek gekeken naar het toepassen van maskeringstechnieken op teksten voor auteurherkenning. Door maskeringstechnieken te gebruiken en bepaalde woorden te maskeren, komt het onderwerp van de tekst minder naar voren waardoor deze een minder grote rol speelt in de auteurherkenning. Dit is gunstig want men wil immers niet dat twee teksten aan dezelfde auteur worden toegeschreven, alleen maar omdat ze over hetzelfde onderwerp gaan. Aan de andere kant, als er te veel woorden gemaskeerd worden, zal er te weinig informatie overblijven om de teksten met auteurherkenningsmethoden juist te classificeren. Door het maken van verschillende afwegingen moet hier een juiste balans in worden gevonden.

De onderzoeksvraag die daarmee in dit onderzoek centraal staat is als volgt:

’Wat is het optimale aantal woorden om te maskeren in teksten voor auteurherkenning, rekening houdend met de classificatieprestaties en onderwerprobuustheid van het model?’

Om tot een antwoord op deze onderzoeksvraag te komen zullen er verschillende stappen ondernomen

worden die worden besproken in de verschillende hoofdstukken van dit verslag.

Allereerst wordt er een voorbereidend literatuuronderzoek uitgevoerd, wat beschreven wordt in hoofdstuk 2. Er wordt gekeken naar de geschiedenis van auteurherkenning, toepassingen van auteurherkenning, beschikbare datasets en eerder gedaan onderzoek naar auteurherkenning en de toepassing van maskeringstechnieken. Hieruit wordt besloten om voor drie verschillende datasets, bestaande uit verschillende soorten teksten, te kijken naar twee verschillende maskeringstechnieken voor twee verschillende classificatiemethoden.

Vervolgens moeten alle drie de datasets voorbereid ofwel 'schoongemaakt' worden voordat deze gebruikt kunnen worden. Hiervoor wordt voor iedere dataset heel specifiek gekeken of er onregelmatigheden of onrelevante informatie in staat. Dit proces staat beschreven in hoofdstuk 3.

Als dat is gebeurd, kunnen de maskeringstechnieken toegepast worden op de datasets, in hoofdstuk 4. Er worden twee verschillende maskeringstechnieken onderzocht: maskering door middel van een algemene woordenlijst uit de Engelse taal en maskering door middel van een eigen woordenlijst op basis van de dataset.

Vervolgens zal er gekeken worden naar op welke manieren de prestaties van de modellen gevalideerd kunnen worden in hoofdstuk 5.

In hoofdstuk 6 worden dan de prestaties van de verschillende maskeringstechnieken bij de verschillende classificatiemethoden (support vector machines en logistische regressie) voor de verschillende datasets weergegeven in grafieken en worden deze geanalyseerd.

Daarna zal er gekeken worden naar de onderwerprobuustheid van de verschillende mogelijkheden in hoofdstuk 7. Het is namelijk belangrijk om de prestaties in combinatie met de onderwerprobuustheid te zien om een goede afweging wat betreft het optimale aantal te maskeren woorden te kunnen maken.

In hoofdstuk 8 worden er uiteindelijk conclusies over de onderzoeksvraag getrokken. Ook wordt er gekeken naar de beperkingen van dit onderzoek en worden er aanbevelingen voor vervolgonderzoek gedaan.

.

2

Voorbereidend literatuuronderzoek

2.1. Verschilllende probleemstellingen binnen auteurherkenning

In de literatuur zijn verschillende definities van de term 'auteurherkenning' te vinden. Een definitie is als volgt: 'Auteurherkenning identificeert de auteur van een onbekend document, tekst, bron, code of binair bestand' [7]. Dit is een definitie die alleen geschreven teksten omvat. Er zijn ook bredere definities te vinden zoals die van Juola: 'Auteurherkenning is een poging om uit taalkundige data kenmerken van de auteur af te leiden'. Dit is een bredere definitie die bijvoorbeeld ook spraakherkenning omvat [10].

Binnen de auteurherkenning zijn verschillende onderzoeksgebieden en probleemstellingen te onderscheiden.

Allereerst zijn de probleemstellingen op te delen in 'gesloten-verzamelingproblemen' en 'open-verzamelingproblemen'. Bij een gesloten-verzamelingprobleem is vooraf een verzameling van mogelijke auteurs beschikbaar en is het zeker dat een van deze auteurs de onbekende tekst geschreven heeft. Bij een open-verzamelingprobleem is vooraf niet duidelijk of de onbekende tekst geschreven is door een van de beoogde auteurs [14].

Binnen deze open- en gesloten-verzamelingproblemen zijn nog specifiekere probleemstellingen te onderscheiden. Enkele daarvan zijn [7]:

- Auteurherkenning ('Authorship Attribution', kortweg AA): het identificeren van de auteur(s) van een tekst of verzameling van teksten.
- Auteurtypering ('Authorship Characterization', kortweg AC): het bepalen van eigenschappen van de auteur zoals bijvoorbeeld geslacht, leeftijd, beroep of opleidingsniveau op basis van de tekst die hij/zij heeft geschreven.
- Auteurverificatie ('Authorship Verification', kortweg AV): checken of een verzameling van teksten door dezelfde auteur is geschreven of door verschillende auteurs.
- Plagiaatdetectie ('Plagiarism Detection', kortweg PD): onderzoeken of zinnen of paragrafen geproduceerd zijn uit bestaande teksten van een auteur.

In dit onderzoek zal ingegaan worden op de eerste mogelijkheid, auteurherkenning (AA).

2.2. Geschiedenis van de auteurherkenning

In de geschiedenis van de auteurherkenning zijn vele theorieën opgekomen. En ook vele theorieën die na beter onderzoek weer zijn verworpen. Deze theorieën betreffen factoren die te maken zouden kunnen hebben met het identificeren van schrijvers. Denk hierbij aan gemiddelde zinslengte, gemiddelde woordlengte, gemiddeld aantal lettergrepen per woord, verdeling van woordsoorten en type-token-ratio (de hoeveelheid unieke woorden in een tekst ten opzichte van het totaal aantal woorden in een tekst)

[10].

Zo werd de cumulatieve sommethode ook als een geschikt middel geacht om te gebruiken voor auteurherkenning. De cumulatieve sommethode is een methode om overeenkomsten tussen verschillende reeksen te meten [10]. Als men van verschillende teksten bijvoorbeeld de woordlengtes in een reeks zet, kan er bepaald worden wat de cumulatieve som van deze verschillende reeksen is. Dat geeft de overeenkomst aan tussen de woordlengtes van de reeksen en op basis daarvan zou dan de auteur bepaald kunnen worden.

Deze cumulatieve sommethode werd in 1991 al snel als forensische methode gebruikt in verschillende Engelse rechtszaken. Maar niet lang daarna werd de nauwkeurigheid van deze methode in sterk twijfel getrokken. Er werden verschillende onderzoeken gepubliceerd die lieten zien dat er niet genoeg onderbouwing voor de cumulatieve sommethode is en dat de resultaten niet accuraat genoeg zijn om in de praktijk (specifiek in rechtszaken) te gebruiken [10].

Wie er wel in slaagden om met nieuwe, succesvolle methoden te komen voor auteurherkenning, waren Mosteller en Wallace. Zij speelden een grote rol in een van de bekendste cases in de geschiedenis waarin auteurherkenning wordt toegepast: de 'Federalist Papers' [8].

Tussen 1787 en 1788 werden 77 artikelen (geschreven in briefvorm) gepubliceerd in vijf New Yorkse kranten. Deze brieven hadden als doel om New Yorkers te overtuigen om het nieuwe ontwerp van de Amerikaanse Grondwet te ondersteunen en verschenen onder het pseudoniem 'Publius'. Velen speculeerden over de identiteit van de geheimzinnige Publius. Uiteindelijk gaf Alexander Hamilton toe dat hij de Federalist Papers had geschreven. Maar het bleek dat hij ze niet allemaal alleen had geschreven [8].

De Federalist Papers zijn dus deels door Alexander Hamilton geschreven. Alexander Hamilton was een generaal. Hij stierf in 1804 tijdens een gevecht met Aaron Burr. In 1807 ontving een tijdschrift uit Philadelphia een lijst die Hamilton vlak voor het fatale duel had opgesteld. Op die lijst had hij de specifieke auteurs van de Federalist Papers gezet: hijzelf, John Jay en James Madison (de vierde president van de Verenigde Staten). Madison heeft tot het eind van zijn presidentschap ontkend dat hij een auteur van de Federalist Papers was. Maar uiteindelijk gaf hij aan dat hij nummers 49 tot en met 58 en nummers 62 en 63 had geschreven. Echter, dit waren nummers waarvan Hamilton ook al had aangegeven dat hij die had geschreven. 12 van de 58 artikelen werden geclaimd door zowel Hamilton als Madison, maar wie had deze écht geschreven? [8]

Lange tijd was er onduidelijkheid over wie de artikelen precies geschreven had. Tot in 1964 Mosteller en Wallace een boek 'Inference and Disputed Authorship: the Federalist' publiceerden. Hierin onderzochten zij het auteurschap van de 12 Federalist Papers waarvan het auteurschap onduidelijk was. Dit deden zij op verschillende historische en stilistische manieren. Vervolgens, in 1984, publiceerden zij nog zes verschillende studies die op verschillende manieren die vraag onderzochten [8].

Als eerste ingeving wilden Mosteller en Wallace synoniemenparen gebruiken om te achterhalen wie de artikelen hadden geschreven. Denk bijvoorbeeld aan het synoniemenpaar: 'kapot' en 'stuk'. De auteur heeft complete vrijheid om zelf te bepalen welke hij gebruikt zonder dat de betekenis van de zin verandert. Door te kijken of de ene auteur consistent voor het ene synoniem kiest en de andere auteur voor de ander kan er op een consistente, onderwerponafhankelijke manier bepaald worden wie de auteur is [10].

Mosteller en Wallace probeerden dit toe te passen op de Federalist Papers maar kwamen erachter dat er niet genoeg synoniemenparen waren om dit te doen. In plaats daarvan focusten ze zich vervolgens op functiewoorden. Voorbeelden van functiewoorden zijn lidwoorden, voegwoorden, voornaamwoorden, telwoorden en voorzetsels. Deze woorden hebben zelf weinig inhoud maar geven goed de relatie tussen de woorden in een zin. Functiewoorden zijn onderwerponafhankelijk en kunnen daarom gebruikt worden als nuttige indicatoren om een auteur te bepalen [10].

Mosteller en Wallace analyseerden de verdeling van 30 functiewoorden in de Federalist Papers. Zij kwamen hierdoor tot de conclusie dat alle betwiste teksten door Madison waren geschreven (en niet door Hamilton). Deze analyse werd een van de bekendste en meest geciteerde, statische analyses van auteurherkenning. Hierdoor werden de Federalist Papers daarna een belangrijke maatstaf om nieuwe auteurherkenningsmethoden op te testen [10].

Tegenwoordig zijn er ontzettend veel manieren waarop men tot auteurherkenning kan komen. Er kan naar stylometrische kenmerken worden gekeken, maar er kan goed ook machine learning of deep learning worden toegepast. Het ligt aan de situatie welke manier het beste werkt [6].

2.3. Toepassingen van auteurherkenning

Auteurherkenning heeft een brede en interdisciplinaire belangstelling. Het speelt een grote rol in veel vakgebieden zoals forensische analyse, plagiaatcontrole, detectie van beveiligingsaanvallen, patentclaims, schending van auteursrecht en softwarediefstal [7].

De mensen die geïnteresseerd zijn in de uitkomsten van auteurherkenningsmethoden en dit gebruiken in hun vakgebied, zijn dus vaak geen statistische of taalkundige professionals. Het zijn leraren die kijken naar tekenen van plagiaat, journalisten die de validiteit van bronnen checken, rechercheurs die een misdaad onderzoeken of advocaten die twisten over een omstreden testament [10].
Daar komt de moeilijkheid bij kijken dat zij vaak niet alle details van de methode weten. Dit kan leiden tot een onjuiste interpretatie van de resultaten. Een goede auteurherkenningsmethode heeft daarom een ingebouwde betrouwbaarheidsbeoordeling voor de gebruiker nodig [10].

Traditioneel gezien gaat auteurherkenning over het identificeren van formele teksten zoals essays en romans. Maar recentelijk wordt er steeds meer aandacht gegeven aan teksten die gegenereerd zijn door online gebruikers, zoals e-mails, blogs en chatberichten. Auteurherkenning van online teksten is een andere en moeilijkere taak dan bij traditionele teksten. Online teksten zijn namelijk vaak kort, bevatten vaak spreektaal en spellingsfouten, en het aantal mogelijke auteurs is ook veel groter [14].

Zoals ook in de Inleiding besproken is een van de bijkomende problemen in het digitale tijdperk de verspreiding van desinformatie. Denk hierbij aan nepnieuws, wetenschapsontkenning en haatuitingen, verspreid via sociale media. Auteurherkenning kan hier ook op toegepast worden om desinformatie en valse identiteiten te identificeren [7].

Het is niet alleen zo dat auteurherkenning in andere vakgebieden gebruikt wordt, andere vakgebieden hebben ook invloed op de ontwikkeling van de auteurherkenning. Zo is de techniek van de auteurherkenning de afgelopen jaren sterk verbeterd door de nieuwe ontwikkelingen in de taalverwerking, machine learning en statistische analyse [14].

2.4. Datasets

In dit onderzoek zal er naar auteurherkenning gekeken worden in drie verschillende datasets.

De eerste dataset bevat tweets van vijf bekende Amerikanen: Barack Obama, Ellen DeGeneres, Sebastian Ruder, Katy Perry en Neil deGrasse Tyson. Deze dataset is verkregen via Kaggle (`https://www.kaggle.com/`). Kaggle is een groot datawetenschapsplatform waarop datasets en handige tools voor datascience en machine learning beschikbaar worden gesteld. De dataset bevat ongeveer 10.000 teksten. Omdat de teksten in deze dataset tweets zijn, zijn het relatief korte teksten met daarin vaak informeel taalgebruik. De teksten zijn na 2000 verkregen.

De tweede dataset is een van de grootste openbaar beschikbare datasets op het gebied van auteurherkenning. Deze Engelse dataset bevat teksten van vijftig bekende auteurs uit de Victoriaanse tijd (Verenigd Koninkrijk, 1837 tot 1901). De dataset genaamd ‘Gungor 2018 VictorianAuthorAttribution Data’ is afkomstig uit het onderzoek van Gungor. Gungor stelde deze dataset op met als doel om het onderzoeksgebied van de auteurherkenning te voorzien van een grote beschikbare dataset. Ook publiceerde hij in zijn onderzoek enkele functionaliteiten en codes die als eerste stap gebruikt

kunnen worden in onderzoek naar auteursherkenning [5]. In totaal bevat de dataset meer dan 50.000 teksten. Omdat deze teksten uit boeken komen, zijn de teksten in deze dataset erg lang en formeel (het is gepubliceerd werk). De teksten komen uit de 19de eeuw dus bevatten ook taalgebruik uit die tijd.

De derde dataset bevat de 'Federalist Papers', ook al besproken in 2.2. Deze teksten komen uit de 18e eeuw (1787 en 1788) en het zijn brieven om New Yorkers te overtuigen om het nieuwe ontwerp van de Amerikaanse Grondwet te ondersteunen. In totaal bevat deze dataset 74 teksten van drie verschillende auteurs: Alexander Hamilton, John Jay en James Madison. Deze dataset is dus substantieel kleiner dan de andere twee datasets. Deze set bevat middellange, formele teksten met taalgebruik uit de 18e eeuw. Deze dataset is, net als de eerste dataset, via Kaggle (`https://www.kaggle.com/`) verkregen.

Zoals te zien is, zijn de datasets die in dit onderzoek gebruikt worden erg verschillend. De datasets bevatten lange en korte teksten, formele en informele teksten, en teksten uit verschillende tijdperken. Het is interessant om te vergelijken in hoeverre prestaties en onderwerpafhankelijkheid wat betreft auteurherkenning in combinatie met het maskeren verschilt tussen deze verschillende datasets.

Omdat de rekentijd van de classificatie bij de Victoriaanse dataset heel groot bleek te zijn (door de grootte van de dataset) is ervoor gekozen om de classificatie en berekeningen voor de Victoriaanse dataset in dit onderzoek uit te voeren met een kwart (25 procent) van het totaal aantal teksten in de hele Victoriaanse dataset.

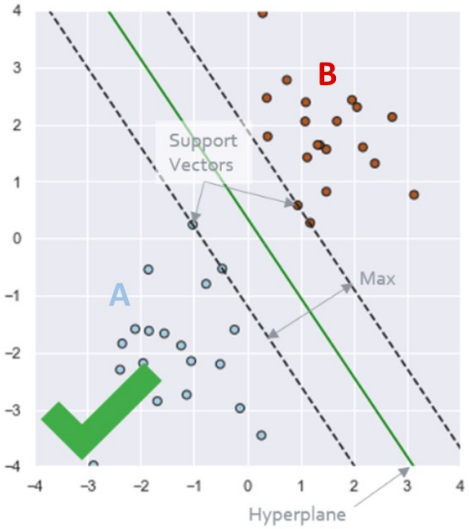
2.5. Support vector machines

Auteurherkenning is in veel gevallen een multiklasseverificatieprobleem ('multiclass verification'). Bij auteurherkenning moet er namelijk een tekst geclassificeerd worden die mogelijk van meerdere auteurs zou kunnen zijn. Deze verschillende auteurs corresponderen met meerdere, verschillende klassen [4].

Een multiklasseprobleem houdt in dat het originele probleem opgedeeld kan worden in binaire subproblemen. Elk binair subprobleem wordt dan individueel behandeld en opgelost. Nadat dit bij elk subprobleem is gebeurd, wordt het uiteindelijke resultaat van het grote probleem bepaald. Multiklasseverificatieproblemen kunnen op verschillende manieren met machine learning worden aangepakt [4].

Een van de machine-learningtechnieken om een multiklasseverificatieprobleem op te lossen is het gebruik van support vector machines (SVMs). Uit eerder onderzoek naar auteurherkenning ([6]) blijkt dat SVMs goed presteren voor auteurherkenning in vergelijking met andere methoden. Daarom zal SVM een van de methodes zijn die in dit onderzoek voor auteurherkenning wordt gebruikt.

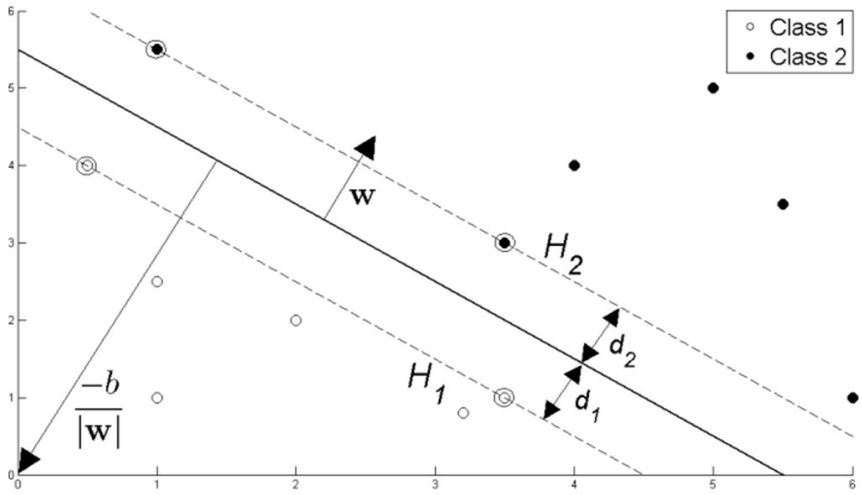
Figure 2.1: Grafische weergave van SVM [16]



Om de wiskunde achter de methode van SVMs te begrijpen, zal er eerst gekeken worden naar een grafische weergave van SVM. Een grafische weergave hiervan is weergegeven in 2.1. In de figuur is te zien dat er sprake is van een dataset met twee verschillende eigenschappen. De dataset kan op basis van die eigenschappen worden onderverdeeld in twee groepen: groep A (lichtblauw) en groep B (rood). Deze groepen kunnen bijvoorbeeld overeenkomen met teksten van een auteur en teksten van een andere auteur. Er is nu een methode nodig om voor een willekeurig nieuw datapunt te kunnen bepalen tot welke groep hij hoogstwaarschijnlijk toebehoort. SVM doet dat door gebruik te maken van 'support vectoren'. Dat zijn de datapunten van de verschillende groepen waar, als daar parallele lijnen doorheen getrokken worden, de onderlinge afstand tussen de lijnen het maximaal is [16]. De lijn of het hypervlak wat daar precies tussenin ligt, geeft dan de scheidingslijn tussen de twee groepen aan. In figuur 2.1 worden dus alle datapunten onder de groene lijn toegewezen aan groep A en boven de groene lijn aan groep B.

Er zal nu ook een stukje wiskundige uitleg van SVM gegeven worden. Bij de wiskundige uitleg van SVM zal de onderstaande figuur 2.2 ondersteuning bieden.

Figure 2.2: Hypervlak tussen twee scheidbare klassen [3]



Laat L het aantal trainingsdatapunten waar elke input $\mathbf{x}_i$ dimensie D heeft en in klasse $y_i = +1$ of $y_1 = -1$ is [3].

De trainingsdata is dan dus van de vorm:

$\{\mathbf{x}_i, y_i\}$ waar $i = 1, 2, ..., L$, $y_i = \{-1, 1\}$, $\mathbf{x} \in \mathbb{R}^D$ [3]

Hier wordt aangenomen dat de data lineair scheidbaar is. Dat betekent dat als $D = 2$ er een lijn getrokken kan worden in een grafiek van x_1 versus x_2 die de twee klassen scheidt en als $D > 2$ er een hypervlak in de grafieken van $x_1, x_2, ..., x_D$ getrokken kan worden die de klassen scheidt [3].

Dit hypervlak, wat te zien is in figuur 2.2, kan beschreven worden als $\mathbf{w} \cdot \mathbf{x} + b = 0$ [3]. Hierin is $\mathbf{w}$ de normaalvector van het hypervlak en $\frac{|b|}{||\mathbf{w}||}$ is de loodrechte afstand van het hypervlak naar de oorsprong [3].

De support vectoren zijn de vectoren van de datapunten die het dichtstbij het scheidende hypervlak liggen. Deze punten zijn omcirkeld in figuur 2.2. Het doel van SVM is om het hypervlak zó te orienteren dat het zo ver als mogelijk van de dichtsbijzijnde datapunten in de klassen ligt. Kijkend naar figuur 2.2 komt dat neer op het selecteren van de variabelen $\mathbf{w}$ en b zó dat de trainingsdata beschreven wordt als [3]:

$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1$ voor $y_i = +1$

$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1$ voor $y_i = -1$

Dit kan gecombineerd worden tot [3]:

$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \; \forall i$

Als er in figuur 2.2 wordt gekeken naar de support vectoren dan zijn er twee vlakken waar deze punten op liggen, H_1 en H_2, hiervoor geldt [3]:

$\mathbf{x}_i \cdot \mathbf{w} + b = +1$ voor H_1

$\mathbf{x}_i \cdot \mathbf{w} + b = -1$ voor H_2

Definieer d_1 als de afstand van H_1 naar het hypervlak en d_2 de afstand van H_2 naar het hypervlak [3].

Omdat het hypervlak gelijke afstand heeft tot H_1 en H_2 betekent dat dat $d_1 = d_2$. Dit is de SVM's marge en deze moet gemaximaliseerd worden om het hypervlak zo ver als mogelijk van de support vectoren te krijgen [3].

Deze marge is gelijk aan $\frac{1}{||\mathbf{w}||}$ en om dat te maximaliseren krijgt men het volgende probleem [3]:

min$||\mathbf{w}||$ zo dat $y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \; \forall i$

Dit is een optimalisatieprobleem wat op te lossen is door Lagrange multipliers te gebruiken.

Bij hypervlakken die de data in verschillende klassen scheiden maar niet-lineair zijn, kan men de kernel trick toepassen op SVM. Binnen de toepassing van SVM op auteurherkenning wordt dit niet vaak gebruikt. In het onderzoek van Hajer is ook te zien dat dit zelfs leidde tot een verslechtering van de prestaties van SVM voor auteurherkenning [6]. Daarom zal in dit onderzoek geen gebruik gemaakt worden van kernels.

Twee bekende classificatiemethoden van SVMs die gebruikt worden bij auteurherkenning zijn: een-naar-een-classificatie (one-vs-one) en een-naar-rest-classificatie (one-vs-rest) [6].

Bij een-naar-een-classificatie worden er per geval twee klassen behandeld. Met behulp van de trainingsdata wordt voor elk mogelijk paar klassen een individuele SVM gemaakt. Dit geeft dan bij n verschillende auteurs (en dus n verschillende klassen) $\frac{n(n-1)}{2}$ classificatoren. Voor een onbekende tekst (testdatapunt) bepaalt iedere classificator tot welke van de twee klassen het testdatapunt het waarschijnlijkst toebehoort. Uiteindelijk wordt de onbekende tekst toegewezen aan de klasse (auteur) die het meest naar voren kwam in alle individuele classificatoren [6].

Bij een-naar-rest-classifatie worden er n classificatoren gemaakt. Elke classificator is een SVM die de trainingsdata (bekende teksten) van een klasse (auteur) vergelijkt met de trainingsdata van alle andere klassen. Elke classificator berekent dan een score die de afstand aangeeft tot de onbekende tekst. Uiteindelijk wordt de onbekende tekst toegewezen aan de klasse met de hoogste score [6].

2.6. Logistische regressie

Naast SVMs blijkt uit eerder onderzoek dat logistische regressie ook een effectief machine learning model is om te gebruiken voor auteurherkenning [13].

Logistische regressie wordt vaak gebruikt als methode om binaire variabelen te classificeren [18]. Bij logistische regressie is het doel om de waarschijnlijkheid te bepalen dat een voorbeeld tot een van de twee categorieën behoort. Bijvoorbeeld dat een tekst geschreven is door auteur A of dat een tekst niet geschreven is door auteur A.

Om die waarschijnlijkheid te bepalen wordt de logistische functie $f(z) = \frac{1}{1+e^{-z}}$ gebruikt. Hierbij geldt dat $z = \beta_0 + \sum_{i=1}^{n} \beta_i x_i$. Hierin is β_0 de bias, de verschuiving van de voorspelling. $x_1, x_2, \ldots, x_n$ zijn de verschillende voorspellende variabelen en $\beta_1, \beta_2, \ldots, \beta_n$ zijn de bijbehorende gewichten van de voorspellende variabelen x_i. Naast deze waarden en variabelen bevat het model ook nog een binaire, afhankelijke variabele y. y heeft de waarde 0 of 1. Aan die waarden kan men zelf de context koppelen, bijvoorbeeld 1 is dat de tekst wel door auteur A is geschreven en 0 is dat de tekst niet door auteur A is geschreven [18].

Om de waarschijnlijkheid te bepalen dat de afhankelijke variabele y 0 of 1 is, wordt gekeken naar

de verwachtingswaarde [18]:

$$E(y) = \frac{1}{1 + e^{\beta_0 + \sum_{i=1}^{n} \beta_i x_i}} \tag{2.1}$$

Omdat men hier te maken heeft met een discrete kansverdeling geldt het volgende

$$E(y) = \sum_{y} y * p(y) = 0 * p(y = 0) + 1 * p(y = 1) = p(y = 1) \tag{2.2}$$

Binaire logistische regressie, zoals hierboven beschreven, is ook uit te breiden naar meer dan twee klassen. Dit heet multinomiale logistische regressie [9].

2.7. Maskeringstechnieken

Auteurherkenningsmethoden kunnen beïnvloed worden door het onderwerp of genre van de teksten. Dat is vervelend want bij auteurherkenning is het niet altijd realistisch om aan te nemen dat de bekende teksten van de auteurs (trainingsdata) en de onbekende teksten (testdata) van hetzelfde genre zijn of over hetzelfde onderwerp gaan [15].

Er zijn verschillende modellen in auteurherkenning waarbij onderwerp en genres in acht worden genomen [15]:

1. Cross-onderwerpherkenning (‘cross-topic attribution’): hierbij is het onderwerp van de testdata anders dan het onderwerp van de trainingsdata. Wel behoren de test- en trainingsdata tot hetzelfde genre.
2. Cross-genreherkenning (‘cross-genre-attribution’): hierbij behoren de trainingsdata en testdata tot verschillende genres. Maar wel alle data behoren tot hetzelfde thematische domein (qua onderwerp).
3. Cross-onderwerp-en-genreherkenning (‘cross-topic-and-genre attribution’): dit is het lastigste geval waarbij zowel het onderwerp als het genre van de trainings- en testdata verschillen.

Een van de manieren waarop teksten geclassificeerd kunnen worden, waarbij de methode niet beïnvloed wordt door genre of onderwerp is door maskeringstechnieken te gebruiken. Hierbij worden de teksten (zowel trainingsdata als onbekende teksten) gedeeltelijk aangepast en de minst voorkomende woorden worden gemaskeerd.
Het maskeren wordt gedaan door de minst voorkomende woorden te vervangen door een symbool, bijvoorbeeld *. Ook worden cijfers vervangen door een ander symbool, bijvoorbeeld # [15].
Dit resulteert in een tekst waarin vooral onderwerpafhankelijke en genrespecifieke woorden worden gemaskeerd waardoor uiteindelijk vooral de tekstuele structuur van de tekst over blijft. De teksten worden aangepast alvorens de teksten door de auteurherkenningsmethode verwerkt worden. Hierdoor wordt de auteurherkenningsmethode niet verstoord door niet-relevante informatie [15].

Om te kunnen maskeren is een lijst van minstvoorkomende woorden nodig. Men kan een lijst gebruiken van de minstvoorkomende van een taal (deze is dan uit een grote taalcorpus geconstrueerd) of zelf een lijst van de minstvoorkomende woorden uit de trainingsdata maken. Beide maskeringtechnieken worden in dit onderzoek voor de verschillende datasets onderzocht.

W_k is de lijst van k meestvoorkomende woorden van een taal of dataset [15].

Dan zijn er verschillende manieren om woorden te maskeren [15]:

1. DV-MA (‘Distorted View – Multiple Asterisks’): elk woord dat niet in W_k zit wordt gemaskeerd door elke letter in dat woord te vervangen door een asterisk (*). Elk cijfer wordt gemaskeerd door een hashtag (#).
2. DV-SA (‘Distorted View – Single Asterisk’): elk woord dat niet in W_k zit wordt gemaskeerd door het woord te vervangen door een enkele asterisk (*). Elk cijfer wordt gemaskeerd door een hashtag (#). Deze techniek past dus de lengte van de tekst aan.

3. DV-EX (‘Distorted View – Exterior Characters’): elk woord dat niet in W_k zit wordt gemaskeerd door alle letters behalve de eerste en de laatste letter van het woord te vervangen door een asterisk (*). De buitenste karakters van een woord blijven dus intact. Elk cijfer wordt gemaskeerd door een hashtag (#). Op deze manier blijven bepaalde stijlfiguren zoals alliteraties wel zichtbaar.
4. DV-L2: (‘Distorted View – Last 2 Characters): elk woord dat niet in W_k zit wordt gemaskeerd door alle letters behalve de laatste twee van het woord te vervangen door een asterisk (*). Elk cijfer wordt gemaskeerd door een hashtag (#). Hierdoor blijft bijvoorbeeld rijm in de tekst zichtbaar.

In dit onderzoek wordt enkel de DV-SA manier gebruikt om te maskeren (en cijfers worden in dit onderzoek op dezelfde manier gemaskeerd als woorden). Hiervoor is gekozen omdat in het onderzoek van Hajer te zien is dat er geen significant verschil in de prestaties is te zien tussen DV-SA en DV-MA [6]. Rekeninghoudend met de rekenefficiëntie is daarom gekozen voor DV-SA. DV-EX en DV-L2 zijn ingewikkeldere maskeringstechnieken en zijn in het kader van de tijd van dit onderzoek buiten beschouwing gelaten maar zouden goed gebruikt kunnen worden voor vervolgonderzoek.

De parameter k is een belangrijke parameter bij het maskeren van onderwerp- en genregerelateerde informatie. Hoe groter k, hoe meer onderwerpgerelateerde informatie zichtbaar is in de tekst. Hoe kleiner k, hoe meer onderwerpgerelateerde informatie gemaskeerd is. Deze parameter k wordt geanalyseerd om te kijken bij hoeveel te maskeren woorden, bij welke k, de auteurherkenningsmethode het beste werkt [15].

3

Datasets voorbereiden

Voor het gebruik van SVMs en logistische regressie op de datasets moeten de datasets voorbereid ofwel 'schoongemaakt' worden. Voor iedere dataset moet er specifiek gekeken worden wat er aangepast of gefilterd moet worden voordat de data gebruikt kan worden.

Allereerst is het goed om bewust te zijn van eventuele spelfouten. Zowel in tweets als in gepubliceerde teksten in boeken of artikelen kunnen spelfouten voorkomen. In sommige soorten teksten komen spelfouten meer voor dan in andere soorten teksten maar dit probleem speelt in alle datasets.

Verkeerd gespelde woorden zijn een probleem omdat dan woorden die eigenlijk niet gemaskeerd zouden moeten worden toch gemaskeerd worden, omdat ze verkeerd gespeld zijn. Ze horen eigenlijk thuis in de lijst van veelvoorkomende woorden en bevatten weinig onderwerpafhankelijke informatie, maar worden niet zo gedetecteerd. Aan de andere kant kunnen consistente spelfouten ook juist een aanwijzing naar een specifieke auteur zijn [6].

Een mogelijke optie die met deze aspecten rekening houdt is de volgende: eerst worden in een dataset alle spelfouten gesignaleerd en verbeterd, en daarna wordt deze verbeterde tekst op een juiste manier gemaskeerd met behulp van de lijst van meest voorkomende woorden. Hierdoor blijven verkeerd gespelde versies van veelvoorkomende woorden wel in de tekst staan.

Deze manier om om te gaan met spelfouten wordt in dit onderzoek toegepast op alle datasets.

3.1. Tweets dataset

In de voorbereiding van de Tweets dataset moet rekening gehouden worden met een aantal details. Er moeten keuzes gemaakt worden .

Allereerst zijn sommige tweets enkel een 'retweet' van een ander persoon. Een tweet van Biden die wordt geretweet door Barack Obama wordt dan aan Obama toegeschreven terwijl deze eigenlijk door Biden is geschreven. Daarom worden alle tweets die een retweet (RT) bevatten uit de dataset gefilterd en niet meegenomen in de auteurherkenning.

Daarnaast bestaan sommige tweets uit een reactie op een tweet van een ander persoon. Dit kan herkend worden aan de dubbele aanhalingstekens (''). De tekst tussen de dubbele aanhalingstekens is tekst van een andere persoon, en daarna wordt er door de auteur van de tweet op gereageerd. De tekst tussen de aanhalingstekens is dus niet geschreven door de auteur van de tweet. Daarom wordt alle tekst tussen dubbele aanhalingstekens binnen de tweets eruit gehaald voordat de tweets gebruikt worden.

Verder bevatten sommige tweets URL-links. Deze links kunnen herkend worden aan het begin met

'http'. Omdat de links niet relevant zijn in het kader van auteurherkenning, worden binnen de tweets ook de URL-links eruit gehaald voor gebruik.

In tweets worden ook vaak hashtags gebruikt. In de hashtags staan vaak woorden die te maken hebben met het onderwerp van de tweet. Dat maakt de informatie in de hashtags niet zo relevant voor de auteurherkenning. Daarom worden de hashtags uit de tweets verwijderd.

Ook bevatten tweets vaak emoji's. Emoji's geven in sommige gevallen emoties weer, en in andere gevallen bijvoorbeeld het onderwerp van de tweet. Het gebruik van emoji's kan wel degelijk iets zeggen over de schrijfstijl van een persoon. Voor het maskeren is er een lijst met de meestvoorkomende emoji's te vinden. Maar het is lastig om in te schatten hoe vaak emoji's in vergelijking tot bepaalde woorden gebruikt worden. Dus waar ze in de lijst komen te staan van woorden die wel of niet gemaskeerd moeten worden. Omdat dit niet goed te bepalen is, is er voor gekomen om de emoji's uit de tweets te halen alvorens SVM en logistische regressie toegepast worden.

Nadat de bovenstaande aanpassingen zijn gedaan aan de tekst wordt de spelling van de woorden in de tweets gecontroleerd en aangepast. Hiervoor gebruikt men in Python de functie 'SpellChecker'. Deze functie maakt gebruik van de Levenshteinafstand: het minimale aantal bewerkingen dat nodig is om de ene string in de andere te veranderen. Bij ieder woord wordt gekeken naar permutaties binnen een Levenshteinafstand van 2 van het oorspronkelijke woord. Vervolgens vergelijkt de functie alle permutaties met bekende woorden in een woordfrequentielijst, om op basis daarvan te bepalen of het woord goed gespeld is of dat een ander woord waarschijnlijker is [2].

3.2. Victoriaanse dataset

De Victoriaanse dataset bestaat, in tegenstelling tot de Tweets dataset, uit gepubliceerde teksten. Gepubliceerde teksten zijn zorgvuldiger geschreven en gecheckt dan tweets, die vaak 'even' snel geschreven en verstuurd worden. Hierdoor zitten er waarschijnlijk minder spelfouten in de Victoriaanse dataset.

Toch wordt ook van de woorden in deze teksten de spelling gecontroleerd voordat er SVM en logistische regressie toegepast wordt. Ook hier wordt de functie SpellChecker voor gebruikt, die de huidige taal als referentie gebruikt. Het is goed om bewust te zijn van het feit dat de Victoriaanse dataset verouderd taalgebruik kan bevatten, omdat de teksten uit de 19de eeuw komen. In plaats van SpellChecker zou er een spellingschecker specifiek voor deze dataset gemaakt kunnen worden met een corpus van teksten en woorden uit de 19de eeuw. Maar omdat dit veel werk is en niet het doel van dit onderzoek is besloten om SpellChecker te gebruiken. Er is ook gekeken hoeveel de teksten na het gebruik van SpellChecker verschilden van de oorspronkelijke teksten en dit bleek niet zo heel veel te zijn.

Naast het toepassen van SpellChecker worden er verder geen andere aanpassingen aan deze dataset gedaan.

3.3. Federalist Papers dataset

De Federalist Papers dataset bestaat, net als de Victoriaanse dataset, uit gepubliceerde teksten. Het zijn artikelen die gepubliceerd werden in vijf New Yorkse kranten. Hierdoor zullen er, om dezelfde reden als bij de Victoriaanse dataset, waarschijnlijk weinig spelfouten in zitten.

Toch wordt ook hier de spelling gecontroleerd voordat SVM en logistische regressie worden toegepast. Dit wordt wederom gedaan met behulp van SpellChecker in Python. Omdat de Federalist Papers uit 1787 en 1788 komen is het ook goed om hier bewust te zijn van verouderd taalgebruik. Maar om dezelfde redenen als bij de Victoriaanse dataset wordt SpellChecker ook hier gebruikt.

Ook hier worden er naast SpellChecker geen andere aanpassingen gedaan.

4

Maskeren

Nadat de datasets zijn voorbereid, kan het maskeren en daarna het toepassen van SVM en logistische regressie beginnen. In dit onderzoek worden twee verschillende manieren van maskeren onderzocht.

Allereerst kan er gemaskeerd worden door een lijst van de meest voorkomende woorden uit de hele Engelse taal te gebruiken. Dit wordt gedaan met een lijst die gebaseerd is op ongeveer één biljoen woorden uit de Corpus of Contemporary American English (COCA). Dit is een grote Engelse, up-to-date corpus die verschillende genres bevat. In totaal bevat de lijst 60.000 woorden, maar de gratis, openbaar beschikbare lijst bevat 5050 woorden. Deze lijst van 5050 woorden is gebruikt om de drie datasets op deze manier te maskeren [19].

Daarnaast kan er ook met een aangepaste frequentielijst gemaskeerd worden. Per dataset wordt er een frequentielijst gemaakt van de meestvoorkomende woorden in die dataset specifiek. Met behulp van deze datasetspecifieke frequentielijst worden de datasets ook gemaskeerd.

Deze twee manieren van maskeren worden allebei toegepast op alle drie de datasets om te kijken welke manier bij welke datasets de beste resultaten geeft.

5

Prestaties algemeen

Om uitspraken te kunnen doen over wat de beste manier van maskeren is voor de verschillende datasets, wordt er onder andere gekeken naar de prestaties. In de literatuur komt het vaak voor dat bij het gebruik van SVM en logistische regressie gekeken wordt naar de precisie, recall, accuraatheid en F-score van het model om waarde aan de prestaties te geven [6].
In dit onderzoek zal daarom naar deze gegevens worden gekeken om de verschillende manieren van maskeren en de verschillende classificatiemethoden te vergelijken.

5.1. Precisie

Laat $A = \{a_1, a_2, \ldots, a_n\}$ de set zijn van alle n auteurs die teksten hebben geschreven.
Om SVM of logistische regressie te kunnen toepassen moet er een trainingsdataset T en een validatiedataset V zijn. De validatiedataset bevat de 'onbekende' teksten die geclassificeerd worden [6].
Het model dat gebruikt wordt voor classificeren wordt m genoemd. M is dan een n keer n matrix. In M geeft element M_{ij} aantal teksten in V weer wat geschreven is door auteur a_i en is toegewezen aan a_j door model m [6].

Voor elke auteur a_i waarbij $i \in \{1, 2, \ldots, n\}$ is de precisie van m als volgt gedefinieerd:
de **precisie** is de verhouding van alle teksten in V met als juiste auteur a_i tegenover alle teksten in V waarvan a_i daadwerkelijk de auteur is [6]. Ofwel:

$$precisie(m, a_i, V) = \frac{M_{ii}}{\sum_{j=1}^{n} M_{ij}} \text{[6]}$$

5.2. Recall

Voor alle auteurs a_i waarbij $i \in \{1, 2, \ldots, n\}$ kan de gevoeligheid, ook wel 'recall' genoemd, van model m als volgt worden beschreven:
de **recall** is de verhouding van alle teksten in V met als juiste auteur a_i tegenover alle teksten in V die (goed én fout) zijn toegewezen aan auteur a_i [6]. Ofwel:

$$recall(m, a_i, V) = \frac{M_{ii}}{\sum_{j=1}^{n} M_{ji}} \text{[6]}$$

5.3. Micro- en macrogemiddelde

Het is relevant om te kijken naar het micro- en macrogemiddelde van de precisie en de recall (gevoeligheid) omdat die iets zeggen over het hele model en niet alleen over het geval van een bepaalde auteur.
Bij het berekenen van het microgemiddelde wordt elke tekst in V even zwaar meegenomen in de weging. Dat ziet er als volgt uit:

$precisie_{micro}(m,V) = \frac{\sum_{i=1}^{n} M_{ii}}{\sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}}$[6]

$recall_{micro}(m,V) = \frac{\sum_{i=1}^{n} Mii}{\sum_{i=1}^{n}\sum_{j=1}^{n} Mji}$[6]

Er is hierboven te zien dat $precisie_{micro}(m,V) = recall_{micro}(m,V)$.
Het microgemiddelde van de precisie wordt ook wel de **micro-accuraatheid** of kortweg **accuraatheid** van het model m genoemd [6].
Er kan een probleem optreden bij het gebruik van de microgemiddeldes van precisie en recall om de kwaliteit van het model te beoordelen. Dat probleem kan ontstaan als de verschillende klassen met teksten van ongelijke grootte zijn. In de berekening van de microgemiddeldes wegen alle teksten even zwaar waardoor sommige klassen veel zwaarder in de berekening wegen dan andere klassen. Er kan dus een grote ongelijkheid tussen de zwaartes van de klassen ontstaan waardoor de microgemiddeldes een onbetrouwbaar beeld geven van het model.

Er is ook nog een andere manier om de gemiddeldes van precisie en recall te berekenen: het **macro-gemiddelde**. Bij de berekening van het macrogemiddelde wordt iedere klasse in V even zwaar meegenomen in de berekening. Dat ziet er als volgt uit:

$precisie_{macro}(m,V) = \frac{1}{n}\sum_{i=1}^{n} precisie(m,a_i,V) = \frac{1}{n}\sum_{i=1}^{n} \frac{M_{ii}}{\sum_{j=1}^{n} M_{ij}}$[6]

$recall_{macro}(m,V) = \frac{1}{n}\sum_{i=1}^{n} recall(m,a_i,V) = \frac{1}{n}\sum_{i=1}^{n} \frac{M_{ii}}{\sum_{j=1}^{n} M_{ji}}$[6]

Het macrogemiddelde van de precisie wordt ook wel de **macro-accuraatheid** van het model m genoemd [6].

De macrogemiddeldes laten alle klassen even zwaar wegen in de berekening. Bij een model met klassen van zeer ongelijke grootte zijn deze waarden betrouwbaarder dan de microgemiddeldes om te gebruiken bij het formuleren van een uitspraak over de kwaliteit van het model.

5.4. F-score

Naast de voorgaande gegevens kan er ook naar de F_β-score van het model worden gekeken. De F_β-score is een gewogen gemiddelde van de precisie en recall:

$F_\beta(m,V) = (1+\beta^2)\frac{precisie(m,V)\cdot recall(m,V)}{\beta^2\cdot precisie(m,V)+recall(m,V)}$[6]

Er kan ook gekeken worden naar de microgemiddelde of macrogemiddelde F_β-score. Deze kan op twee verschillende manieren gedefinieerd worden. Er kan in de berekening van de F_β-score gebruik worden gemaakt van de microgemiddeldes of macrogemiddeldes van de precisie en recall. Of er kan er gekeken worden naar het microgemiddelde of macrogemiddelde van de verschillende F_β-scores voor de verschillende klassen. Dit levert verschillende resultaten op. Als er in dit onderzoek gesproken wordt over de micro of macro F_β-score dan heeft men het over de tweede definitie. Er wordt namelijk gebruik gemaakt van de scikit-learn bibliotheek in Python en die houdt ook deze tweede definitie aan.

Het is specifiek interessant om te kijken naar de **macro F_1-score**. Door $\beta = 1$ te nemen is de waarde van de precisie even belangrijk als de waarde van de recall in de berekening:

$F_{1,macro}(m,V) = \frac{1}{n}\sum_{i=1}^{n} F_1(m,a_i,V) = \frac{2}{n}\sum_{i=1}^{n} \frac{precisie(m,a_i,V)\cdot recall(m,a_i,V)}{precisie(m,a_i,V)+recall(m,a_i,V)}$[6]

De macro F_1-score, ofwel kortweg de F_1-score is een goede waarde om te gebruiken bij het kijken naar de betrouwbaarheid van het model m. De F_1-score bevat namelijk zowel de informatie van de precisie als de recall en neemt deze beiden mee in de berekening. Omdat de F_1-score het macrogemiddelde van de F_1-scores van de verschillende klassen gebruikt is deze ook zeer geschikt in ongebalanceerde datasets waarbij de klassen niet even groot zijn.

6

Prestaties van de datasets

Van de datasets in dit onderzoek zijn de Victoriaanse dataset en de Federalist Papers dataset ongebalanceerd: sommige klassen in deze datasets zijn veel groter dan andere klassen. De Tweets dataset bevat wel allemaal even grote klassen. Omdat men te maken heeft met ongebalanceerde datasets is ervoor gekozen om voor de prestaties naar $precisie_{macro}$, $recall_{macro}$ en de F_1-scores van de modellen in de verschillende datasets te kijken en deze met elkaar te vergelijken.

Zoals in hoofdstuk 4 te lezen is, worden alle datasets op twee verschillende manieren gemaskeerd: maskering met behulp van de COCA-woordenlijst (kortweg COCA-maskering) en maskering met behulp van een eigen woordenlijst (kortweg eigen maskering).

6.1. Prestaties bij support vector machines

6.1.1. Tweets dataset

Eerst zal er gekeken worden naar de Tweets dataset. In 6.1, 6.2 en 6.3 zijn de grafieken bij COCA-maskering te vinden van de F_1-scores, $precisie_{macro}$ en $recall_{macro}$ uitgezet tegen k. Onthoud dat k is in W_k de lijst van k meest voorkomende woorden. In alle figuren is ook een rode referentielijn weergegeven. Dit is de waarde van de F_1-scores, $precisie_{macro}$ en $recall_{macro}$ als er SVM toegepast wordt op ongemaskeerde teksten.

Figure 6.1: Tweets dataset COCA-maskering F1-score uitgezet tegen k

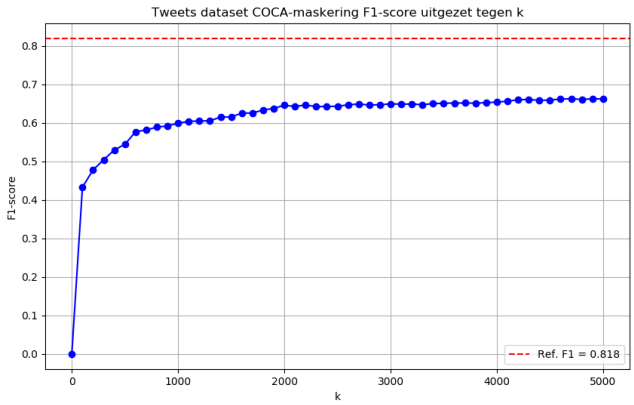


Figure 6.2: Tweets dataset COCA-maskering macroprecisie uitgezet tegen k

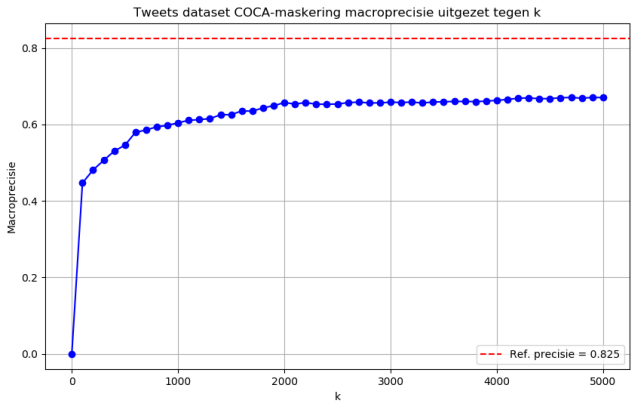
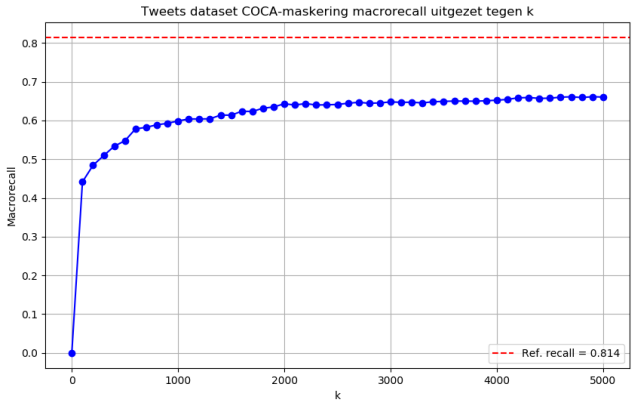


Figure 6.3: Tweets dataset COCA-maskering macrorecall uitgezet tegen k



In alle bovenstaande figuren (6.1, 6.2, 6.3) is te zien dat de prestaties van het gebruik van COCA-maskering niet boven de referentielijnen van zonder maskering komt te liggen. Dit is logisch en te verklaren door het feit dat er bij het maskeren informatie wordt weggehaald waardoor de SVM minder goed werkt en de prestaties lager uitvallen.

Verder is er in de grafieken ook te zien dat van $k = 0$ tot $k = 100$ de meeste winst wordt behaald voor de waarden van F_1-score, $precisie_{macro}$ en $recall_{macro}$. Daar stijgt de grafiek het meest. De prestaties worden daar zo sterk beïnvloed omdat tussen $k = 0$ en $k = 100$ heel veel woorden gemaskeerd worden. Daarna vlakt in alle figuren de grafiek wat meer af. Waarna er te zien is dat er vanaf $k = 3000$ een stabilisatie plaatsvindt. Dat is te verklaren doordat er tussen $k = 3000$ en $k = 5000$ weinig woorden extra worden gemaskeerd in de tweets omdat die woorden maar heel weinig in de tweets voorkomen. Tweets bevatten veel veelvoorkomende woorden en weinig weinigvoorkomende woorden. Daarom is er tussen $k = 3000$ en $k = 5000$ weinig verschil in de prestaties meer te zien.

De waarden voor eigen maskering bij de Tweets dataset worden in de onderstaande figuren 6.4, 6.5 en 6.6 weergegeven.

Figure 6.4: Tweets dataset eigen maskering F1-score uitgezet tegen k

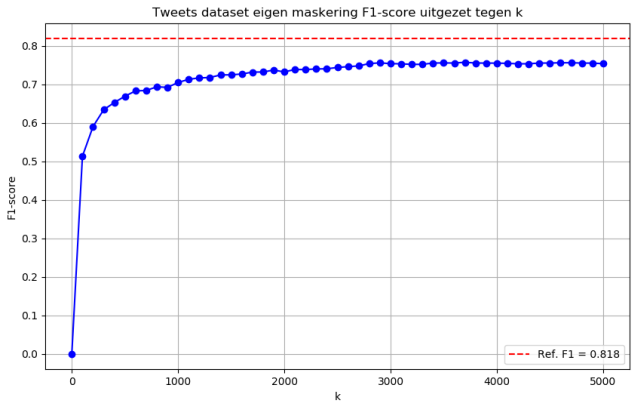


Figure 6.5: Tweets dataset eigen maskering macroprecisie uitgezet tegen k

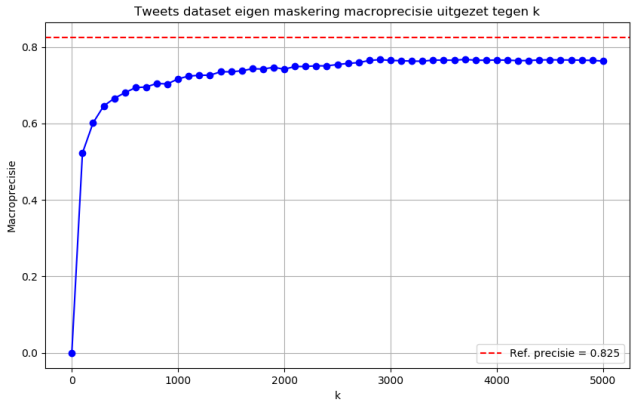
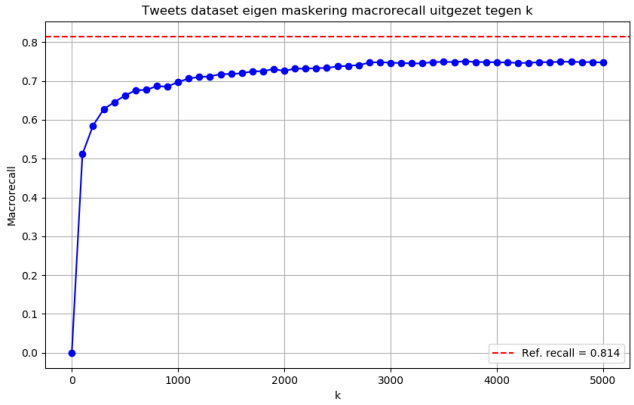


Figure 6.6: Tweets dataset eigen maskering macrorecall uitgezet tegen k



Net als bij COCA-maskering (in figuren 6.1, 6.2 en 6.3) is bij eigen maskering (in bovenstaande figuren 6.4, 6.5 en 6.6) te zien dat de prestaties niet boven de referentielijn van geen maskering komen te liggen. Dit is op dezelfde manier te verklaren als bij de COCA-maskering. Verder is ook in deze figuren te zien dat van $k = 0$ tot $k = 100$ de meeste winst wordt behaald, door een grote stijging van de grafiek.

Daarna vlakt de grafiek af waarna er een stabilisatie plaatsvindt. Deze stabilisatie vindt al eerder plaats dan bij de COCA-maskering, namelijk bij $k = 2000$. Deze stabilisatie is verklaarbaar met dezelfde redden als bij de COCA-maskering. Het is ook logisch dat deze stabilisatie eerder plaatsvindt dan bij de COCA-maskering omdat hier een eigen frequentielijst is gebruikt bij maskeren en er na $k = 2000$ dus al bijna geen extra woorden worden gemaskeerd.

Verder liggen de grafieken voor eigen maskering voor zowel F_1-score als voor de macroprecisie en macrorecall boven de grafieken van COCA-maskering bij de Tweets dataset. Eigen maskering levert bij SVM dus betere prestaties op dan COCA-maskering voor de Tweets dataset.

6.1.2. Victoriaanse dataset

De prestaties van SVM voor de Victoriaanse dataset worden in deze subsectie weergegeven. Allereerst zijn de prestaties van COCA-maskering te zien in figuren 6.7, 6.8 en 6.9.

Figure 6.7: Victoriaanse dataset COCA-maskering F1-score uitgezet tegen k

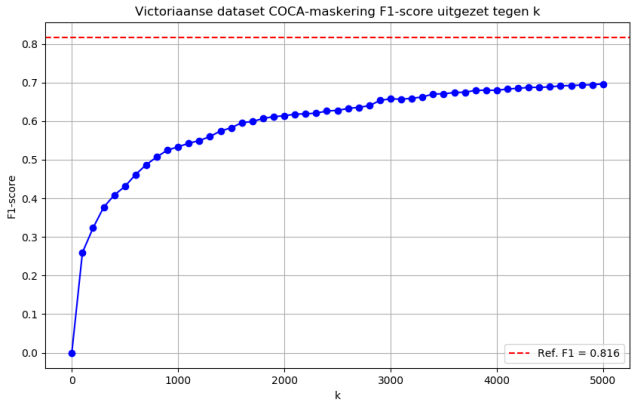


Figure 6.8: Victoriaanse dataset COCA-maskering macroprecisie uitgezet tegen k

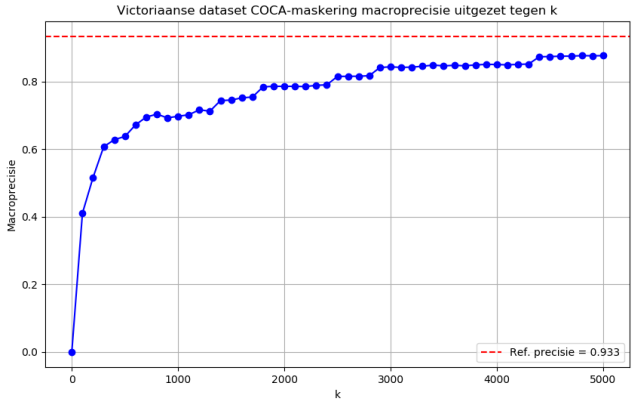
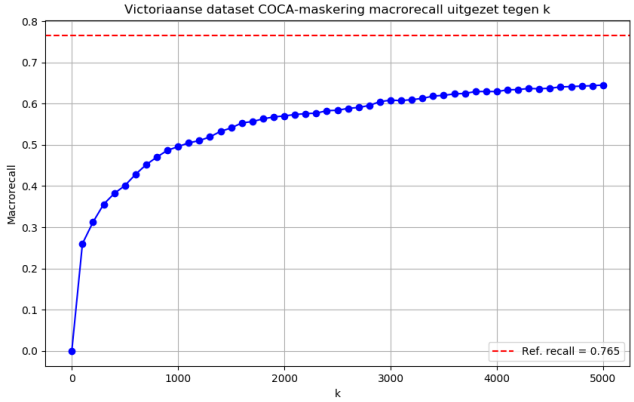


Figure 6.9: Victoriaanse dataset COCA-maskering macrorecall uitgezet tegen k



Er is in deze figuren te zien dat de prestaties niet boven de referentielijnen uitkomen. De grafieken in figuur 6.7 en 6.9 hebben een soortgelijk verloop. De stijging van de grafiek van $k = 0$ tot $k = 100$ is minder steil dan bij de Tweets dataset, en de grafiek blijft tot $k = 5000$ stijgen. Er vindt geen stabilisatie plaats. Deze twee aspecten zijn te verklaren doordat Victoriaanse teksten minder veelvoorkomende woorden bevatten dan tweets. De verscheidenheid aan woorden in deze teksten is groter dan bij de tweets. Hierdoor is het aantal gemaskeerde woorden tussen $k = 0$ en $k = 100$ groter dan bij de Tweets dataset en daarom nemen de prestaties minder snel toe. Dat is ook de reden waarom er niet echt een stabilisatie plaatsvindt: de Victoriaanse dataset bevat veel weinigvoorkomende woorden dus ook nog vanaf bijvoorbeeld $k = 3000$ zie je dat er nog gemaskeerde woorden zijn. Daarvan neemt het aantal tot $k = 5000$ nog af waardoor de prestaties blijven toenemen.

Verder lijkt het verloop van figuur 6.8 op dat van figuren 6.7 en 6.9, alleen is er een meer trapsgewijs verloop te zien. Dit verloop is te verklaren door te kijken naar hoe de (macro)precisie gedefinieerd is (zie 5.1). De precisie is de verhouding van alle teksten in V met juiste auteur a_i tegenover alle teksten in V die (goed én fout) zijn toegewezen aan auteur a_i. Het trapsgewijze verloop is te verklaren doordat er misschien niet zoveel teksten van bepaalde auteurs beschikbaar in V zijn. Dit lijkt tegenstrijdig omdat de Victoriaanse dataset een hele grote dataset is en van alle vijftig auteurs ongeveer duizend teksten beschikbaar heeft. Maar, zoals te lezen is in paragraaf 2.4, wordt er in dit onderzoek maar een kwart van de hele dataset gebruikt.

De prestaties van maskering met eigen frequentielijst bij de Victoriaanse dataset zijn te zien in figuren 6.10, 6.11 en 6.12.

Figure 6.10: Victoriaanse dataset eigen maskering F1-score uitgezet tegen k

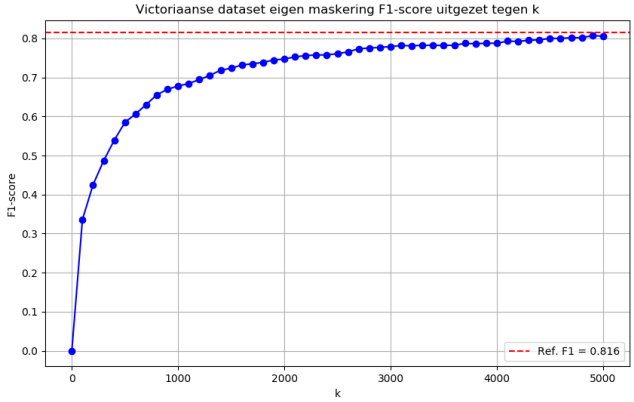


Figure 6.11: Victoriaanse dataset eigen maskering macroprecisie uitgezet tegen k

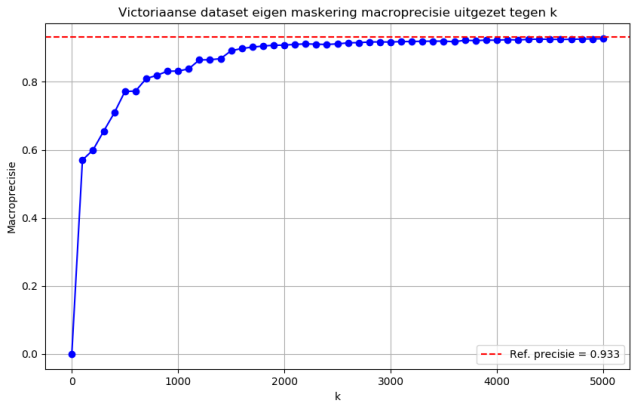
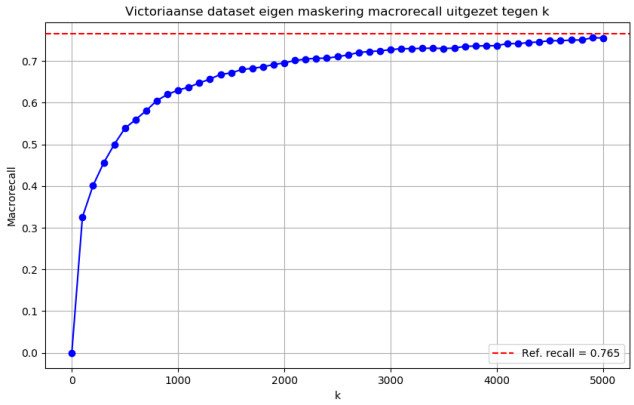


Figure 6.12: Victoriaanse dataset eigen maskering macrorecall uitgezet tegen k



De grafieken in de bovenstaande figuren hebben een soortgelijk verloop als bij de COCA-maskering maar de eindwaarden op $k = 5000$ komen, in tegenstelling tot bij de COCA-maskering wel dichtbij of op de referentielijn te liggen. De prestaties van het gebruik van eigen frequentielijst bij de maskering bij SVM zijn dus beter dan bij COCA-maskering voor de Victoriaanse dataset.

6.1.3. Federalist Papers dataset

In deze subsectie zijn de prestaties van SVM voor de Federalist Papers dataset te zien. Allereerst zijn de prestaties van de COCA-maskering te zien in figuren 6.13, 6.14 en 6.15.

Figure 6.13: Federalist Papers dataset COCA-maskering F1-score uitgezet tegen k

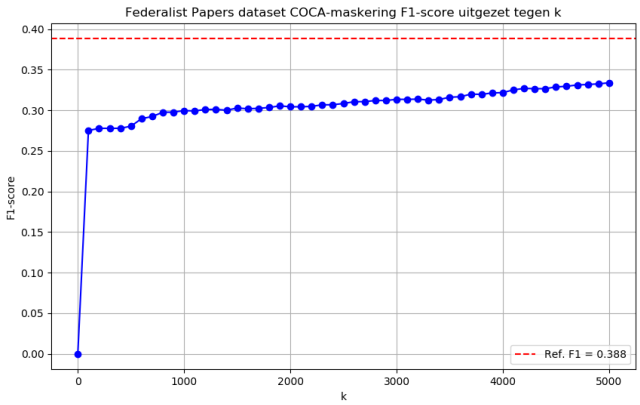


Figure 6.14: Federalist Papers dataset COCA-maskering macroprecisie uitgezet tegen k

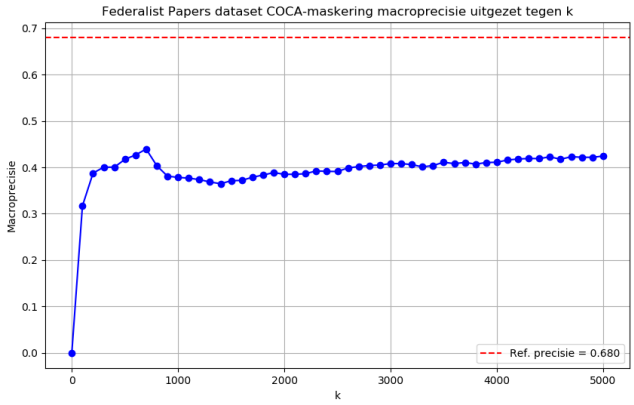
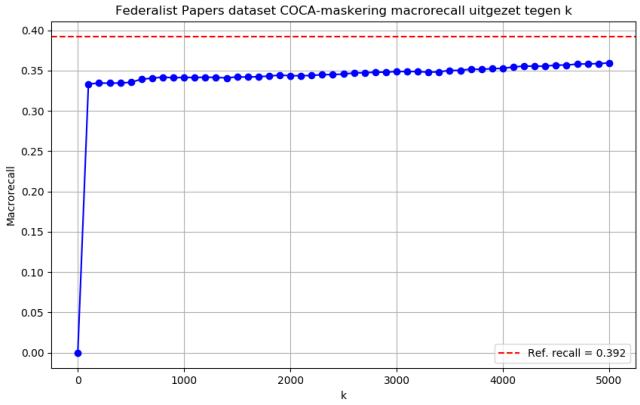


Figure 6.15: Federalist Papers dataset COCA-maskering macrorecall uitgezet tegen k



Allereerst is het in bovenstaande grafieken opvallend dat de referentiewaarden van de F_1-score, $precisie_{macro}$ en $recall_{macro}$ bij de Federalist Papers dataset een stuk lager liggen dan bij de Tweets dataset en bij de Victoriaanse dataset. Dit betekent dat dit een moeilijk te classificeren dataset is. Dat zou kunnen komen doordat de Federalist Papers brieven zijn in een reeks over hetzelfde onderwerp. De auteurs

hebben wellicht geprobeerd om niet al te veel verschil in schrijfstijl tussen de verschillende brieven te laten zien zodat de reeks een eenheid vormt en de lezers niet doorhebben dat deze door verschillende auteurs is geschreven. Een andere verklaring voor het feit dat de teksten moeilijk te classificeren zijn zou kunnen zijn dat het komt doordat de kranten de teksten geëditeerd hebben.

De prestaties komen in alle grafieken niet boven de referentielijnen uit. Er is ook te zien dat tussen $k = 100$ en $k = 5000$ maar een lichte stijging in de waarde van de prestaties is. Er vind geen stabilisatie plaats maar er is sprake van een continue lichte steiging. De stijging is erg klein. Dit zou kunnen komen doordat de gebruikte woordenlijst, de COCA-woordenlijst, niet goed past bij de Federalist Papers dataset. Dat is logisch te verklaren omdat de Federalist Papers dataset weinigvoorkomend en ook eventueel verouderd taalgebruik bevat, en de COCA-woordenlijst is een lijst met veelgebruikte, hedendaagse woorden.

Verder is er in figuur 6.14 een opvallend maximum te zien bij $k = 700$, wat geen maximum is in de andere grafieken. Hier kon geen verklaring voor worden gevonden.
Bij $k = 700$ bereikt de $precisie_{macro}$ een hoogtepunt. Maar de F_1 en $recall_{macro}$ bereiken geen hoogtepunt bij $k = 700$. Als men naar al deze waarden kijkt, kan toch $k = 5000$ als de waarde met de beste prestaties genomen worden, omdat de waarde van het maximum van de $precisie_{macro}$ bij $k = 700$ maar 0.01 of 0.02 verschilt van $k = 5000$ en bij $k = 5000$ zijn de F_1-score en $recall_{macro}$ wel significant hoger.

De prestaties van eigen maskering toegepast zijn te zien in de figuren 6.16, 6.17 en 6.18.

Figure 6.16: Federalist Papers dataset eigen maskering F1-score uitgezet tegen k

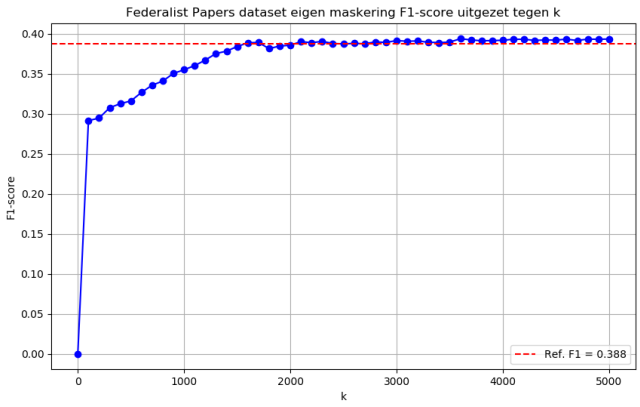


Figure 6.17: Federalist Papers dataset eigen maskering macroprecisie uitgezet tegen k

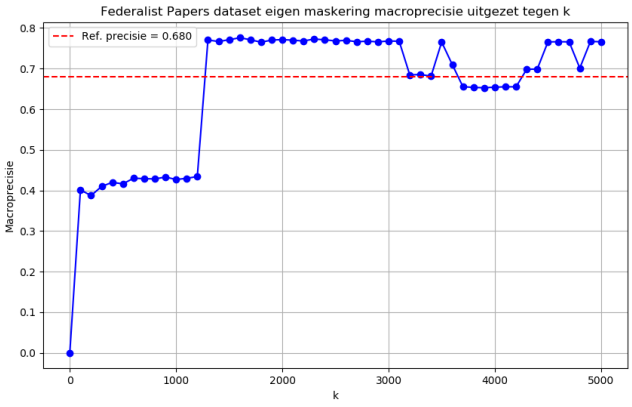
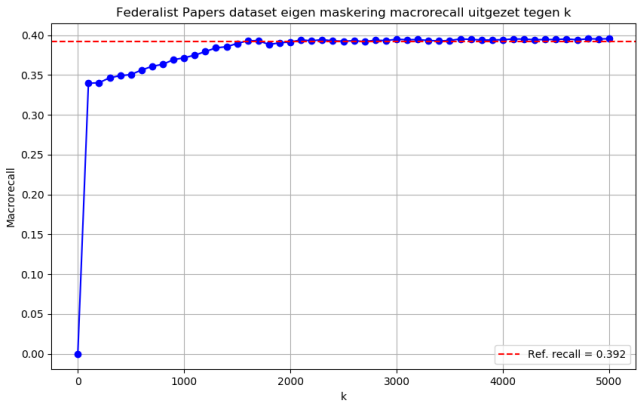


Figure 6.18: Federalist Papers eigen maskering macrorecall uitgezet tegen k



Wat als eerste opvalt in bovenstaande figuren is dat er waarden zijn waarvoor de prestaties beter zijn dan de referentielijn. Vanaf ongeveer $k = 1600$ zijn de prestaties beter of gelijk aan de referentielijn. Dit betekent dat er met eigen maskering bij de Federalist Papers dataset veel gemaskeerd kan worden zonder dat er qua prestaties veel verloren gaat.

Verder is er vanaf $k = 100$ tot $k = 1600$ een stijging te zien waarna de grafiek vanaf $k = 1600$ stabiliseert.

Het verloop van figuur 6.17 is erg opvallend vergeleken met figuren 6.16 en 6.18. Er is een trapsgewijs verloop te zien. Dit is op dezelfde manier te verklaren als voor figuur 6.8. De Federalist Papers dataset is een kleine dataset dus is het trapsgewijze verloop te verklaren doordat er niet zoveel teksten van bepaalde auteurs beschikbaar zijn. Verder schommelt de grafiek ook heel erg. Dit is ook opvallend. De grafiek komt voor sommige waarden ver boven de referentielijn uit en voor andere waarden weer eronder. Deze schommelingen kunnen dezelfde oorzaak hebben als het trapsgewijze verloop.

Verder liggen de grafieken van eigen maskering bij de Federalist Papers dataset boven de grafieken van COCA-maskering. Er kan daarom gezegd worden dat voor de Federalist Papers dataset eigen maskering bij SVM betere prestaties oplevert dan COCA-maskering.

6.2. Prestaties logistische regressie

In deze paragraaf worden de prestaties van het gebruik van logistische regressie als model voor auteurherkenning voor de verschillende datasets besproken en worden deze vergeleken met de prestaties van SVM uit de vorige sectie 6.1.

In de grafieken zijn, net als in de vorige sectie, de F_1-scores, $precisie_{macro}$ en $recall_{macro}$ uitgezet tegen parameter k bij zowel de COCA-maskering als bij maskering met behulp van een eigen frequentielijst (eigen maskering). Ook is er in alle figuren een roze referentielijn te zien. Dit is de waarde van de F_1-score, $precisie_{macro}$ en $recall_{macro}$ als er logistische regressie wordt toegepast op ongemaskeerde teksten in de datasets.

6.2.1. Tweets dataset

De prestaties bij logistische regressie voor de Tweets dataset worden in deze subsectie weergegeven. Allereerst zijn de waarden voor COCA-maskering in de figuren 6.19, 6.20 en 6.21 weergegeven.

Figure 6.19: Tweets dataset COCA-maskering F1-score uitgezet tegen k (logistische regressie)

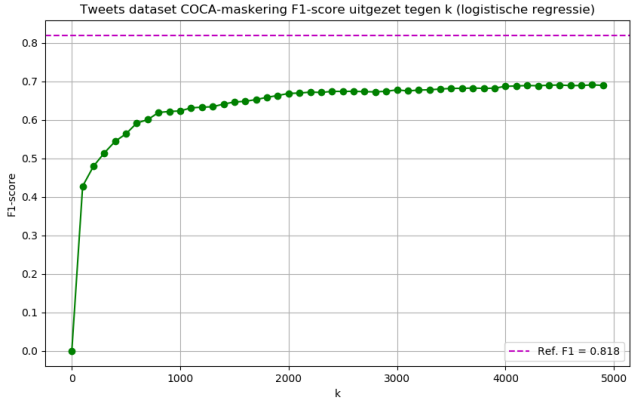


Figure 6.20: Tweets dataset COCA-maskering macroprecisie uitgezet tegen k (logistische regressie)

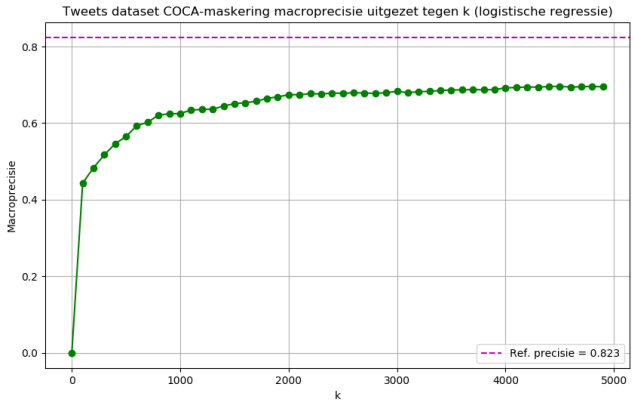
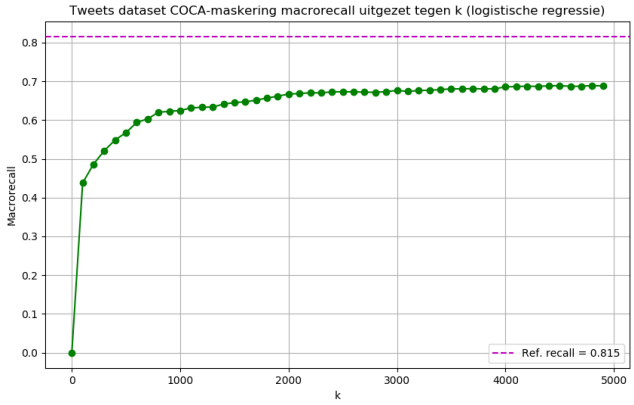


Figure 6.21: Tweets dataset COCA-maskering macrorecall uitgezet tegen k (logistische regressie)



Het verloop van de grafieken in 6.19, 6.20 en 6.21 is zoals het verloop bij diezelfde grafieken voor SVM. Er zijn geen gekke uitschieters en het afvlakkend stijgende verloop is zoals verwacht. De verklaringen voor dit verloop zijn al uitgebreid beschreven in de vorige sectie 6.1.

Wel is te zien dat de grafieken, zeker bij hogere waarden van k, iets boven de waarden van SVM liggen. Hierdoor zijn de maximale waarden, die bereikt worden bij $k = 5000$, bij logistische regressie hoger dan bij SVM. Dit verschil is niet groot maar hierdoor kan gezegd worden dat logistische regressie een lichte voorkeur heeft boven SVM bij COCA-maskering op de Tweets dataset.

De waarden voor eigen maskering bij de Tweets dataset zijn weergegeven in de figuren 6.22, 6.23 en 6.24.

Figure 6.22: Tweets dataset eigen maskering F1-score uitgezet tegen k (logistische regressie)

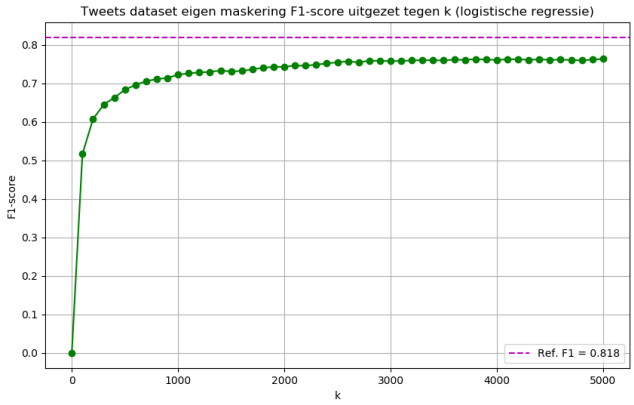


Figure 6.23: Tweets dataset eigen maskering macroprecisie uitgezet tegen k (logistische regressie)

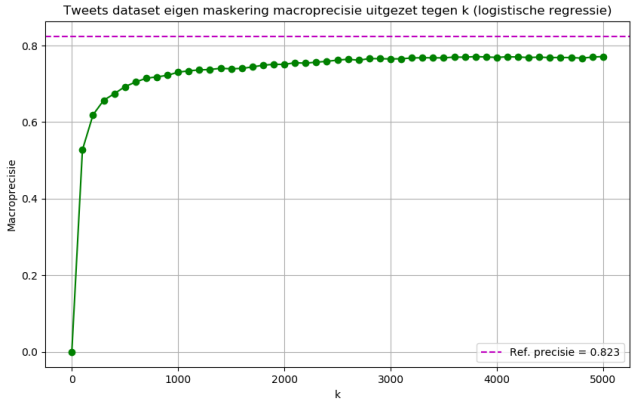
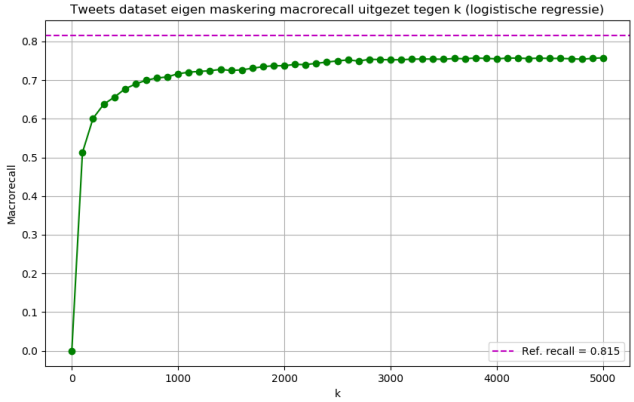


Figure 6.24: Tweets dataset eigen maskering macrorecall uitgezet tegen k (logistische regressie)



Net als bij de grafieken voor COCA-maskering van de Tweets dataset bij logistische regressie (figuren 6.19, 6.20 en 6.21) is het verloop van de grafieken in alle figuren zoals verwacht, en vergelijkbaar met de grafieken in de figuren van SVM. Dit verloop is besproken in sectie 6.1.

Bij eigen maskering is het verschil tussen de grafieken van SVM en logistische regressie minder groot. Ook al is het verschil niet groot, er is wel te zien dat de grafiek van logistische regressie altijd boven of gelijk aan de grafiek van SVM valt. Hieruit en uit de figuren 6.19, 6.20 en 6.21 kan dus gezegd worden dat logistische regressie leidt tot betere prestaties voor de Tweets dataset, zowel bij het gebruik van COCA-maskering als bij eigen maskering. Daarmee is logistische regressie beter geschikt dan SVM voor deze dataset.

Verder is ook te zien dat de prestaties van eigen maskering beter zijn dan bij COCA-maskering voor de Tweets dataset. Eigen maskering werkt dus beter dan COCA-maskering voor de Tweets dataset.

6.2.2. Victoriaanse dataset

De prestaties bij logistische regressie voor de Victoriaanse dataset worden in deze subsectie weergegeven. Allereerst zijn de waarden voor COCA-maskering in figuren 6.25, 6.26 en 6.27 te zien.

Figure 6.25: Victoriaanse dataset COCA-maskering F1-score uitgezet tegen k (logistische regressie)

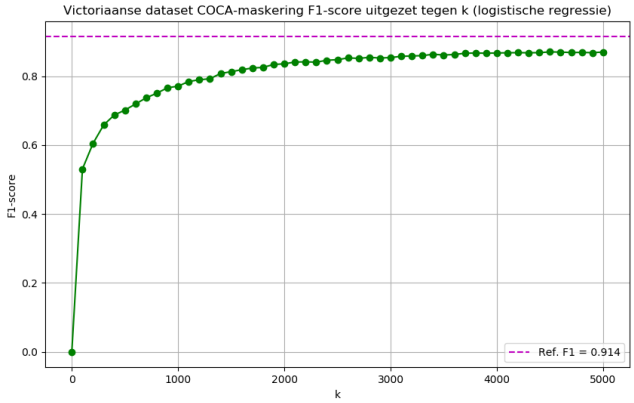


Figure 6.26: Victoriaanse dataset COCA-maskering macroprecisie uitgezet tegen k (logistische regressie)

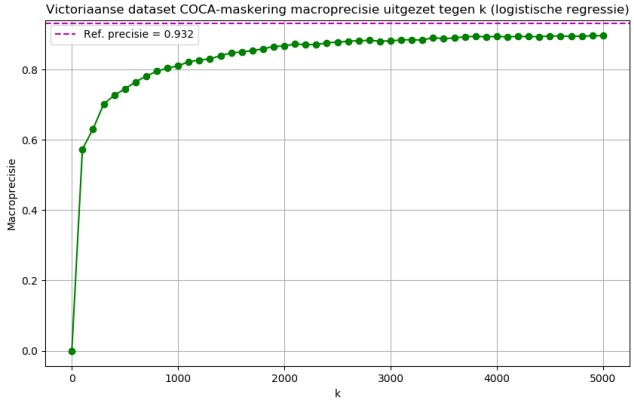
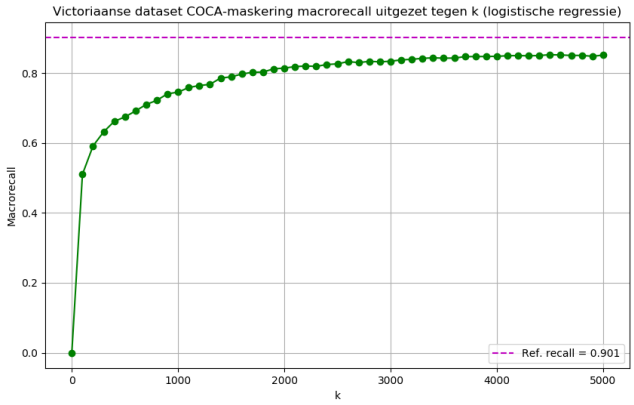


Figure 6.27: Victoriaanse dataset COCA-maskering macrorecall uitgezet tegen k (logistische regressie)



Er is te zien dat de grafieken een afnemend stijgend verloop hebben, vergelijkbaar met de grafieken bij SVM (figuren 6.7, 6.8 en 6.9). De grafiek bij logistische regressie ligt ook voor alle waarden boven die van SVM. Daarnaast is in deze figuren, in tegenstelling tot de figuren van SVM, een stabilisatie rond de referentielijn te zien vanaf $k = 3000$. In de figuren van SVM is geen stabilisatie te zien.

De prestaties van eigen maskering zijn weergegeven in figuren 6.28, 6.29 en 6.30.

Figure 6.28: Victoriaanse dataset eigen maskering F1-score uitgezet tegen k (logistische regressie)

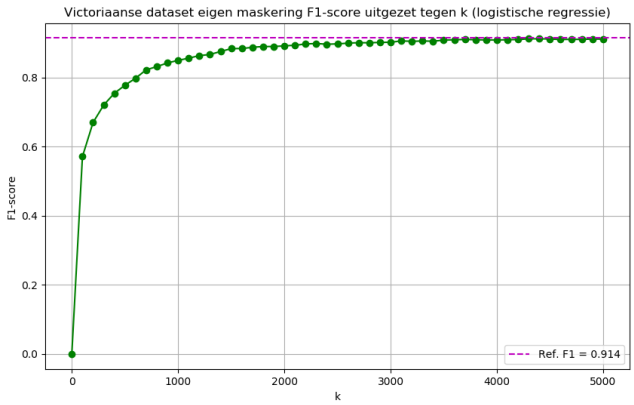


Figure 6.29: Victoriaanse dataset eigen maskering macroprecisie uitgezet tegen k (logistische regressie)

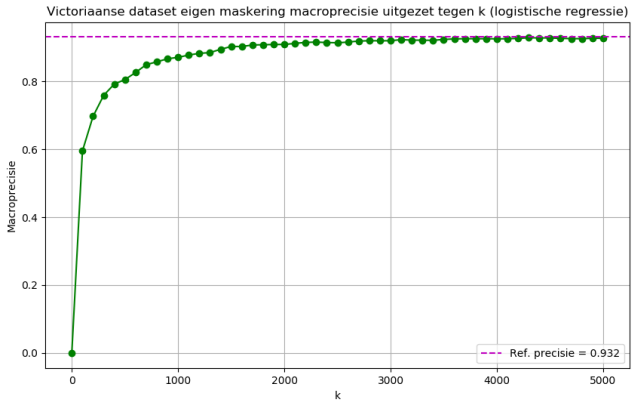
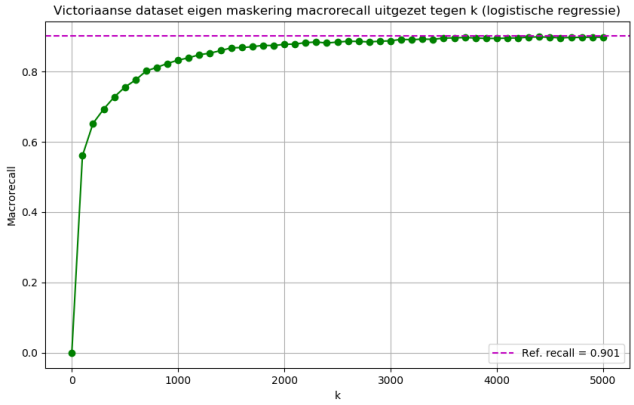


Figure 6.30: Victoriaanse dataset eigen maskering macrorecall uitgezet tegen k (logistische regressie)



Hierin valt hetzelfde op als bij de grafieken voor COCA-maskering (in figuren 6.25, 6.26 en 6.27). Daar komt bovenop dat de grafieken bij eigen maskering de waarden van de referentielijnen raken en daar stabiliseren. Bij COCA-maskering vindt de stabilisatie onder de referentielijn plaats.

Er kan gesteld worden dat logistische regressie als model betere prestaties oplevert dan SVM voor de Victoriaanse dataset. Verder is ook te stellen dat eigen maskering bij de Victoriaanse dataset beter werkt dan COCA-maskering. De grafieken van eigen maskering liggen boven de grafieken van COCA-maskering en ook de maximale waarden rond $k = 5000$ zijn hoger dan bij COCA-maskering.

6.2.3. Federalist Papers dataset

De prestaties van logistische regressie bij de Federalist Papers dataset worden in deze subsectie weergegeven. Allereerst zijn de prestaties voor COCA-maskering in de figuren 6.31, 6.32 en 6.33 te zien.

Figure 6.31: Federalist Papers dataset COCA-maskering F1-score uitgezet tegen k (logistische regressie)

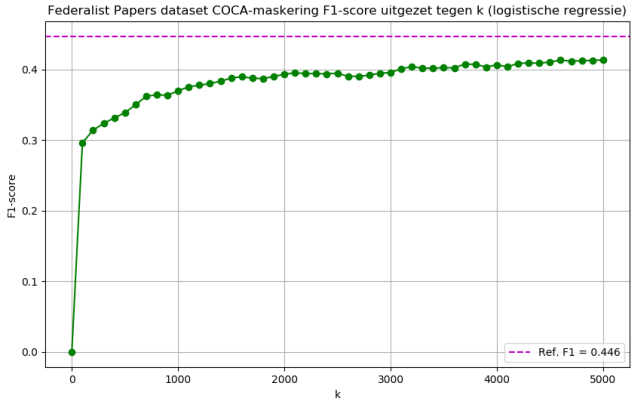


Figure 6.32: Federalist Papers dataset COCA-maskering macroprecisie uitgezet tegen k

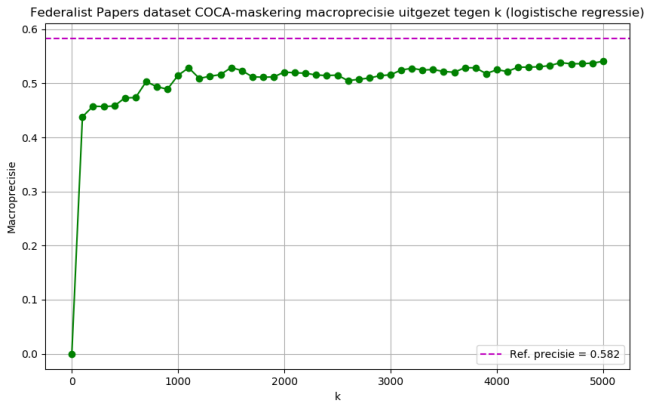
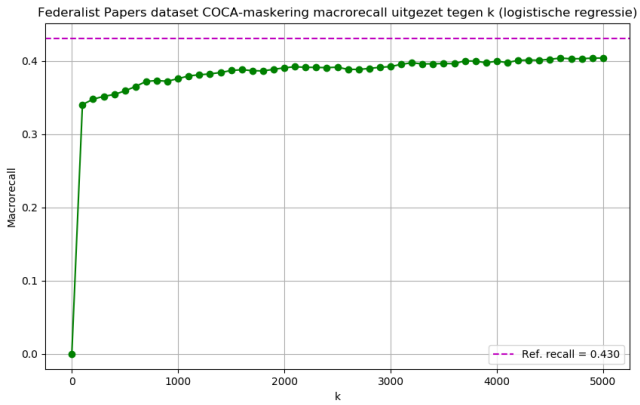


Figure 6.33: Federalist Papers dataset COCA-maskering macroprecisie uitgezet tegen k (logistische regressie)



Het verloop van de grafieken in bovenstaande figuren wijkt niet erg af van het verloop bij SVM (zie figuren 6.13, 6.14 en 6.15). Het verloop van de grafiek in figuur 6.31 is wat ronder en minder lineair dan het verloop bij SVM. Verder zijn er in figuur 6.32 wat schommelingen te zien in de grafiek. Bij SVM waren deze schommelingen ook al te zien, maar bij logistische regressie zitten de schommelingen wel op andere waarden van k. Verder liggen alle grafieken van logistische regressie met COCA-maskering bij de Federalist Papers dataset boven de grafieken van SVM (in figuren 6.13, 6.14 en 6.15).

De waarden voor eigen maskering zijn in onderstaande figuren 6.34, 6.35 en 6.36 uitgezet.

Figure 6.34: Federalist Papers dataset eigen maskering F1-score uitgezet tegen k (logistische regressie)

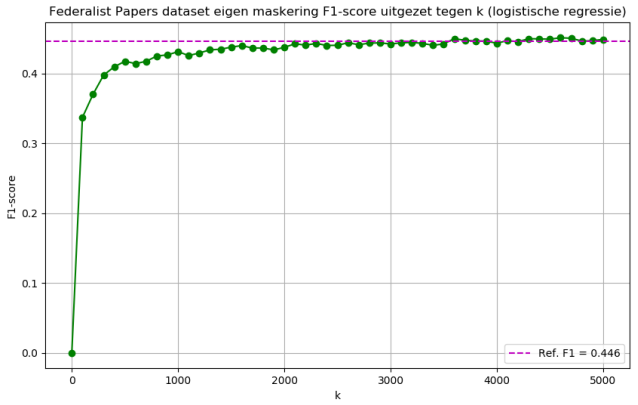


Figure 6.35: Federalist Papers dataset eigen maskering macroprecisie uitgezet tegen k (logistische regressie)

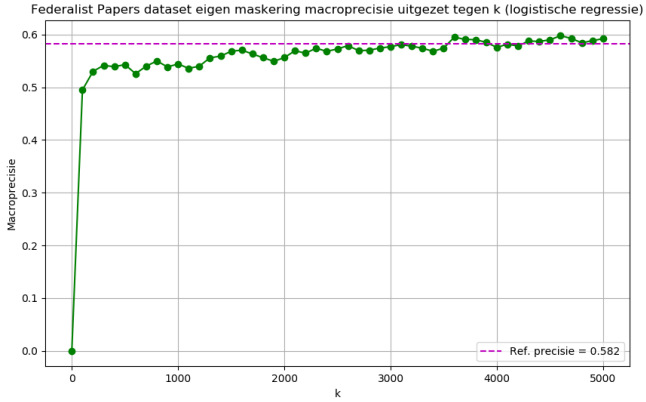
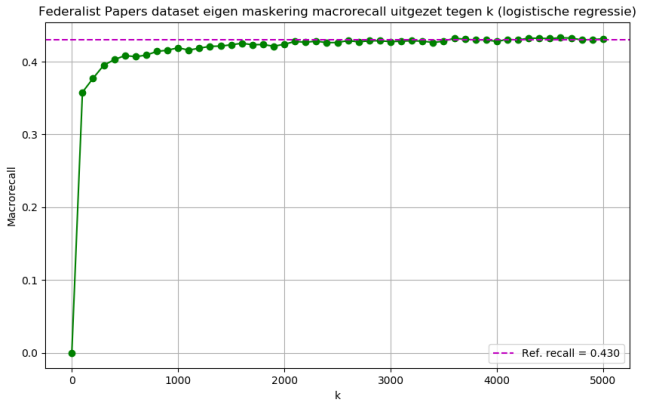


Figure 6.36: Federalist Papers dataset eigen maskering macrorecall uitgezet tegen k (logistische regressie)



In de bovenstaande grafieken van de F_1-score en de $recall_{macro}$ is te zien dat er vanaf $k = 2000$ een stabilisatie rond de referentielijn plaatsvindt. Dit is vergelijkbaar met de grafieken bij SVM (te zien in figuren 6.16, 6.17 en 6.18).

Verder had de grafiek van de $precisie_{macro}$ bij SVM een erg schommelachtig verloop. Ditzelfde schommelachtige verloop, al dan niet minder, is hier te zien in de grafiek van de $precisie_{macro}$ bij logistische regressie (figuur 6.35). De waarden schommelen vanaf $k = 3000$ rond de referentielijn. Door de schommelingen is de waarde voor $k = 5000$ niet de maximale waarde, die wordt bereikt bij $k = 3600$.

Al met al is ook voor de Federalist Papers dataset te concluderen dat logistische regressie, bij het gebruik van maskeren, beter werkt dan SVM. Ook zijn de prestaties voor het gebruik van eigen maskering beter dan voor COCA-maskering.

6.3. Gecombineerde prestaties

Voor het overzicht en om de prestaties van alle verschillende mogelijkheden goed te kunnen vergelijken worden in deze paragraaf alle voorgaande resultaten gecombineerd weergegeven.

Allereerst is hieronder een tabel te zien met de referentiewaarden als er niet gemaskeerd in de teksten van de verschillende datasets.

Figure 6.37: Tabel met referentiewaarden van de F1-score, macroprecisie en macrorecall zonder het toepassen van maskeringstechnieken (onderstreept is de hoogste waarde als SVM en logistische regressie worden vergeleken)

		SVM	*Logistische regressie*
Tweets dataset	*F1-score*	0.818	0.818
	Macroprecisie	<u>0.825</u>	0.823
	Macrorecall	0.814	<u>0.815</u>
Victoriaanse dataset	*F1-score*	0.816	<u>0.914</u>
	Macroprecisie	<u>0.933</u>	0.932
	Macrorecall	0.765	<u>0.901</u>
Federalist Papers dataset	*F1-score*	0.388	<u>0.446</u>
	Macroprecisie	<u>0.680</u>	0.582
	Macrorecall	0.392	<u>0.430</u>

Allereerst valt het voor de Tweets dataset op dat de referentiewaarden van SVM en logistische regressie ontzettend dicht bij elkaar liggen. De waarden verschillen op ongeveer het derde decimaal achter de komma. Dit betekent dat er in de situatie zonder maskeren geen voorkeur is voor SVM of logistische regressie; beide modellen presteren dan even goed.

Voor de Victoriaanse dataset is te zien dat de referentiewaarden voor de F_1-score en de macrorecall substantieel hoger liggen bij logistische regressie. De referentiewaarde voor de macroprecisie is bij logistische regressie ongeveer gelijk aan de waarde bij SVM. Er kan dus gezegd worden dat er voor auteurherkenning bij de Victoriaanse dataset, als er niet gemaskeerd wordt, een voorkeur is voor logistische regressie boven SVM.

Bij de Federalist Papers is er, door de referentiewaarden te vergelijken, niet duidelijk te zeggen of SVM of logistische regressie beter werkt als er niet gemaskeerd wordt. De waarden van de referentie-F_1-score en de referentie-macrorecall liggen bij logistische regressie hoger dan bij SVM, maar de referentie-macroprecisie ligt hoger bij SVM.

In de onderstaande grafieken zijn de prestaties van alle datasets voor alle verschillende mogelijkheden (SVM met COCA-maskering, SVM met eigen maskering, logistische regressie met COCA-maskering en logistische regressie met eigen maskering) weergegeven.

De gecombineerde prestaties voor de Tweets dataset zijn in onderstaande figuren 6.38, 6.39 en 6.40 te zien. Blauw is SVM met COCA-maskering, rood is SVM met eigen maskering, groen is logistische regressie met COCA-maskering en paars is logistische regressie met eigen maskering.

Figure 6.38: Tweets dataset F1-score uitgezet tegen k

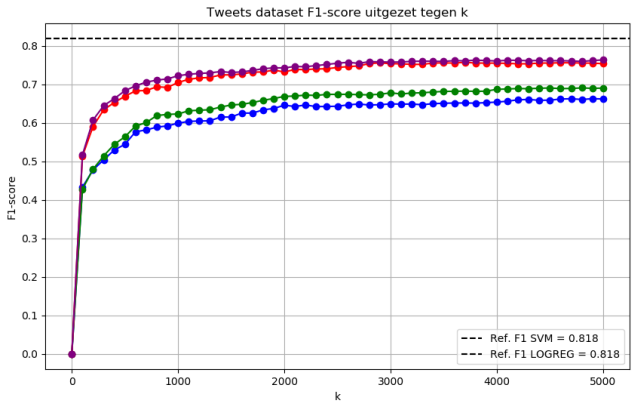


Figure 6.39: Tweets dataset macroprecisie uitgezet tegen k

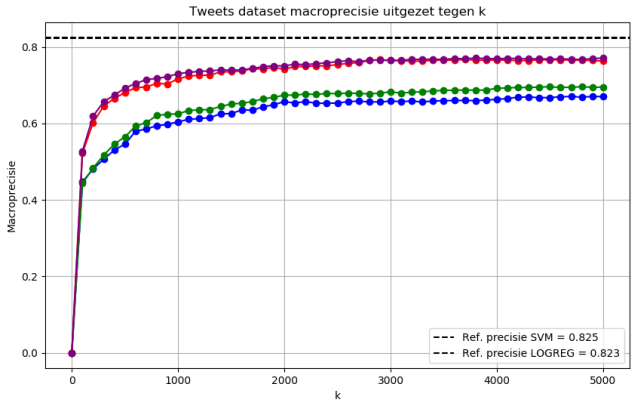
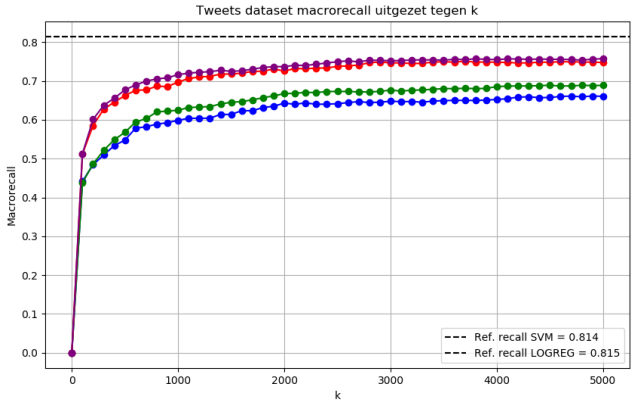


Figure 6.40: Tweets dataset macrorecall uitgezet tegen k



In de grafieken in bovenstaande figuren is goed te zien dat voor de Tweets dataset eigen maskering (paars en rood) beter werkt dan COCA-maskering (blauw en groen). Ook is er een lichte voorkeur te zien voor logistische regressie boven SVM. Bij COCA-maskering is dit verschil goed te zien (vergelijk blauw en groen). Bij eigen maskering is dit verschil een stuk kleiner (vergelijk paars en rood). De

deelconclusie dat eigen maskering beter werkt voor de Tweets dataset dan COCA-maskering en dat logistische regressie een (lichte) voorkeur heeft boven SVM komt overeen met wat er al eerder besproken is in subsectie 6.2.1.

De gecombineerde prestaties voor de Victoriaanse dataset zijn in de onderstaande figuren 6.41, 6.42 en 6.43 weergegeven. Hierin geldt wederom: blauw is SVM met COCA-maskering, rood is SVM met eigen maskering, groen is logistische regressie met COCA-maskering en paars is logistische regressie met eigen maskering.

Figure 6.41: Victoriaanse dataset F1-score uitgezet tegen k

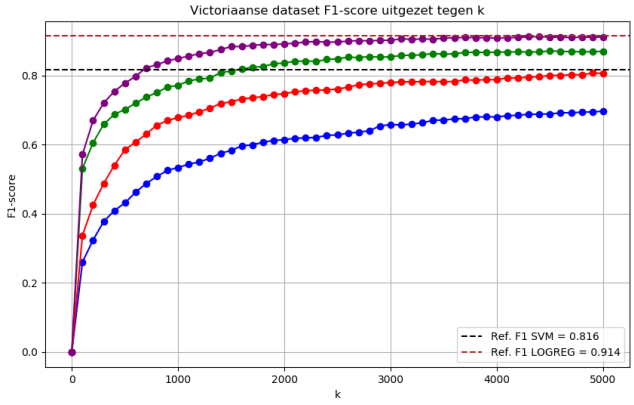


Figure 6.42: Victoriaanse dataset macroprecisie uitgezet tegen k

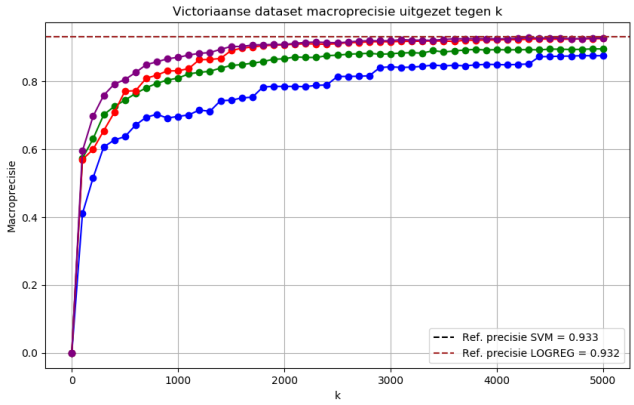
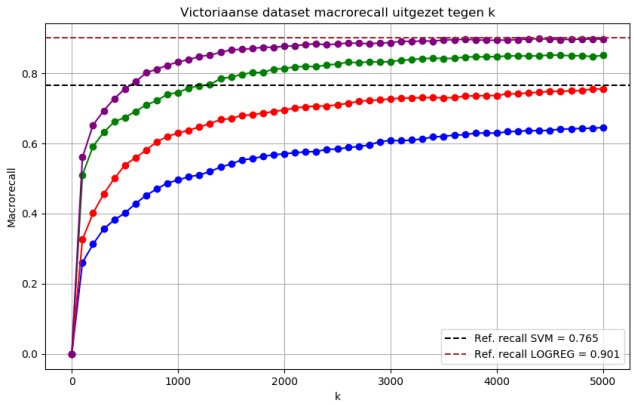


Figure 6.43: Victoriaanse dataset macrorecall uitgezet tegen k



In bovenstaande grafieken is te zien dat logistische regressie (paars en groen) voor de Victoriaanse dataset beter werkt dan SVM (rood en blauw). En ook is duidelijk te zien dat eigen maskering beter werkt dan COCA-maskering (vergelijk rood en blauw, en paars en groen). Dus logistische regressie en eigen maskering werken het beste voor de Victoriaanse dataset. Dit komt overeen met de informatie die al naar voren kwam in subsectie 6.2.2.

De gecombineerde prestaties voor de Federalist Papers dataset zijn in onderstaande figuren 6.44, 6.45 en 6.46 te zien. Hierin geldt wederom: blauw is SVM met COCA-maskering, rood is SVM met eigen maskering, groen is logistische regressie met COCA-maskering en paars is logistische regressie met eigen maskering.

Figure 6.44: Federalist Papers dataset F1-score uitgezet tegen k

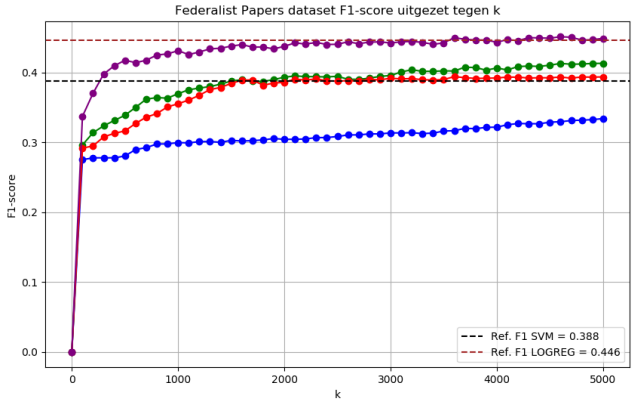


Figure 6.45: Federalist Papers dataset macroprecisie uitgezet tegen k

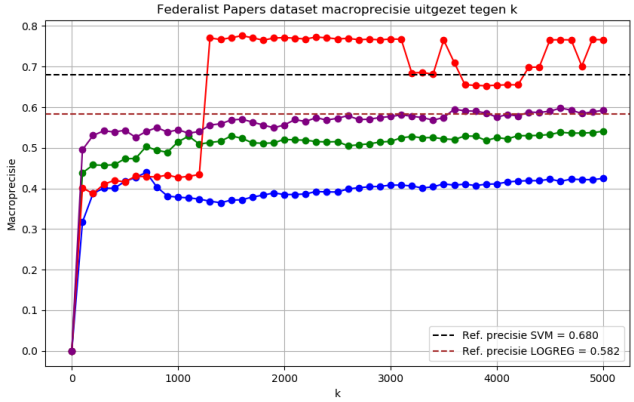
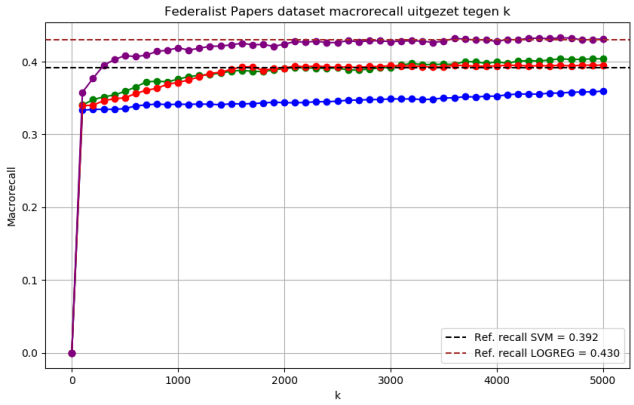


Figure 6.46: Federalist Papers dataset macrorecall uitgezet tegen k



In bovenstaande figuren is, net als bij de Tweets dataset en Victoriaanse dataset, te zien dat logistische regressie in combinatie met eigen maskering ook het beste werkt voor de Federalist Papers dataset. Al is de grafiek van de macroprecisie (figuur 6.45) daarop wel een uitzondering. Dit grillige verloop is al besproken in subsectie 6.1.3, en vanwege de grote uitschieters worden deze waarden met een korreltje zout genomen in het kijken naar het trekken van een deelconclusie voor de Federalist Papers dataset. Net als bij de vorige datasets kan er dus geconcludeerd worden dat logistische regressie als model beter werkt dan SVM en dat eigen maskering ook betere resultaten oplevert dan COCA-maskering. Dit komt overeen met wat er al besproken is in subsectie 6.2.3.

7

Onderwerprobuustheid

De reden waarom maskeringstechnieken gebruikt worden in auteurherkenning is om de onderwerpafhankelijkheid van het model te verminderen, zoals werd uitgelegd in sectie 2.7, en het model onderwerprobuust te maken. In hoofdstuk 5 is er gekeken naar de prestaties maar in deze prestaties wordt niet de onderwerprobuustheid meegenomen. Daar wordt in dit hoofdstuk naar gekeken.

De moeilijkheid zit erin dat er in dit onderzoek drie datasets gebruikt worden waarvan teksten en auteur bekend zijn, maar niet het onderwerp van de teksten. Allereerst zal er daarom geprobeerd worden om de teksten van de Tweets dataset op onderwerp te scheiden en daarmee iets te zeggen over hoe de onderwerprobuustheid verandert door het maskeren. Er is voor de Tweets dataset gekozen omdat tweets hashtags bevatten waaruit het onderwerp van de tweet afgeleid kan worden. Daarna zal er gekeken worden of wat er vanuit de literatuur te vinden is over onderwerprobuustheid en maskeren.

7.1. Onderwerp extraheren op basis van hashtags

De Tweets dataset bevat tweets en daarin komen vaak hashtags voor. De hashtags die men er in paragraaf 3.1 uit heeft gehaald voor de toepassing van maskeringstechnieken en analyse van de prestaties kunnen juist belangrijk zijn als het gaat om het bepalen van het onderwerp van teksten. Hashtags bevatten namelijk vaak informatie over het onderwerp van de tweet.

Om te kijken of de tweets te groeperen zijn in groepen van verschillende onderwerpen gaat er eerst gekeken worden naar welke hashtags er allemaal voorkomen in alle tweets en hoe vaak deze voorkomen.

Hieruit konden er vier groepen van onderwerpen gevormd worden waar vaak over geschreven werd:

- *Klimaatopwarming*: Als een tweet #ActOnClimate bevat, dan bevat de tweet informatie over de klimaatopwarming en valt de tweet in deze groep.
- *Televisieprogramma 'American Idol'*: Als een tweet #AmericanIdol of #americanidol bevat, dan bevat de tweet informatie over het televisieprogramma 'American Idol' en valt de tweet in deze groep.
- *Algoritmes en technologie:* Als een tweet #NLProc, #DeepLearning, #MachineLearning, #deeplearning of #machinelearning bevat, dan bevat de tweet informatie over algoritmes en technologie en valt de tweet in deze groep.
- *Zorgsysteem en zorgverzekeringen:* Als een tweet #GetCovered of #Obamacare bevat, dan bevat de tweet informatie over het zorgsysteem en zorgverzekeringen en valt de tweet in deze groep

De hashtags van de hierboven gekozen groepen komen veel in de tweets voor (tussen 100 en 350 keer). Daarom zijn deze groepen veelbelovend.

Vervolgens moet er gecontroleerd worden of alle (of in ieder geval meerdere) auteurs teksten over verschillende onderwerpen hebben geschreven. Dit gaf een teleurstellend resultaat. Hieruit kwam naar voren dat bijvoorbeeld Barack Obama veel tweets heeft geschreven over klimaatopwarming en het zorgsysteem, maar de andere auteurs niet. En dat bijvoorbeeld Katy Perry veel schreef over het televisieprogramma 'American Idol' maar verder geen enkele andere auteur. Hetzelfde geldt voor de groep over algoritmes en technologie; daar schreef alleen Sebastian Ruder over.

In de Tweets dataset kunnen er dus, op basis van het analyseren van de hashtags, geen onderwerpen gevonden worden waar meerdere auteurs over hebben geschreven. Hierdoor kan er in de Tweets dataset niet specifiek onderzocht worden wat het maskeren doet met de onderwerprobuustheid van het model.

7.2. Onderwerprobuustheid op basis van literatuur

7.2.1. Kwalitatieve analyse

Omdat de onderwerpen van de teksten in de Victoriaanse dataset en de Federalist Papers dataset niet beschikbaar zijn en in het geval van de Tweets dataset geen onderwerpen zijn waar meerdere auteurs over geschreven hebben, kunnen er geen testen met de datasets worden gedaan om te kijken wat er gebeurt met de onderwerprobuustheid als er gemaskeerd wordt. Daarom zal er gekeken worden wat er in de literatuur te vinden is wat betreft maskeren en onderwerprobuustheid, en of conclusies daaruit ook zouden kunnen gelden voor de datasets in dit onderzoek.

Allereerst is het belangrijk om de onderwerprobuustheid te zien in combinatie met de prestaties van het model. Een lage onderwerpafhankelijkheid, en daarmee een hoge onderwerprobuustheid, maakt het model namelijk niet per definitie beter. Een model dat bijvoorbeeld alle teksten aan auteur a_i toewijst heeft een onderwerpafhankelijkheid van 0 en is helemaal onderwerprobuust, maar het heeft ook een hele lage F_1-score [6].

In de figuren van het vorige hoofdstuk is te zien dat de prestaties (F_1-score, macroprecisie en macrorecall) toenemen als de waarde van k stijgt. Maar als de waarde van k stijgt, wordt de onderwerprobuustheid van het model minder, omdat er minder woorden (en daarmee minder onderwerpafhankelijke woorden) gemaskeerd worden. Daarom is er geen waarde van k te vinden waarbij zowel de prestaties als de onderwerprobuustheid optimaal zijn. Er moet daarin dus een afweging gemaakt worden. Deze afweging kan het beste per specifieke dataset gemaakt worden [6].

Bij datasets waarin de teksten over dezelfde onderwerpen gaan, is het belangrijker om een model te hebben dat goede prestaties heeft dan een goede onderwerprobuustheid. Aan de andere kant, bij datasets waarin de teksten heel erg verschillen van onderwerp, gaat de voorkeur juist uit naar een onderwerprobuust model, ook al gaat dit ten koste van de prestaties [6].

Maar uit literatuur blijkt ook dat schrijfstijl meegenomen moet worden in deze afweging. Onderwerpafhankelijke informatie kan juist waardevol zijn in een dataset waarin de teksten over verschillende onderwerpen gaan maar de schrijfstijl van verschillende auteurs heel erg op elkaar lijkt. Met alleen de schrijfstijl, die je overhoudt als er veel gemaskeerd wordt, kun je dan de teksten niet goed classificeren. Datasets waarin er dus sprake is van dezelfde schrijfstijl zullen niet baten bij het maskeren van onderwerpgerelateerde informatie [12].

Zoals in sectie 7.1 te zien is bevat de Tweets dataset tweets over veel verschillende onderwerpen. Verder is het lastig om veel verschillen in schrijfstijl tussen de verschillende auteurs te onderscheiden omdat tweets erg korte teksten zijn, maar dit is wel mogelijk. Doordat de Tweets dataset dus veel verschillende onderwerpen bevat en er in sommige gevallen wel een verschillende schrijfstijl te zien kan zijn, kan de onderwerprobuustheid het beste verkozen worden boven de prestaties.

In de Victoriaanse dataset is het aannemelijk dat de teksten gaan over verschillende onderwerpen en is het duidelijk dat er veel verschil zit in schrijfstijl. Charles Darwin, bijvoorbeeld, schrijft over hele andere onderwerpen en met een hele andere schrijfstijl dan Rudyard Kipling. In het geval van deze

dataset kan onderwerprobuustheid dus het beste verkozen worden boven de prestaties. Hierom zou er dus voor een lagere k gekozen worden.

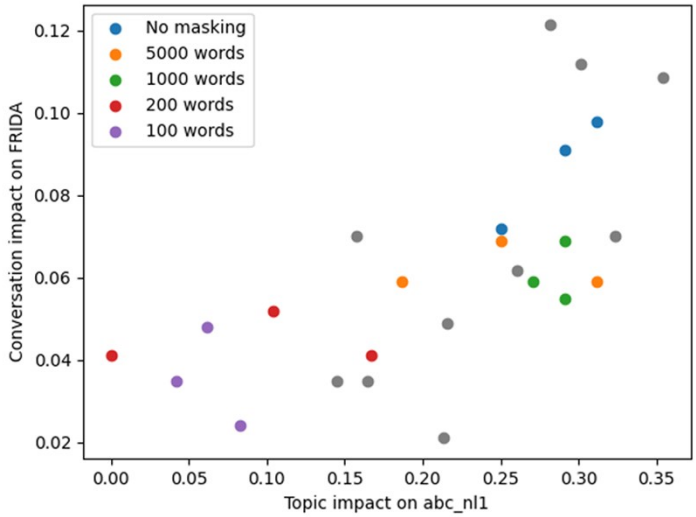
Bij de Federalist Papers gaan de teksten juist allemaal over hetzelfde onderwerp en is de schrijfstijl van de verschillende auteurs ook soortgelijk. Daarom gaan prestaties boven onderwerprobuustheid en kan er het beste voor een zo groot mogelijke k worden gekozen.

7.2.2. Kwantitatieve analyse

In de vorige subsectie 7.2.1 is er op een kwalitatieve manier iets gezegd over de onderwerprobuustheid in combinatie met het maskeren voor de datasets. Naast deze kwalitatieve analyse kan er ook gekeken worden of er op een kwantitatieve manier iets gezegd kan worden over deze onderwerprobuustheid.

Onderstaande figuur komt uit het onderzoek van Hajer [6]. In deze figuur is de onderwerpafhankelijkheid van een dataset genaamd abc_nl1 uitgezet tegen de converstatie-impact van een andere dataset genaamd FRIDA [6].

Figure 7.1: Onderwerpafhankelijkheid van dataset abc_nl1 uitgezet tegen de conversatie-impact van dataset FRIDA voor SVM bij verschillende aantallen ongemaskeerde woorden [6]



FRIDA is een dataset die gecreëerd is door het Nederlands Forensisch Instituut (NFI) en deze dataset bevat telefoongesprekken tussen individuen [17]. De abc_nl1 dataset is in 1999 gecreëerd door de Radboud Universiteit en bevat teksten over verschillende onderwerpen geschreven door acht studenten [1].

In dit onderzoek heeft men het nog niet eerder over conversatie-impact gehad. Conversatie-impact wordt in het onderzoek van Hajer gebruikt als proxy om naar de onderwerprobuustheid van een model te kunnen kijken [6]. In conversaties is het namelijk aannemelijk dat beide gesprekspartners het over hetzelfde onderwerp hebben, waardoor de conversatie-impact ook iets kan zeggen over de onderwerpafhankelijkheid. Datasets waarin conversatie-impact bestudeerd kan worden zijn ook meer openbaar beschikbaar dan datasets waarin onderwerprobuustheid bestudeerd kan worden. Voor onderwerprobuustheid zijn namelijk teksten nodig waarbij één auteur over verschillende onderwerpen teksten heeft geschreven, dit komt niet zo vaak voor, en er zijn meerdere auteurs nodig die over deze dezelfde onderwerpen hebben geschreven, dus dat maakt het nog ingewikkelder. Er zijn weinig openbare datasets beschikbaar die aan deze eisen voldoen. Daarom is het erg handig om conversatie-impact als proxy voor onderwerprobuustheid te gebruiken.

In het onderzoek van Hajer zijn, met behulp van verschillende testsets, maten gedefinieerd om aan de conversatie-impact en de onderwerpafhankelijkheid getalwaardes te kunnen geven zodat deze onderzocht kunnen worden [6]. Er werd ook gevonden dat conversatie-impact als proxy niet perfect werkt maar er is wel een correlatie van 0.68 gevonden met de onderwerpafhankelijkheid [6].

In figuur 7.1 is voor SVM de onderwerpafhankelijkheid van abc_nl1 uitgezet tegen de conversatie-impact van FRIDA bij verschillende aantallen ongemaskeerde woorden. Dit is te zien aan de gekleurde punten. Bij de grijze punten is geen SVM gebruikt maar zijn andere classificatiemethoden gebruikt. Deze worden in dit onderzoek niet behandeld en zijn dus niet relevant.

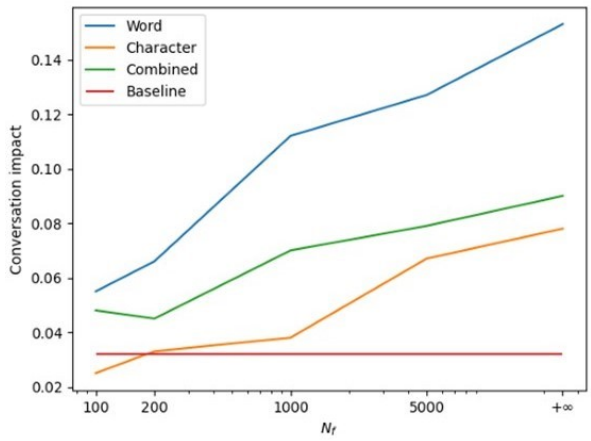
De grafiek in figuur 7.1 bevestigt de intuïtie dat de onderwerpafhankelijkheid en de conversatie-impact afnemen naarmate er meer gemaskeerde woorden zijn.

Er valt te zeggen dat dataset abc_nl1 overeenkomsten vertoont met de Victoriaanse dataset en de Federalist Papers. Wellicht klinkt het raar om de teksten die geschreven zijn door studenten te vergelijken met teksten die zijn geschreven door grote meesters, maar de structuur van de teksten zal wel op elkaar lijken en is bijvoorbeeld heel anders dan bij de Tweets dataset. Er is dan in figuur 7.1 te zien dat er weinig verschil zit in de onderwerpafhankelijkheid als er geen woorden gemaskeerd worden en als er voor $k = 1000$ gekozen wordt. Dit is iets wat meegenomen kan worden in het trekken van de conclusies over de optimale k voor deze datasets in dit onderzoek.

Om daarnaast ook meer te kunnen zeggen over de onderwerpafhankelijkheid van de Tweets dataset in dit onderzoek kan onderstaande figuur 7.2 uit het onderzoek van Hajer gebruikt worden [6].

In figuur 7.2 is de conversatie-impact bij het gebruik van drie verschillende SVM modellen uitgezet tegen het aantal gemaskeerde woorden in een dataset genaamd RFM.

Figure 7.2: Conversatie-inpact bij het gebruik van drie SVM modellen uitgezet tegen het aantal gemaskeerde woorden in dataset RFM [6]



(a) RFM

RFM is een dataset die chatberichten uit de forensische opsporing bevat [6]. Dataset RFM vertoont overeenkomsten met de Tweets dataset in dit onderzoek. Beide datasets bevatten over het algemeen korte, online berichten.

Aangezien conversatie-impact een proxy is voor onderwerpafhankelijkheid en hier een correlatie tussen gevonden is in het onderzoek van Hajer, kan deze grafiek in figuur 7.2 helpen bij de interpretatie van de onderwerpafhankelijkheid op de Tweets dataset [6]. De blauwe lijn in deze figuur komt het meest

overeen met het model van SVM dat in dit onderzoek is gebruikt.

8

Conclusie en Discussie

Allereerst is er gekeken naar de prestaties voor de twee verschillende manieren van maskeren (COCA-maskering en eigen maskering) en de twee classificatiemethoden (SVM en logistische regressie). Wat betreft deze prestaties kwamen er twee deelconclusies naar voren.

Er blijkt dat logistische regressie voor alle datasets geschikter is dan SVM als classificatiemethode. Dit is opvallend en lijkt het onderzoek van Hajer tegen te spreken, waarin juist SVM veel beter werkt dan logistische regressie [6]. Maar in het onderzoek van Hajer wordt als baseline logistische regressie bij 100 ongemaskeerde woorden ($k = 100$) gebruikt en dan is het inderdaad zo dat SVM al gauw beter presteert dan de baseline [6]. De resultaten in dit onderzoek spreken het onderzoek van Hajer dus niet tegen.

Verder blijkt ook dat maskering door middel van een eigen woordenlijst (eigen maskering) voor alle datasets beter werkt dan maskering met de algemene COCA-woordenlijst (COCA-maskering). Dit is mogelijk te verklaren door het taalgebruik in de datasets. De datasets bevatten allemaal verschillend taalgebruik, maar tweets bevatten vaak 'slang' en de andere teksten zijn historisch waardoor ze eigenlijk allemaal geen 'normaal' taalgebruik bevatten dat goed overeenkomt met de COCA-woordenlijst. Daarom is het logisch dat eigen maskering in alle datasets beter past.

In de onderzoeksvraag die in hoofdstuk 1 is gesteld, wil men een aanbeveling kunnen doen wat betreft het optimale k aantal woorden om te maskeren waarbij naast de prestaties ook de onderwerprobuustheid is meegenomen.

Allereerst zal er gekeken worden naar de Tweets dataset. Bij deze dataset kan onderwerprobuustheid het beste boven de prestaties verkozen worden (zie hoofdstuk 7). Er kan dan dus het beste een lage k gekozen worden. Het is bijvoorbeeld een goede keuze om een k tussen $k = 700$ en $k = 1000$ te kiezen. Bij deze waarden zijn de prestaties niet maximaal maar omdat de prestaties tussen $k = 1000$ en $k = 5000$ nauwelijks toenemen zijn de prestaties nog steeds wel goed genoeg en wijken niet heel veel af van de maximale waarden (zie figuren 6.38, 6.39 en 6.40).

Ook bij de Victoriaanse dataset is de onderwerprobuustheid van het model belangrijker dan de prestaties (zie hoofdstuk 7). Daarom kan hier dus ook het beste een lage k gekozen. In tegenstelling tot bij de Tweets dataset, stijgen de prestaties bij de Victoriaanse dataset tussen $k = 0$ en $k = 2000$ nog aanzienlijk en stabiliseert deze pas vanaf $k = 2000$ to $k = 5000$ (zie figuren 6.41, 6.42 en 6.43). Rekening houdend met het belang van de onderwerprobuustheid maar ook oog houdend op de prestaties kan hier het beste voor een k tussen $k = 1000$ en $k = 2000$ gekozen kan worden.

Tot slot, bij de Federalist Papers kunnen de prestaties het beste boven de onderwerprobuustheid verkozen worden (zie hoofdstuk 7). Daarom kan er in figuren 6.44, 6.45 en 6.46 gekeken worden naar de waarde van k waarbij de prestaties maximaal zijn. Zoals eerder is geconcludeerd, zijn de

prestaties het beste bij logistische regressie met eigen maskering. Als bij deze omstandigheden in de figuren naar de waarden van de F_1-score, macroprecisie en macrorecall wordt gekeken, dan bereiken deze bij $k = 3600$ maximale waarden (ongeveer gelijk als bij $k = 5000$). Voor de Federalist Papers dataset kan er dan het beste een k van 3600 gekozen worden.

Naar aanleiding van dit onderzoek zijn er nog enkele beperkingen die aangepast en onderzocht zouden kunnen worden in een vervolgonderzoek.

Allereerst zijn de aanbevelingen wat betreft de onderwerprobuustheid in dit onderzoek niet ontzettend sterk. Conclusies uit de literatuur zijn geprobeerd toe te passen op de datasets in dit onderzoek. Het zou sterker zijn om de onderwerprobuustheid specifiek voor de datasets te onderzoeken. Er zou nog naar de Victoriaanse dataset en Federalist Papers dataset gekeken kunnen worden of de onderwerpen te extraheren zijn (bijvoorbeeld door Latent Dirichlet Allocation, LDA, te gebruiken) zodat er gekeken kan worden of de onderwerprobuustheid voor de datasets specifiek onderzocht kan worden. Of er moeten andere datasets gekozen worden waarbij vooraf al bekend is over welke onderwerpen de teksten zijn geschreven en meerdere auteurs over dezelfde onderwerpen hebben geschreven. Bijvoorbeeld de abc_nl1 dataset zou hier geschikt voor zijn [1].

In het voorbereiden ('schoonmaken') zijn een aantal keuzes gemaakt om bepaalde details uit de teksten te halen en buiten beschouwing te laten. Er zou onderzocht kunnen worden wat er gebeurt als er daarin andere keuzes worden gemaakt. Bijvoorbeeld wat er gebeurt als de emoji's in de Tweets dataset wél meegenomen worden. Ook is bij het schoonmaken de keuze gemaakt om eerst de spelling te verbeteren en dan te maskeren, om zo om te gaan met spelfouten. Een manier die wellicht nog beter zou werken is om eerst de spelling te verbeteren, dan te maskeren en dan weer de oorspronkelijke spelling van de woorden terug te halen [6].

Verder is er in hoofdstuk 2 te lezen dat er verschillende manieren zijn om woorden te maskeren. Zo is er in dit onderzoek gemaskeerd door elk gemaskeerd woord door een enkele asterisk * te vervangen. Dit is de DV-SA methode. Er zou gekeken kunnen worden of het maskeren door middel van DV-MA, DV-EX of DV-L2 andere of misschien nog betere resultaten oplevert. Ook is het een mogelijkheid om bij het maskeren een historische woordenlijst te gebruiken. Omdat in dit onderzoek twee van de drie datasets historische teksten bevatten zou maskeren met een woordenlijst uit bijbehorende historische tijdperken wellicht betere resultaten kunnen opleveren.

Ook is er in dit onderzoek gebruik gemaakt van slechts 25 procent van alle teksten in de Victoriaanse dataset. Bij het gebruik van een sterkere computer of bij het beschikbaar hebben van meer (reken)tijd zou 100 procent van deze dataset gebruikt kunnen worden. Dit zou nog accuratere resultaten opleveren.

In dit onderzoek was het hoofddoel om een aanbeveling te doen wat betreft het optimale aantal te maskeren woorden. Maar als deelonderzoek is er ook gekeken naar twee verschillende classificatiemethoden (SVM en logistische regressie) en zijn deze vergeleken. In een vervolgonderzoek zou er naar nog meer verschillende modellen gekeken kunnen worden, zoals bijvoorbeeld neurale netwerken of transformer-gebaseerde modellen, en kan er gekeken worden naar de invloed die maskeringstechnieken daarop hebben.

Bibliography

[1] H. Baayen and et al. ”an experiment in authorship attribution”. *6th JADT*, 1:69–75, 2002.

[2] T. Barrus. pyspellchecker, 2018.

[3] T. Fletcher. Support vector machines explained, 2008.

[4] R. Galante Negri, L. Vieira Dutra, and S. João Siqueira Sant'Anna. An innovative support vector machine based method for contextual image classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2013.

[5] A. Gungor. Benchmarking authorship attribution techniques using over a thousand books by fifty victorian era novelists. Master's thesis, Purdue University, 2018.

[6] W. Hajer. Authorship attribution in a forensic setting. Master's thesis, Delft University of Technology, 2024.

[7] X. He, A.H. Lashkari, N. Vombatkere, and D.P Sharma. Authorship attribution methods, challenges, and future research directions: A comprehensive survey. *Information*, 15(3):131, 2024.

[8] D.I. Holmes and R.S. Forsyth. The federalist revisited: New directions in authorship attribution. *Literary and Linguistic Computing*, 10(2), 1995.

[9] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor. *An Introduction to Statistical Learning: with Applications in Python*. Springer Texts in Statistics, 2023.

[10] P. Juola. Authorship attribution. *Foundation and Trends in Information Retrieval*, 1(3):223–334, 2008.

[11] Rijksoverheid. Desinformatie, misinformatie en nepnieuws, 2024.

[12] Y. Sara, M. Stevenson, and A. Vlachos. Topic or style? exploring the most useful features for authorship attribution. *International Conference on Computational Linguistics*, 2018.

[13] J. Savoy. *Machine Learning Methods for Stylometry: Authorship Attribution and Author Profiling*. Springer Nature Switzerland, 2020.

[14] Y. Seroussi, Zukerman I., and Bohnert F. Authorship attribution with topic models. *Association for Computational Linguistics*, 40(2):42, 2014.

[15] E. Stamatatos. Masking topic-related information to enhance authorship attribution. *Journal of the Association for Information Science and Technology*, 69(3):461–473, 2018.

[16] P. Tieleman. Support vector machine in python: Uitleg tutorial, 2020.

[17] D. Van der Vloed and et al. ”nfi-frida-forensically realistic interdevice audio database and initial experiments”. *27th Annual Conference of the International Association for Forensic Phonetics and Acoustics (IAFPA)*, pages 25–27, 2018.

[18] D.M. Vescovi. Best practices in authorship attribution of english essays. Master's thesis, Duquesne University, 2011.

[19] WordFrequency.info. Word frequency data.

A

Pythoncodes Tweets dataset

In de eerste, onderstaande, code wordt de Tweets dataset schoongemaakt voor gebruik en wordt er SVM toegepast:

```
import pandas as pd
from tabulate import tabulate
data_tweets = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Celebrity_Tweets\\
    tweet_with_authors.csv", delimiter=",", error_bad_lines=False, header=0, encoding="utf-8"
    )
# data wordt ingelezen, gescheiden op komma, met juiste encoding

from spellchecker import SpellChecker

spel = SpellChecker()

filter_rt= data_tweets['tweet'].str.contains("RT")
data_tweets = data_tweets[~filter_rt] #tweets die een retweet zijn worden eruit gehaald

import re
import emoji

def remove_emojis(text):
    return emoji.replace_emoji(text, replace='')

data_tweets['tweet']=data_tweets['tweet'].apply(remove_emojis) #emojis uit de tweet
    verwijderd

def clean_text(text):
    # Verwijder woorden die starten met # of http
    text = re.sub(r'\s#\S+|\b#\S+|\shttp\S+|\bhttp\S+', '', text)
    # Verwijder tekst tussen dubbele aanhalingstekens
    text = re.sub(r'".*?"', '', text)
    # Verwijder overbodige spaties
    text = ' '.join(text.split())
    return text

data_tweets['tweet']=data_tweets['tweet'].apply(clean_text)


#print(tabulate(data_tweets))
#print(data_tweets.columns)


auteurs_groep = data_tweets.groupby('author')['tweet'].apply(list).to_dict() #dictionary met
    auteursgroepen

gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

for auteur, teksten in auteurs_groep.items():
    gecorrigeerde_teksten[auteur] = []
```

```
    for tekst in teksten:
        woorden = tekst.split()
        gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [woord] for woord in woorden]
        gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in gecorrigeerde_woorden]))
#spelling gecorrigeerd

#printen om te checken
#for auteur, teksten in gecorrigeerde_teksten.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

#keys = gecorrigeerde_teksten.keys()
#print(keys)

rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in gecorrigeerde_teksten.items() for tweet in tweets]
df = pd.DataFrame(rows)

# Import train_test_split function
from sklearn.model_selection import train_test_split

x = df['tweet']
y = df['auteur']

from tfIdfInheritVectorizer.feature_extraction.vectorizer import TfidfVectorizer

vectorizer = TfidfVectorizer()
X_tfidf = vectorizer.fit_transform(x)


# Split dataset into training set and test set
tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y, test_size=0.3,random_state=109) # 70% training and 30% test

#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(tweet_train, author_train)

#Predict the response for test dataset
author_pred = clf.predict(tweet_test)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy: how often is the classifier correct?
#print("Accuraatheid:",metrics.accuracy_score(author_test, author_pred))
print("Macroprecisie:",metrics.precision_score(author_test, author_pred, average='macro')) #bij average kan je micro, macro of weighted instellen
print("Gevoeligheid:",metrics.recall_score(author_test, author_pred, average='macro')) #ook hier bij average micro, macro of weighted kiezen
print("F1-score:",metrics.f1_score(author_test, author_pred, average='macro'))
```

Bij de volgende code wordt de Tweets dataset schoongemaakt en wordt er logistische regressie toegepast:

```
import pandas as pd
from tabulate import tabulate
data_tweets = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Celebrity_Tweets\\tweet_with_authors.csv", delimiter=",", error_bad_lines=False, header=0, encoding="utf-8")
# data wordt ingelezen, gescheiden op komma, met juiste encoding

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

from spellchecker import SpellChecker

spel = SpellChecker()

filter_rt= data_tweets['tweet'].str.contains("RT")
data_tweets = data_tweets[~filter_rt] #tweets die een retweet zijn worden eruit gehaald

import re
import emoji

def remove_emojis(text):
    return emoji.replace_emoji(text, replace='')

data_tweets['tweet']=data_tweets['tweet'].apply(remove_emojis) #emojis uit de tweet verwijderd

def clean_text(text):
    # Verwijder woorden die starten met # of http
    text = re.sub(r'\s#\S+|\b#\S+|\shttp\S+|\bhttp\S+', '', text)
    # Verwijder tekst tussen dubbele aanhalingstekens
    text = re.sub(r'".*?"', '', text)
    # Verwijder overbodige spaties
    text = ' '.join(text.split())
    return text

data_tweets['tweet']=data_tweets['tweet'].apply(clean_text)


#print(tabulate(data_tweets))
#print(data_tweets.columns)


auteurs_groep = data_tweets.groupby('author')['tweet'].apply(list).to_dict() #dictionary met auteursgroepen

gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

for auteur, teksten in auteurs_groep.items():
    gecorrigeerde_teksten[auteur] = []
    for tekst in teksten:
        woorden = tekst.split()
        gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [woord] for woord in woorden]
        gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in gecorrigeerde_woorden]))
#spelling gecorrigeerd

#printen om te checken
#for auteur, teksten in gecorrigeerde_teksten.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

#keys = gecorrigeerde_teksten.keys()
#print(keys)

rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in gecorrigeerde_teksten.items() for tweet in tweets]
df = pd.DataFrame(rows)

x = df['tweet']
y = df['auteur']

vectorizer = CountVectorizer(stop_words=None)
X_tfidf = vectorizer.fit_transform(x)


# Split dataset into training set and test set
```

```
tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y, test_size=0.3,random_state=109) # 70% training and 30% test

model = LogisticRegression()
model.fit(tweet_train, author_train)

author_pred = model.predict(tweet_test)
print(classification_report(author_test, author_pred))


#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy: how often is the classifier correct?
#print("Accuraatheid:",metrics.accuracy_score(author_test, author_pred))
print("Macroprecisie:",metrics.precision_score(author_test, author_pred, average='macro')) # bij average kan je micro, macro of weighted instellen
print("Gevoeligheid:",metrics.recall_score(author_test, author_pred, average='macro')) #ook hier bij average micro, macro of weighted kiezen
print("F1-score:",metrics.f1_score(author_test, author_pred, average='macro'))
```

In de volgende code wordt COCA-maskering op de Tweets dataset gebruikt en wordt er SVM toegepast voor de classificatie. Dit wordt voor verschillende waarden van k gedaan waardoor er grafieken uit komen:

```
import pandas as pd
from tabulate import tabulate
data_tweets = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Celebrity_Tweets\\tweet_with_authors.csv", delimiter=",", error_bad_lines=False, header=0, encoding="utf-8")
# data wordt ingelezen, gescheiden op komma, met juiste encoding

import numpy as np
import matplotlib.pyplot as plt
 #Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

from spellchecker import SpellChecker

spel = SpellChecker()

filter_rt= data_tweets['tweet'].str.contains("RT")
data_tweets = data_tweets[~filter_rt] #tweets die een retweet zijn worden eruit gehaald

import re
import emoji
from tfIdfInheritVectorizer.feature_extraction.vectorizer import TfidfVectorizer


def remove_emojis(text):
    return emoji.replace_emoji(text, replace='')

data_tweets['tweet']=data_tweets['tweet'].apply(remove_emojis) #emojis uit de tweet verwijderd

def clean_text(text):
    # Verwijder woorden die starten met # of http
    text = re.sub(r'\s#\S+|\b#\S+|\shttp\S+|\bhttp\S+', '', text)
    # Verwijder tekst tussen dubbele aanhalingstekens
    text = re.sub(r'".*?"', '', text)
    # Verwijder overbodige spaties
    text = ' '.join(text.split())
    return text

data_tweets['tweet']=data_tweets['tweet'].apply(clean_text)


#print(tabulate(data_tweets))
#print(data_tweets.columns)

```

```
43
44 auteurs_groep = data_tweets.groupby('author')['tweet'].apply(list).to_dict() #dictionary met
       auteursgroepen
45
46 gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten
47
48 for auteur, teksten in auteurs_groep.items():
49     gecorrigeerde_teksten[auteur] = []
50     for tekst in teksten:
51         woorden = tekst.split()
52         gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [
       woord] for woord in woorden]
53         gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in
       gecorrigeerde_woorden]))
54 #spelling gecorrigeerd
55
56 #printen om te checken
57 #for auteur, teksten in gecorrigeerde_teksten.items():
58 #    print(f"Auteur: {auteur}")
59 #    for tekst in teksten:
60 #        print(f"  Tekst: {tekst}")
61
62 #keys = gecorrigeerde_teksten.keys()
63 #print(keys)
64
65 frequentwords = pd.read_excel("C:\\Users\\Isabelle\\Documents\\BEP\\Maskeren\\wordFrequency.
       xlsx", sheet_name=1)
66 #lijst gebaseerd op COCA wordt ingeladen
67
68 #print(frequentwords.columns)
69
70 lemma_kolom = frequentwords['lemma']
71
72 #k = 1000 #deze parameter is met de hand aan te passen
73
74
75 k_values = range(0, 5001, 100)
76 f1_scores = []
77
78 precisie_scores = []
79
80 recall_scores = []
81
82
83 for k in k_values:
84     if k == 0:
85         f1_scores.append(0)  # Als k=0, sla over (geen features mogelijk)
86         precisie_scores.append(0)
87         recall_scores.append(0)
88         continue
89
90     W_k = lemma_kolom.head(k).tolist()
91
92     #print(W_k)
93
94     def vervang_woorden_tekst(string, W_k):
95         #woorden_string = ' '.join(string)
96         woorden = string.split()
97         nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in
       woorden]
98         return ' '.join(nieuwe_woorden)
99
100    final_teksten = {}
101
102    # TEST VOOR NIEUWE MANIER
103    for auteur, teksten in gecorrigeerde_teksten.items():
104        final_teksten[auteur] = []
105        for tekst in teksten:
106            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))
107
108    #printen om te checken
```

```
    #for auteur, teksten in final_teksten.items():
    #    print(f"Auteur: {auteur}")
    #    for tekst in teksten:
    #        print(f"  Tekst: {tekst}")

    rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
    for tweet in tweets]
    df = pd.DataFrame(rows)

    # Import train_test_split function
    from sklearn.model_selection import train_test_split

    x = df['tweet']
    y = df['auteur']
    # print(x)


    vectorizer = TfidfVectorizer()

    X_tfidf = vectorizer.fit_transform(x)


    # Split dataset into training set and test set
    tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    #Import svm model
    from sklearn import svm

    #Create a svm Classifier
    clf = svm.SVC(kernel='linear') # Linear Kernel

    #Train the model using the training sets
    clf.fit(tweet_train, author_train)

    #Predict the response for test dataset
    author_pred = clf.predict(tweet_test)

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    f1_scores.append(f1score)
    precisie_scores.append(precisiescore)
    recall_scores.append(recallscore)

# Model Accuracy: how often is the classifier correct?
#print("Accuraatheid:",metrics.accuracy_score(author_test, author_pred))

#print("Macroprecisie:",metrics.precision_score(author_test, author_pred, average='macro')) #
    bij average kan je micro, macro of weighted instellen
#print("Gevoeligheid:",metrics.recall_score(author_test, author_pred, average='macro')) #ook
    hier bij average micro, macro of weighted kiezen
#print("F1-score:",metrics.f1_score(author_test, author_pred, average='macro'))

reference_f1 = 0.8179987553835479
reference_precision = 0.8245307292305608
reference_recall = 0.8137992278192308


plt.figure(figsize=(10, 6))
plt.plot(k_values, f1_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_f1, color='r', linestyle='--', label=f'Ref. F1 = {reference_f1:.3f}')
plt.title("Tweets dataset COCA-maskering F1-score uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("F1-score")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, precisie_scores, marker='o', linestyle='-', color='b')
```

```
plt.axhline(y=reference_precision, color='r', linestyle='--', label=f'Ref. precisie = {reference_precision:.3f}')
plt.title("Tweets dataset COCA-maskering macroprecisie uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("Macroprecisie")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, recall_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_recall, color='r', linestyle='--', label=f'Ref. recall = {reference_recall:.3f}')
plt.title("Tweets dataset COCA-maskering macrorecall uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("Macrorecall")
plt.legend()
plt.grid()
plt.show()
```

In de volgende code wordt eigen maskering in combinatie met SVM op de Tweets dataset toegepast. Ook dit wordt voor verschillende waarden van k gedaan zodat er grafieken uit komen.

```
import pandas as pd
from tabulate import tabulate
data_tweets = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Celebrity_Tweets\\tweet_with_authors.csv", delimiter=",", error_bad_lines=False, header=0, encoding="utf-8")
# data wordt ingelezen, gescheiden op komma, met juiste encoding

import numpy as np
import matplotlib.pyplot as plt
from spellchecker import SpellChecker

spel = SpellChecker()

filter_rt= data_tweets['tweet'].str.contains("RT")
data_tweets = data_tweets[~filter_rt] #tweets die een retweet zijn worden eruit gehaald

import re
import emoji
from tfIdfInheritVectorizer.feature_extraction.vectorizer import TfidfVectorizer


def remove_emojis(text):
    return emoji.replace_emoji(text, replace='')

data_tweets['tweet']=data_tweets['tweet'].apply(remove_emojis) #emojis uit de tweet verwijderd

def clean_text(text):
    # Verwijder woorden die starten met # of http
    text = re.sub(r'\s#\S+|\b#\S+|\shttp\S+|\bhttp\S+', '', text)
    # Verwijder tekst tussen dubbele aanhalingstekens
    text = re.sub(r'".*?"', '', text)
    # Verwijder overbodige spaties
    text = ' '.join(text.split())
    return text

data_tweets['tweet']=data_tweets['tweet'].apply(clean_text)


#print(tabulate(data_tweets))
#print(data_tweets.columns)


auteurs_groep = data_tweets.groupby('author')['tweet'].apply(list).to_dict() #dictionary met auteursgroepen

gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

```

```
for auteur, teksten in auteurs_groep.items():
    gecorrigeerde_teksten[auteur] = []
    for tekst in teksten:
        woorden = tekst.split()
        gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [woord] for woord in woorden]
        gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in gecorrigeerde_woorden]))
#spelling gecorrigeerd

#printen om te checken
#for auteur, teksten in gecorrigeerde_teksten.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

#keys = gecorrigeerde_teksten.keys()
#print(keys)

from collections import Counter

#k = 200

k_values = range(0, 5001, 100)
f1_scores = []
precisie_scores = []
recall_scores = []

for k in k_values:
    if k == 0:
        f1_scores.append(0)  # Als k=0, sla over (geen features mogelijk)
        precisie_scores.append(0)
        recall_scores.append(0)
        continue

    word_counter = Counter()

    for tweets in gecorrigeerde_teksten.values():
        for tweet in tweets:
            # Split de tweet in woorden en tel ze
            geenleestekens_words = re.sub(r'[^\w\s-]', '', tweet.lower())
            words = geenleestekens_words.split()
            words = [word.lstrip('-').rstrip('-') for word in words]
            word_counter.update(words)

    frequentwords = word_counter.most_common(k)

    # Print de gesorteerde lijst van woorden en hun frequentie
    #for word, count in frequentwords[:100]:
    #    print(f"{word} {count}")

    W_k = [word for word, _ in frequentwords]
    #print(W_k)

    def vervang_woorden_tekst(string, W_k):
    #woorden_string = ' '.join(string)
        woorden = string.split()
        nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in woorden]
        return ' '.join(nieuwe_woorden)

    final_teksten = {}

    # TEST VOOR NIEUWE MANIER
    for auteur, teksten in gecorrigeerde_teksten.items():
        final_teksten[auteur] = []
        for tekst in teksten:
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    #printen om te checken
    #for auteur, teksten in final_teksten.items():
```

```
    #     print(f"Auteur: {auteur}")
    #     for tekst in teksten:
    #         print(f" Tekst: {tekst}")


    rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
    for tweet in tweets]
    df = pd.DataFrame(rows)

    # Import train_test_split function
    from sklearn.model_selection import train_test_split

    x = df['tweet']
    y = df['auteur']
    # print(x)


    vectorizer = TfidfVectorizer()
    X_tfidf = vectorizer.fit_transform(x)


    # Split dataset into training set and test set
    tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    #Import svm model
    from sklearn import svm

    #Create a svm Classifier
    clf = svm.SVC(kernel='linear') # Linear Kernel

    #Train the model using the training sets
    clf.fit(tweet_train, author_train)

    #Predict the response for test dataset
    author_pred = clf.predict(tweet_test)

    #Import scikit-learn metrics module for accuracy calculation
    from sklearn import metrics

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    f1_scores.append(f1score)
    precisie_scores.append(precisiescore)
    recall_scores.append(recallscore)




    # Model Accuracy: how often is the classifier correct?
    #print("Accuraatheid:",metrics.accuracy_score(author_test, author_pred))

    #print("Macroprecisie:",metrics.precision_score(author_test, author_pred, average='macro
    ')) #bij average kan je micro, macro of weighted instellen
    #print("Gevoeligheid:",metrics.recall_score(author_test, author_pred, average='macro')) #
    ook hier bij average micro, macro of weighted kiezen
    #print("F1-score:",metrics.f1_score(author_test, author_pred, average='macro'))

reference_f1 = 0.8179987553835479
reference_precision = 0.8245307292305608
reference_recall = 0.8137992278192308


plt.figure(figsize=(10, 6))
plt.plot(k_values, f1_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_f1, color='r', linestyle='--', label=f'Ref. F1 = {reference_f1:.3f}')
plt.title("Tweets dataset eigen maskering F1-score uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("F1-score")
plt.legend()
```

```
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, precisie_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_precision, color='r', linestyle='--', label=f'Ref. precisie = {reference_precision:.3f}')
plt.title("Tweets dataset eigen maskering macroprecisie uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("Macroprecisie")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, recall_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_recall, color='r', linestyle='--', label=f'Ref. recall = {reference_recall:.3f}')
plt.title("Tweets dataset eigen maskering macrorecall uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("Macrorecall")
plt.legend()
plt.grid()
plt.show()
```

In de volgende code wordt COCA-maskering in combinatie met logistische regressie op de Tweets dataset toegepast en worden hier grafieken van gemaakt:

```
import pandas as pd
from tabulate import tabulate
data_tweets = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Celebrity_Tweets\\tweet_with_authors.csv", delimiter=",", error_bad_lines=False, header=0, encoding="utf-8")
# data wordt ingelezen, gescheiden op komma, met juiste encoding

import numpy as np
import matplotlib.pyplot as plt
 #Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

from spellchecker import SpellChecker

spel = SpellChecker()

filter_rt= data_tweets['tweet'].str.contains("RT")
data_tweets = data_tweets[~filter_rt] #tweets die een retweet zijn worden eruit gehaald

import re
import emoji

def remove_emojis(text):
    return emoji.replace_emoji(text, replace='')

data_tweets['tweet']=data_tweets['tweet'].apply(remove_emojis) #emojis uit de tweet verwijderd

def clean_text(text):
    # Verwijder woorden die starten met # of http
    text = re.sub(r'\s#\S+|\b#\S+|\shttp\S+|\bhttp\S+', '', text)
    # Verwijder tekst tussen dubbele aanhalingstekens
    text = re.sub(r'".*?"', '', text)
    # Verwijder overbodige spaties
    text = ' '.join(text.split())
    return text

data_tweets['tweet']=data_tweets['tweet'].apply(clean_text)

```

```
41
42 #print(tabulate(data_tweets))
43 #print(data_tweets.columns)
44
45
46 auteurs_groep = data_tweets.groupby('author')['tweet'].apply(list).to_dict() #dictionary met
       auteursgroepen
47
48 gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten
49
50 for auteur, teksten in auteurs_groep.items():
51     gecorrigeerde_teksten[auteur] = []
52     for tekst in teksten:
53         woorden = tekst.split()
54         gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [
       woord] for woord in woorden]
55         gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in
       gecorrigeerde_woorden]))
56 #spelling gecorrigeerd
57
58 #printen om te checken
59 #for auteur, teksten in gecorrigeerde_teksten.items():
60 #    print(f"Auteur: {auteur}")
61 #    for tekst in teksten:
62 #        print(f"  Tekst: {tekst}")
63
64 #keys = gecorrigeerde_teksten.keys()
65 #print(keys)
66
67 frequentwords = pd.read_excel("C:\\Users\\Isabelle\\Documents\\BEP\\Maskeren\\wordFrequency.
       xlsx", sheet_name=1)
68 #lijst gebaseerd op COCA wordt ingeladen
69
70 #print(frequentwords.columns)
71
72 lemma_kolom = frequentwords['lemma']
73
74 #k = 1000 #deze parameter is met de hand aan te passen
75
76
77 k_values = range(0, 5000, 100)
78 f1_scores = []
79 precisie_scores = []
80 recall_scores = []
81
82 #from sklearn.feature_extraction.text import CountVectorizer
83
84 for k in k_values:
85     if k == 0:
86         f1_scores.append(0)  # Als k=0, sla over (geen features mogelijk)
87         precisie_scores.append(0)
88         recall_scores.append(0)
89         continue
90
91     W_k = lemma_kolom.head(k).tolist()
92
93     #print(W_k)
94
95     def vervang_woorden_tekst(string, W_k):
96         #woorden_string = ' '.join(string)
97         woorden = string.split()
98         nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in
       woorden]
99         return ' '.join(nieuwe_woorden)
100
101     final_teksten = {}
102
103     # TEST VOOR NIEUWE MANIER
104     for auteur, teksten in gecorrigeerde_teksten.items():
105         final_teksten[auteur] = []
106         for tekst in teksten:
```

```
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    #printen om te checken
    #for auteur, teksten in final_teksten.items():
    #    print(f"Auteur: {auteur}")
    #    for tekst in teksten:
    #        print(f"  Tekst: {tekst}")

    rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
    for tweet in tweets]
    df = pd.DataFrame(rows)

    x = df['tweet']
    y = df['auteur']


    vectorizer = CountVectorizer(stop_words=None)
    X_tfidf = vectorizer.fit_transform(x)


    # Split dataset into training set and test set
    tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    model = LogisticRegression()
    model.fit(tweet_train, author_train)

    author_pred = model.predict(tweet_test)

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    f1_scores.append(f1score)
    precisie_scores.append(precisiescore)
    recall_scores.append(recallscore)

# Model Accuracy: how often is the classifier correct?
#print("Accuraatheid:",metrics.accuracy_score(author_test, author_pred))

#print("Macroprecisie:",metrics.precision_score(author_test, author_pred, average='macro')) #
    bij average kan je micro, macro of weighted instellen
#print("Gevoeligheid:",metrics.recall_score(author_test, author_pred, average='macro')) #ook
    hier bij average micro, macro of weighted kiezen
#print("F1-score:",metrics.f1_score(author_test, author_pred, average='macro'))

reference_f1 = 0.8180268282292176
reference_precision = 0.8231712224957434
reference_recall = 0.81478385564269


plt.figure(figsize=(10, 6))
plt.plot(k_values, f1_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_f1, color='m', linestyle='--', label=f'Ref. F1 = {reference_f1:.3f}')
plt.title("Tweets dataset COCA-maskering F1-score uitgezet tegen k (logistische regressie)")
plt.xlabel("k")
plt.ylabel("F1-score")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, precisie_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_precision, color='m', linestyle='--', label=f'Ref. precisie = {
    reference_precision:.3f}')
plt.title("Tweets dataset COCA-maskering macroprecisie uitgezet tegen k (logistische
    regressie)")
plt.xlabel("k")
plt.ylabel("Macroprecisie")
plt.legend()
plt.grid()
plt.show()
```

```
172
173 plt.figure(figsize=(10, 6))
174 plt.plot(k_values, recall_scores, marker='o', linestyle='-', color='g')
175 plt.axhline(y=reference_recall, color='m', linestyle='--', label=f'Ref. recall = {
        reference_recall:.3f}')
176 plt.title("Tweets dataset COCA-maskering macrorecall uitgezet tegen k (logistische regressie)
        ")
177 plt.xlabel("k")
178 plt.ylabel("Macrorecall")
179 plt.legend()
180 plt.grid()
181 plt.show()
```

In de volgende code wordt eigen maskering in combinatie met logistische regressie op de Tweets dataset toegepast en worden hier grafieken van gemaakt:

```
1     import pandas as pd
2 from tabulate import tabulate
3 data_tweets = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Celebrity_Tweets\\
        tweet_with_authors.csv", delimiter=",", error_bad_lines=False, header=0, encoding="utf-8"
        )
4 # data wordt ingelezen, gescheiden op komma, met juiste encoding
5
6 import numpy as np
7 import matplotlib.pyplot as plt
8 from spellchecker import SpellChecker
9 from sklearn.feature_extraction.text import CountVectorizer
10 from sklearn.model_selection import train_test_split
11 from sklearn.linear_model import LogisticRegression
12 from sklearn.metrics import classification_report
13 from sklearn import metrics
14
15 spel = SpellChecker()
16
17 filter_rt= data_tweets['tweet'].str.contains("RT")
18 data_tweets = data_tweets[~filter_rt] #tweets die een retweet zijn worden eruit gehaald
19
20 import re
21 import emoji
22
23 def remove_emojis(text):
24     return emoji.replace_emoji(text, replace='')
25
26 data_tweets['tweet']=data_tweets['tweet'].apply(remove_emojis) #emojis uit de tweet
        verwijderd
27
28 def clean_text(text):
29     # Verwijder woorden die starten met # of http
30     text = re.sub(r'\s#\S+|\b#\S+|\shttp\S+|\bhttp\S+', '', text)
31     # Verwijder tekst tussen dubbele aanhalingstekens
32     text = re.sub(r'".*?"', '', text)
33     # Verwijder overbodige spaties
34     text = ' '.join(text.split())
35     return text
36
37 data_tweets['tweet']=data_tweets['tweet'].apply(clean_text)
38
39
40 #print(tabulate(data_tweets))
41 #print(data_tweets.columns)
42
43
44 auteurs_groep = data_tweets.groupby('author')['tweet'].apply(list).to_dict() #dictionary met
        auteursgroepen
45
46 gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten
47
48 for auteur, teksten in auteurs_groep.items():
49     gecorrigeerde_teksten[auteur] = []
50     for tekst in teksten:
51         woorden = tekst.split()
```

```
        gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [woord] for woord in woorden]
        gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in gecorrigeerde_woorden]))
#spelling gecorrigeerd

#printen om te checken
#for auteur, teksten in gecorrigeerde_teksten.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

#keys = gecorrigeerde_teksten.keys()
#print(keys)

from collections import Counter

#k = 200

k_values = range(0, 5001, 100)
f1_scores = []
precisie_scores = []
recall_scores = []

for k in k_values:
    if k == 0:
        f1_scores.append(0)  # Als k=0, sla over (geen features mogelijk)
        precisie_scores.append(0)
        recall_scores.append(0)
        continue

    word_counter = Counter()

    for tweets in gecorrigeerde_teksten.values():
        for tweet in tweets:
            # Split de tweet in woorden en tel ze
            geenleestekens_words = re.sub(r'[^\w\s-]', '', tweet.lower())
            words = geenleestekens_words.split()
            words = [word.lstrip('-').rstrip('-') for word in words]
            word_counter.update(words)

    frequentwords = word_counter.most_common(k)

    # Print de gesorteerde lijst van woorden en hun frequentie
    #for word, count in frequentwords[:100]:
    #    print(f"{word} {count}")

    W_k = [word for word, _ in frequentwords]
    #print(W_k)

    def vervang_woorden_tekst(string, W_k):
    #woorden_string = ' '.join(string)
        woorden = string.split()
        nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in woorden]
        return ' '.join(nieuwe_woorden)

    final_teksten = {}

    # TEST VOOR NIEUWE MANIER
    for auteur, teksten in gecorrigeerde_teksten.items():
        final_teksten[auteur] = []
        for tekst in teksten:
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    #printen om te checken
    #for auteur, teksten in final_teksten.items():
    #    print(f"Auteur: {auteur}")
    #    for tekst in teksten:
    #        print(f" Tekst: {tekst}")

```

```
120
121     rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
        for tweet in tweets]
122     df = pd.DataFrame(rows)
123
124     x = df['tweet']
125     y = df['auteur']
126     # print(x)
127
128
129     #vectorizer = TfidfVectorizer()
130     vectorizer = CountVectorizer(stop_words=None)
131     X_tfidf = vectorizer.fit_transform(x)
132
133     # Split dataset into training set and test set
134     tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
        test_size=0.3,random_state=109) # 70% training and 30% test
135
136     model = LogisticRegression()
137     model.fit(tweet_train, author_train)
138
139     author_pred = model.predict(tweet_test)
140
141     f1score = metrics.f1_score(author_test, author_pred, average='macro')
142     precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
143     recallscore = metrics.recall_score(author_test, author_pred, average='macro')
144     f1_scores.append(f1score)
145     precisie_scores.append(precisiescore)
146     recall_scores.append(recallscore)
147
148 reference_f1 = 0.8180268282292176
149 reference_precision = 0.8231712224957434
150 reference_recall = 0.81478385564269
151
152
153 plt.figure(figsize=(10, 6))
154 plt.plot(k_values, f1_scores, marker='o', linestyle='-', color='g')
155 plt.axhline(y=reference_f1, color='m', linestyle='--', label=f'Ref. F1 = {reference_f1:.3f}')
156 plt.title("Tweets dataset eigen maskering F1-score uitgezet tegen k (logistische regressie)")
157 plt.xlabel("k")
158 plt.ylabel("F1-score")
159 plt.legend()
160 plt.grid()
161 plt.show()
162
163 plt.figure(figsize=(10, 6))
164 plt.plot(k_values, precisie_scores, marker='o', linestyle='-', color='g')
165 plt.axhline(y=reference_precision, color='m', linestyle='--', label=f'Ref. precisie = {
        reference_precision:.3f}')
166 plt.title("Tweets dataset eigen maskering macroprecisie uitgezet tegen k (logistische
        regressie)")
167 plt.xlabel("k")
168 plt.ylabel("Macroprecisie")
169 plt.legend()
170 plt.grid()
171 plt.show()
172
173 plt.figure(figsize=(10, 6))
174 plt.plot(k_values, recall_scores, marker='o', linestyle='-', color='g')
175 plt.axhline(y=reference_recall, color='m', linestyle='--', label=f'Ref. recall = {
        reference_recall:.3f}')
176 plt.title("Tweets dataset eigen maskering macrorecall uitgezet tegen k (logistische regressie
        )")
177 plt.xlabel("k")
178 plt.ylabel("Macrorecall")
179 plt.legend()
180 plt.grid()
181 plt.show()
```

Onderstaande code is gebruikt om onderwerpen uit de Tweets dataset te extraheren op basis van de

hashtags:

```
import pandas as pd
from tabulate import tabulate
data_tweets = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Celebrity_Tweets\\tweet_with_authors.csv", delimiter=",", error_bad_lines=False, header=0, encoding="utf-8")
# data wordt ingelezen, gescheiden op komma, met juiste encoding

from spellchecker import SpellChecker

spel = SpellChecker()

filter_rt= data_tweets['tweet'].str.contains("RT")
data_tweets = data_tweets[~filter_rt] #tweets die een retweet zijn worden eruit gehaald

import re
import emoji

def remove_emojis(text):
    return emoji.replace_emoji(text, replace='')

data_tweets['tweet']=data_tweets['tweet'].apply(remove_emojis) #emojis uit de tweet verwijderd

def clean_text(text):
    # Verwijder woorden die starten met http
    text = re.sub(r'\shttp\S+|\bhttp\S+', '', text)
    # Verwijder tekst tussen dubbele aanhalingstekens
    text = re.sub(r'".*?"', '', text)
    # Verwijder overbodige spaties
    text = ' '.join(text.split())
    return text

data_tweets['tweet']=data_tweets['tweet'].apply(clean_text)


#print(tabulate(data_tweets))
#print(data_tweets.columns)


auteurs_groep = data_tweets.groupby('author')['tweet'].apply(list).to_dict() #dictionary met auteursgroepen

gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

for auteur, teksten in auteurs_groep.items():
    gecorrigeerde_teksten[auteur] = []
    for tekst in teksten:
        gecorrigeerde_teksten[auteur].append(tekst)

#for auteur, teksten in gecorrigeerde_teksten.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

import re
from collections import Counter

hashtag_counter = Counter()

for auteur, tekst in gecorrigeerde_teksten.items():
        tekst = str(tekst)
        # Vind alle woorden die beginnen met #
        hashtags = re.findall(r'#\w+', tekst)
        hashtags_lower = [hashtag.lower() for hashtag in hashtags]
        # Voeg ze toe aan de Counter
        hashtag_counter.update(hashtags_lower)

# Print het resultaat
#print("Hashtags en frequenties:", hashtag_counter)
```

```

# De voorwaarden
conditions = {
    "condition1": {"#actonclimate"},
    "condition2": {"#americanidol"},
    "condition3": {"#nlproc", "#deeplearning", "#machinelearning"},
    "condition4": {"#getcovered", "#obamacare"},
}

# Functie om de hashtags uit een tekst te extraheren
def extract_hashtags(tekst):
    # Zet de tekst om naar kleine letters en zoek naar hashtags
    hashtags = re.findall(r'#\w+', tekst.lower())  # #\w+ zoekt naar woorden die beginnen met #
    return set(hashtags)

# Functie om te controleren of een auteur aan de voorwaarden voldoet
def check_conditions(teksten, conditions):
    results = {key: False for key in conditions}  # Initieer alle voorwaarden als False
    for tekst in teksten:
        hashtags = extract_hashtags(tekst)  # Haal hashtags uit de tekst
        print(f"Tekst: '{tekst}'")
        print(f"Hashtags in tekst: {hashtags}")
        for condition, required_hashtags in conditions.items():
            print(f"Controleren op {condition}: {required_hashtags}")
            if required_hashtags & hashtags:  # Controleer of er een overlap is
                results[condition] = True
    return results

auteur_results = {}
for auteur, teksten in gecorrigeerde_teksten.items():
    auteur_results[auteur] = check_conditions(teksten, conditions)

# Controleer of elke auteur aan alle voorwaarden voldoet
alle_auteurs_voldoen = all(all(v for v in res.values()) for res in auteur_results.values())

# Print resultaten
print("Individuele resultaten per auteur:")
for auteur, res in auteur_results.items():
    print(f"{auteur}: {res}")


print(f"\nAlle auteurs voldoen aan de voorwaarden: {alle_auteurs_voldoen}")
```

De onderstaande code is gebruikt om de gecombineerde grafieken voor de Tweets dataset te maken:

```
from tabulate import tabulate
from sklearn import metrics
from spellchecker import SpellChecker
from sklearn import svm
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
import re
import emoji
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import svm
from sklearn import metrics
from collections import Counter
from tfIdfInheritVectorizer.feature_extraction.vectorizer import TfidfVectorizer


def remove_emojis(text):
    return emoji.replace_emoji(text, replace='')

def clean_text(text):
```

```
    # Verwijder woorden die starten met # of http
    text = re.sub(r'\s#\S+|\b#\S+|\shttp\S+|\bhttp\S+', '', text)
    # Verwijder tekst tussen dubbele aanhalingstekens
    text = re.sub(r'".*?"', '', text)
    # Verwijder overbodige spaties
    text = ' '.join(text.split())
    return text

def creeer_gecorrigeerde_teksten(data_tweets):
    spel = SpellChecker()
    filter_rt= data_tweets['tweet'].str.contains("RT")
    data_tweets = data_tweets[~filter_rt] #tweets die een retweet zijn worden eruit gehaald
    data_tweets['tweet']=data_tweets['tweet'].apply(remove_emojis) #emojis uit de tweet
    verwijderd
    data_tweets['tweet']=data_tweets['tweet'].apply(clean_text)

    auteurs_groep = data_tweets.groupby('author')['tweet'].apply(list).to_dict() #dictionary
    met auteursgroepen

    gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

    for auteur, teksten in auteurs_groep.items():
        gecorrigeerde_teksten[auteur] = []
        for tekst in teksten:
            woorden = tekst.split()
            gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else
     [woord] for woord in woorden]
            gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in
    gecorrigeerde_woorden]))
    return gecorrigeerde_teksten

def vervang_woorden_tekst(string, W_k):
    #woorden_string = ' '.join(string)
    woorden = string.split()
    nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in
    woorden]
    return ' '.join(nieuwe_woorden)


def svm_coca(k, gecorrigeerde_teksten, lemma_kolom):
    if k == 0:
        return 0,0,0

    W_k = lemma_kolom.head(k).tolist()

    final_teksten = {}

    for auteur, teksten in gecorrigeerde_teksten.items():
        final_teksten[auteur] = []
        for tekst in teksten:
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
    for tweet in tweets]
    df = pd.DataFrame(rows)

    x = df['tweet']
    y = df['auteur']

    vectorizer = TfidfVectorizer()
    X_tfidf = vectorizer.fit_transform(x)

    # Split dataset into training set and test set
    tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    #Create a svm Classifier
    clf = svm.SVC(kernel='linear') # Linear Kernel

    #Train the model using the training sets
    clf.fit(tweet_train, author_train)
```

```
90
91      #Predict the response for test dataset
92      author_pred = clf.predict(tweet_test)
93
94      f1score = metrics.f1_score(author_test, author_pred, average='macro')
95      precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
96      recallscore = metrics.recall_score(author_test, author_pred, average='macro')
97      return f1score, precisiescore, recallscore
98
99  def logreg_coca(k, gecorrigeerde_teksten, lemma_kolom):
100     if k == 0:
101         return 0,0,0
102
103     W_k = lemma_kolom.head(k).tolist()
104
105     final_teksten = {}
106
107     # TEST VOOR NIEUWE MANIER
108     for auteur, teksten in gecorrigeerde_teksten.items():
109         final_teksten[auteur] = []
110         for tekst in teksten:
111             final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))
112
113     rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
        for tweet in tweets]
114     df = pd.DataFrame(rows)
115
116     x = df['tweet']
117     y = df['auteur']
118
119     vectorizer = CountVectorizer(stop_words=None)
120     X_tfidf = vectorizer.fit_transform(x)
121
122     # Split dataset into training set and test set
123     tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
        test_size=0.3,random_state=109) # 70% training and 30% test
124
125     model = LogisticRegression()
126     model.fit(tweet_train, author_train)
127
128     author_pred = model.predict(tweet_test)
129
130     f1score = metrics.f1_score(author_test, author_pred, average='macro')
131     precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
132     recallscore = metrics.recall_score(author_test, author_pred, average='macro')
133     return f1score,precisiescore,recallscore
134
135 def svm_eigen(k, gecorrigeerde_teksten):
136     if k == 0:
137         return 0,0,0
138
139     word_counter = Counter()
140
141     for tweets in gecorrigeerde_teksten.values():
142         for tweet in tweets:
143             # Split de tweet in woorden en tel ze
144             geenleestekens_words = re.sub(r'[^\w\s-]', '', tweet.lower())
145             words = geenleestekens_words.split()
146             words = [word.lstrip('-').rstrip('-') for word in words]
147             word_counter.update(words)
148
149     frequentwords = word_counter.most_common(k)
150     W_k = [word for word, _ in frequentwords]
151
152     final_teksten = {}
153
154     for auteur, teksten in gecorrigeerde_teksten.items():
155         final_teksten[auteur] = []
156         for tekst in teksten:
157             final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))
158
```

```
    rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
    for tweet in tweets]
    df = pd.DataFrame(rows)

    x = df['tweet']
    y = df['auteur']

    vectorizer = TfidfVectorizer()
    X_tfidf = vectorizer.fit_transform(x)

    # Split dataset into training set and test set
    tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    #Create a svm Classifier
    clf = svm.SVC(kernel='linear') # Linear Kernel

    #Train the model using the training sets
    clf.fit(tweet_train, author_train)

    #Predict the response for test dataset
    author_pred = clf.predict(tweet_test)

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    return f1score,precisiescore,recallscore

def logreg_eigen(k, gecorrigeerde_teksten):
    if k == 0:
        return 0,0,0

    word_counter = Counter()

    for tweets in gecorrigeerde_teksten.values():
        for tweet in tweets:
            # Split de tweet in woorden en tel ze
            geenleestekens_words = re.sub(r'[^\w\s-]', '', tweet.lower())
            words = geenleestekens_words.split()
            words = [word.lstrip('-').rstrip('-') for word in words]
            word_counter.update(words)

    frequentwords = word_counter.most_common(k)

    W_k = [word for word, _ in frequentwords]

    final_teksten = {}

    # TEST VOOR NIEUWE MANIER
    for auteur, teksten in gecorrigeerde_teksten.items():
        final_teksten[auteur] = []
        for tekst in teksten:
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
    for tweet in tweets]
    df = pd.DataFrame(rows)

    x = df['tweet']
    y = df['auteur']

    vectorizer = CountVectorizer(stop_words=None)
    X_tfidf = vectorizer.fit_transform(x)

    # Split dataset into training set and test set
    tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    model = LogisticRegression()
    model.fit(tweet_train, author_train)

```

```
    author_pred = model.predict(tweet_test)

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    return f1score,precisiescore,recallscore

# Gebruiken we bij COCA Maskering
def svm_coca_resultaten(k_values,gecorrigeerde_teksten, frequentwords):

    lemma_kolom = frequentwords['lemma']

    f1_scores = []
    precisie_scores = []
    recall_scores = []

    for k in k_values:
        f1_score,precisie_score,recall_score = svm_coca(k, gecorrigeerde_teksten, lemma_kolom
    )
        f1_scores.append(f1_score)
        precisie_scores.append(precisie_score)
        recall_scores.append(recall_score)
    return f1_scores, precisie_scores, recall_scores


def logreg_coca_resultaten(k_values,gecorrigeerde_teksten, frequentwords):
    lemma_kolom = frequentwords['lemma']


    f1_scores = []
    precisie_scores = []
    recall_scores = []

    for k in k_values:
        f1_score,precisie_score,recall_score = logreg_coca(k, gecorrigeerde_teksten,
    lemma_kolom)
        f1_scores.append(f1_score)
        precisie_scores.append(precisie_score)
        recall_scores.append(recall_score)
    return f1_scores, precisie_scores, recall_scores

def svm_eigen_resultaten(k_values,gecorrigeerde_teksten):
    f1_scores = []
    precisie_scores = []
    recall_scores = []

    for k in k_values:
        f1_score,precisie_score,recall_score = svm_eigen(k, gecorrigeerde_teksten)
        f1_scores.append(f1_score)
        precisie_scores.append(precisie_score)
        recall_scores.append(recall_score)
    return f1_scores, precisie_scores, recall_scores

def logreg_eigen_resultaten(k_values,gecorrigeerde_teksten):
    f1_scores = []
    precisie_scores = []
    recall_scores = []

    for k in k_values:
        f1_score,precisie_score,recall_score = logreg_eigen(k, gecorrigeerde_teksten)
        f1_scores.append(f1_score)
        precisie_scores.append(precisie_score)
        recall_scores.append(recall_score)
    return f1_scores, precisie_scores, recall_scores

data_tweets = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Celebrity_Tweets\\
    tweet_with_authors.csv", delimiter=",", error_bad_lines=False, header=0, encoding="utf-8"
    )
frequentwords = pd.read_excel("C:\\Users\\Isabelle\\Documents\\BEP\\Maskeren\\wordFrequency.
    xlsx", sheet_name=1)
k_values = range(0, 5001, 100)
```

```
292
293 gecorrigeerde_teksten = creeer_gecorrigeerde_teksten(data_tweets)
294 svm_coca_f1, svm_coca_precisie, svm_coca_recall = svm_coca_resultaten(k_values,
        gecorrigeerde_teksten, frequentwords)
295 logreg_coca_f1, logreg_coca_precisie, logreg_coca_recall = logreg_coca_resultaten(k_values,
        gecorrigeerde_teksten, frequentwords)
296 svm_eigen_f1, svm_eigen_precisie, svm_eigen_recall = svm_eigen_resultaten(k_values,
        gecorrigeerde_teksten)
297 logreg_eigen_f1, logreg_eigen_precisie, logreg_eigen_recall = logreg_eigen_resultaten(
        k_values,gecorrigeerde_teksten)
298
299 reference_f1_svm = 0.8179987553835479
300 reference_precision_svm = 0.8245307292305608
301 reference_recall_svm = 0.8137992278192308
302
303
304 reference_f1_logreg = 0.8180268282292176
305 reference_precision_logreg = 0.8231712224957434
306 reference_recall_logreg = 0.81478385564269
307
308 plt.figure(figsize=(10, 6))
309 plt.plot(k_values, svm_coca_f1, marker='o', linestyle='-', color='b')
310 plt.plot(k_values, logreg_coca_f1, marker='o', linestyle='-', color='g')
311 plt.plot(k_values, svm_eigen_f1, marker='o', linestyle='-', color='r')
312 plt.plot(k_values, logreg_eigen_f1, marker='o', linestyle='-', color='purple')
313
314 plt.axhline(y=reference_f1_svm, color='black', linestyle='--', label=f'Ref. F1 SVM = {
        reference_f1_svm:.3f}')
315 plt.axhline(y=reference_f1_logreg, color='black', linestyle='--', label=f'Ref. F1 LOGREG = {
        reference_f1_logreg:.3f}')
316 plt.title("Tweets dataset F1-score uitgezet tegen k")
317 plt.xlabel("k")
318 plt.ylabel("F1-score")
319 plt.legend()
320 plt.grid()
321 plt.show()
322
323 plt.figure(figsize=(10, 6))
324 plt.plot(k_values, svm_coca_precisie, marker='o', linestyle='-', color='b')
325 plt.plot(k_values, logreg_coca_precisie, marker='o', linestyle='-', color='g')
326 plt.plot(k_values, svm_eigen_precisie, marker='o', linestyle='-', color='r')
327 plt.plot(k_values, logreg_eigen_precisie, marker='o', linestyle='-', color='purple')
328
329 plt.axhline(y=reference_precision_svm, color='black', linestyle='--', label=f'Ref. precisie
        SVM = {reference_precision_svm:.3f}')
330 plt.axhline(y=reference_precision_logreg, color='black', linestyle='--', label=f'Ref.
        precisie LOGREG = {reference_precision_logreg:.3f}')
331 plt.title("Tweets dataset macroprecisie uitgezet tegen k")
332 plt.xlabel("k")
333 plt.ylabel("Macroprecisie")
334 plt.legend()
335 plt.grid()
336 plt.show()
337
338 plt.figure(figsize=(10, 6))
339 plt.plot(k_values, svm_coca_recall, marker='o', linestyle='-', color='b')
340 plt.plot(k_values, logreg_coca_recall, marker='o', linestyle='-', color='g')
341 plt.plot(k_values, svm_eigen_recall, marker='o', linestyle='-', color='r')
342 plt.plot(k_values, logreg_eigen_recall, marker='o', linestyle='-', color='purple')
343
344 plt.axhline(y=reference_recall_svm, color='black', linestyle='--', label=f'Ref. recall SVM =
        {reference_recall_svm:.3f}')
345 plt.axhline(y=reference_recall_logreg, color='black', linestyle='--', label=f'Ref. recall
        LOGREG = {reference_recall_logreg:.3f}')
346 plt.title("Tweets dataset macrorecall uitgezet tegen k")
347 plt.xlabel("k")
348 plt.ylabel("Macrorecall")
349 plt.legend()
350 plt.grid()
351 plt.show()
```

B

Pythoncodes Victoriaanse dataset

De eerste, onderstaande, code is gebruikt om de Victoriaanse dataset schoon te maken en SVM toe te passen zonder maskering:

```
import pandas as pd

data_victorian = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\18th_19th_Century_Victorian_Authors\\dataset\\dataset\\Gungor_2018_VictorianAuthorAttribution_data-train.csv", delimiter=",", error_bad_lines=False, header=0, encoding="latin-1")
# data wordt ingelezen, gescheiden op komma, met juiste encoding
data_victorian = data_victorian.sample(frac=0.25, random_state=1) #deel van data gebruiken om te kijken of het werkt? HIJ DOET HET!!! maar wel traag

from spellchecker import SpellChecker

spel = SpellChecker()

#print(data_victorian)
#print(data_victorian.columns)

auteurs_groep = data_victorian.groupby('author')['text'].apply(list).to_dict() #dictionary met auteursgroepen

gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

for auteur, teksten in auteurs_groep.items():
    gecorrigeerde_teksten[auteur] = []
    for tekst in teksten:
        woorden = tekst.split()
        gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [woord] for woord in woorden]
        gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in gecorrigeerde_woorden]))

#printen om te checken
#for auteur, teksten in gecorrigeerde_teksten.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

rows = [{'auteur': auteur, 'tekst': tekst} for auteur, teksten in gecorrigeerde_teksten.items() for tekst in teksten]
df = pd.DataFrame(rows)

# Import train_test_split function
from sklearn.model_selection import train_test_split

x = df['tekst']
y = df['auteur']
```

```
39
40 from tfIdfInheritVectorizer.feature_extraction.vectorizer import TfidfVectorizer
41
42 vectorizer = TfidfVectorizer()
43 X_tfidf = vectorizer.fit_transform(x)
44
45
46 # Split dataset into training set and test set
47 tekst_train, tekst_test, author_train, author_test = train_test_split(X_tfidf,y, test_size
       =0.3,random_state=109) # 70% training and 30% test
48
49 #Import svm model
50 from sklearn import svm
51
52 #Create a svm Classifier
53 clf = svm.SVC(kernel='linear') # Linear Kernel
54
55 #Train the model using the training sets
56 clf.fit(tekst_train, author_train)
57
58 #Predict the response for test dataset
59 author_pred = clf.predict(tekst_test)
60
61 #Import scikit-learn metrics module for accuracy calculation
62 from sklearn import metrics
63
64 # Model Accuracy: how often is the classifier correct?
65 print("Macroprecisie:",metrics.precision_score(author_test, author_pred, average='macro')) #
       bij average kan je micro, macro of weighted instellen
66 print("Gevoeligheid:",metrics.recall_score(author_test, author_pred, average='macro')) #ook
       hier bij average micro, macro of weighted kiezen
67 print("F1-score:",metrics.f1_score(author_test, author_pred, average='macro'))
```

De tweede code is gebruikt om de Victoriaanse dataset schoon te maken en logistische regressie toe te passen:

```
1 import pandas as pd
2
3 data_victorian = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\18
      th_19th_Century_Victorian_Authors\\dataset\\dataset\\
      Gungor_2018_VictorianAuthorAttribution_data-train.csv", delimiter=",", error_bad_lines=
      False, header=0, encoding="latin-1")
4 # data wordt ingelezen, gescheiden op komma, met juiste encoding
5 data_victorian = data_victorian.sample(frac=0.25, random_state=1) #deel van data gebruiken om
       te kijken of het werkt? HIJ DOET HET!!! maar wel traag
6
7 from sklearn.feature_extraction.text import CountVectorizer
8 from sklearn.model_selection import train_test_split
9 from sklearn.linear_model import LogisticRegression
10 from sklearn.metrics import classification_report
11
12 from spellchecker import SpellChecker
13
14 spel = SpellChecker()
15
16 #print(data_victorian)
17 #print(data_victorian.columns)
18
19 auteurs_groep = data_victorian.groupby('author')['text'].apply(list).to_dict() #dictionary
      met auteursgroepen
20
21 gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten
22
23 for auteur, teksten in auteurs_groep.items():
24     gecorrigeerde_teksten[auteur] = []
25     for tekst in teksten:
26         woorden = tekst.split()
27         gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [
      woord] for woord in woorden]
28         gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in
      gecorrigeerde_woorden]))
```

```
29
30 #printen om te checken
31 #for auteur, teksten in gecorrigeerde_teksten.items():
32 #    print(f"Auteur: {auteur}")
33 #    for tekst in teksten:
34 #        print(f"  Tekst: {tekst}")
35
36 rows = [{'auteur': auteur, 'tekst': tekst} for auteur, teksten in gecorrigeerde_teksten.items
       () for tekst in teksten]
37 df = pd.DataFrame(rows)
38
39 x = df['tekst']
40 y = df['auteur']
41
42
43 vectorizer = CountVectorizer(stop_words=None)
44 X_tfidf = vectorizer.fit_transform(x)
45
46
47 # Split dataset into training set and test set
48 tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y, test_size
       =0.3,random_state=109) # 70% training and 30% test
49
50 model = LogisticRegression()
51 model.fit(tweet_train, author_train)
52
53 author_pred = model.predict(tweet_test)
54 print(classification_report(author_test, author_pred))
55
56
57 #Import scikit-learn metrics module for accuracy calculation
58 from sklearn import metrics
59
60 # Model Accuracy: how often is the classifier correct?
61 #print("Accuraatheid:",metrics.accuracy_score(author_test, author_pred))
62 print("Macroprecisie:",metrics.precision_score(author_test, author_pred, average='macro')) #
       bij average kan je micro, macro of weighted instellen
63 print("Gevoeligheid:",metrics.recall_score(author_test, author_pred, average='macro')) #ook
       hier bij average micro, macro of weighted kiezen
64 print("F1-score:",metrics.f1_score(author_test, author_pred, average='macro'))
```

In de volgende code is COCA-maskering in combinatie met SVM toegepast op de Victoriaanse dataset en zijn hier grafieken van gemaakt:

```
1 import pandas as pd
2
3 data_victorian = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\18
      th_19th_Century_Victorian_Authors\\dataset\\dataset\\
      Gungor_2018_VictorianAuthorAttribution_data-train.csv", delimiter=",", error_bad_lines=
      False, header=0, encoding="latin-1")
4 # data wordt ingelezen, gescheiden op komma, met juiste encoding
5 data_victorian = data_victorian.sample(frac=0.25, random_state=1) #deel van data gebruiken om
       te kijken of het werkt? HIJ DOET HET!!! maar wel traag
6
7 from spellchecker import SpellChecker
8 import numpy as np
9 import matplotlib.pyplot as plt
10
11 spel = SpellChecker()
12
13 #print(data_victorian)
14 #print(data_victorian.columns)
15
16 auteurs_groep = data_victorian.groupby('author')['text'].apply(list).to_dict() #dictionary
      met auteursgroepen
17
18 gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten
19
20 for auteur, teksten in auteurs_groep.items():
21     gecorrigeerde_teksten[auteur] = []
22     for tekst in teksten:
```

```
        woorden = tekst.split()
        gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [woord] for woord in woorden]
        gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in gecorrigeerde_woorden]))

#printen om te checken
#for auteur, teksten in gecorrigeerde_teksten.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

frequentwords = pd.read_excel("C:\\Users\\Isabelle\\Documents\\BEP\\Maskeren\\wordFrequency.xlsx", sheet_name=1)
#lijst gebaseerd op COCA wordt ingeladen

#print(frequentwords.columns)

lemma_kolom = frequentwords['lemma']

#k = 1000 #deze parameter is met de hand aan te passen


k_values = range(0, 5001, 100)
f1_scores = []
precisie_scores = []
recall_scores = []

for k in k_values:
    if k == 0:
        f1_scores.append(0)  # Als k=0, sla over (geen features mogelijk)
        precisie_scores.append(0)
        recall_scores.append(0)
        continue

    W_k = lemma_kolom.head(k).tolist()

    #print(W_k)

    def vervang_woorden_tekst(string, W_k):
        #woorden_string = ' '.join(string)
        woorden = string.split()
        nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in woorden]
        return ' '.join(nieuwe_woorden)

    final_teksten = {}

    # TEST VOOR NIEUWE MANIER
    for auteur, teksten in gecorrigeerde_teksten.items():
        final_teksten[auteur] = []
        for tekst in teksten:
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    #printen om te checken
    #for auteur, teksten in final_teksten.items():
    #    print(f"Auteur: {auteur}")
    #    for tekst in teksten:
    #        print(f"  Tekst: {tekst}")

    rows = [{'auteur': auteur, 'tekst': tekst} for auteur, teksten in final_teksten.items() for tekst in teksten]
    df = pd.DataFrame(rows)

    # Import train_test_split function
    from sklearn.model_selection import train_test_split

    x = df['tekst']
    y = df['auteur']

    from tfIdfInheritVectorizer.feature_extraction.vectorizer import TfidfVectorizer
```

```
89
90      vectorizer = TfidfVectorizer()
91      X_tfidf = vectorizer.fit_transform(x)
92
93
94      # Split dataset into training set and test set
95      tekst_train, tekst_test, author_train, author_test = train_test_split(X_tfidf,y,
        test_size=0.3,random_state=109) # 70% training and 30% test
96
97      #Import svm model
98      from sklearn import svm
99
100     #Create a svm Classifier
101     clf = svm.SVC(kernel='linear') # Linear Kernel
102
103     #Train the model using the training sets
104     clf.fit(tekst_train, author_train)
105
106     #Predict the response for test dataset
107     author_pred = clf.predict(tekst_test)
108
109     #Import scikit-learn metrics module for accuracy calculation
110     from sklearn import metrics
111
112     # Model Accuracy: how often is the classifier correct?
113     #print("Accuraatheid:",metrics.accuracy_score(author_test, author_pred))
114     #print("Precisie:",metrics.precision_score(author_test, author_pred, average='macro')) #
        bij average kan je micro, macro of weighted instellen
115     #print("Gevoeligheid:",metrics.recall_score(author_test, author_pred, average='macro')) #
        ook hier bij average micro, macro of weighted kiezen
116
117     f1score = metrics.f1_score(author_test, author_pred, average='macro')
118     precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
119     recallscore = metrics.recall_score(author_test, author_pred, average='macro')
120     f1_scores.append(f1score)
121     precisie_scores.append(precisiescore)
122     recall_scores.append(recallscore)
123
124 reference_f1 = 0.8160282931172088
125 reference_precision = 0.932525676251512
126 reference_recall = 0.7652173468005774
127
128
129 plt.figure(figsize=(10, 6))
130 plt.plot(k_values, f1_scores, marker='o', linestyle='-', color='b')
131 plt.axhline(y=reference_f1, color='r', linestyle='--', label=f'Ref. F1 = {reference_f1:.3f}')
132 plt.title("Victoriaanse dataset COCA-maskering F1-score uitgezet tegen k")
133 plt.xlabel("k")
134 plt.ylabel("F1-score")
135 plt.legend()
136 plt.grid()
137 plt.show()
138
139 plt.figure(figsize=(10, 6))
140 plt.plot(k_values, precisie_scores, marker='o', linestyle='-', color='b')
141 plt.axhline(y=reference_precision, color='r', linestyle='--', label=f'Ref. precisie = {
        reference_precision:.3f}')
142 plt.title("Victoriaanse dataset COCA-maskering macroprecisie uitgezet tegen k")
143 plt.xlabel("k")
144 plt.ylabel("Macroprecisie")
145 plt.legend()
146 plt.grid()
147 plt.show()
148
149 plt.figure(figsize=(10, 6))
150 plt.plot(k_values, recall_scores, marker='o', linestyle='-', color='b')
151 plt.axhline(y=reference_recall, color='r', linestyle='--', label=f'Ref. recall = {
        reference_recall:.3f}')
152 plt.title("Victoriaanse dataset COCA-maskering macrorecall uitgezet tegen k")
153 plt.xlabel("k")
154 plt.ylabel("Macrorecall")
```

```
plt.legend()
plt.grid()
plt.show()
```

In de volgende code is eigen maskering in combinatie met SVM toegepast op de Victoriaanse dataset en zijn hier grafieken van gemaakt:

```
import pandas as pd
import re

data_victorian = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\18th_19th_Century_Victorian_Authors\\dataset\\dataset\\Gungor_2018_VictorianAuthorAttribution_data-train.csv", delimiter=",", error_bad_lines=False, header=0, encoding="latin-1")
# data wordt ingelezen, gescheiden op komma, met juiste encoding
data_victorian = data_victorian.sample(frac=0.25, random_state=1) #deel van data gebruiken om te kijken of het werkt? HIJ DOET HET!!! maar wel traag

from spellchecker import SpellChecker
import numpy as np
import matplotlib.pyplot as plt

spel = SpellChecker()

#print(data_victorian)
#print(data_victorian.columns)

auteurs_groep = data_victorian.groupby('author')['text'].apply(list).to_dict() #dictionary met auteursgroepen

gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

for auteur, teksten in auteurs_groep.items():
    gecorrigeerde_teksten[auteur] = []
    for tekst in teksten:
        woorden = tekst.split()
        gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [woord] for woord in woorden]
        gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in gecorrigeerde_woorden]))

from collections import Counter

#k = 200
k_values = range(0, 5001, 100)
f1_scores = []
precisie_scores = []
recall_scores = []

for k in k_values:
    if k == 0:
        f1_scores.append(0)  # Als k=0, sla over (geen features mogelijk)
        precisie_scores.append(0)
        recall_scores.append(0)
        continue

    word_counter = Counter()

    for tweets in gecorrigeerde_teksten.values():
        for tweet in tweets:
            # Split de tweet in woorden en tel ze
            geenleestekens_words = re.sub(r'[^\w\s-]', '', tweet.lower())
            words = geenleestekens_words.split()
            words = [word.lstrip('-').rstrip('-') for word in words]
            word_counter.update(words)

    frequentwords = word_counter.most_common(k)

    # Print de gesorteerde lijst van woorden en hun frequentie
    #for word, count in frequentwords[:100]:
    #    print(f"{word} {count}")
```

```
58
59      W_k = [word for word, _ in frequentwords]
60      #print(W_k)
61
62      def vervang_woorden_tekst(string, W_k):
63      #woorden_string = ' '.join(string)
64          woorden = string.split()
65          nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in
        woorden]
66          return ' '.join(nieuwe_woorden)
67
68      final_teksten = {}
69
70      # TEST VOOR NIEUWE MANIER
71      for auteur, teksten in gecorrigeerde_teksten.items():
72          final_teksten[auteur] = []
73          for tekst in teksten:
74              final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))
75
76      #printen om te checken
77      #for auteur, teksten in final_teksten.items():
78      #    print(f"Auteur: {auteur}")
79      #    for tekst in teksten:
80      #        print(f" Tekst: {tekst}")
81
82      rows = [{'auteur': auteur, 'tekst': tekst} for auteur, teksten in final_teksten.items()
        for tekst in teksten]
83      df = pd.DataFrame(rows)
84
85      # Import train_test_split function
86      from sklearn.model_selection import train_test_split
87
88      x = df['tekst']
89      y = df['auteur']
90
91      from tfIdfInheritVectorizer.feature_extraction.vectorizer import TfidfVectorizer
92
93      vectorizer = TfidfVectorizer()
94      X_tfidf = vectorizer.fit_transform(x)
95
96
97      # Split dataset into training set and test set
98      tekst_train, tekst_test, author_train, author_test = train_test_split(X_tfidf,y,
        test_size=0.3,random_state=109) # 70% training and 30% test
99
100     #Import svm model
101     from sklearn import svm
102
103     #Create a svm Classifier
104     clf = svm.SVC(kernel='linear') # Linear Kernel
105
106     #Train the model using the training sets
107     clf.fit(tekst_train, author_train)
108
109     #Predict the response for test dataset
110     author_pred = clf.predict(tekst_test)
111
112     #Import scikit-learn metrics module for accuracy calculation
113     from sklearn import metrics
114
115     f1score = metrics.f1_score(author_test, author_pred, average='macro')
116     precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
117     recallscore = metrics.recall_score(author_test, author_pred, average='macro')
118     f1_scores.append(f1score)
119     precisie_scores.append(precisiescore)
120     recall_scores.append(recallscore)
121
122     # Model Accuracy: how often is the classifier correct?
123     #print("Accuraatheid:",metrics.accuracy_score(author_test, author_pred))
124     #print("Precisie:",metrics.precision_score(author_test, author_pred, average='macro')) #
        bij average kan je micro, macro of weighted instellen
```

```
    #print("Gevoeligheid:",metrics.recall_score(author_test, author_pred, average='macro')) # ook hier bij average micro, macro of weighted kiezen

reference_f1 = 0.8160282931172088
reference_precision = 0.932525676251512
reference_recall = 0.7652173468005774

plt.figure(figsize=(10, 6))
plt.plot(k_values, f1_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_f1, color='r', linestyle='--', label=f'Ref. F1 = {reference_f1:.3f}')
plt.title("Victoriaanse dataset eigen maskering F1-score uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("F1-score")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, precisie_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_precision, color='r', linestyle='--', label=f'Ref. precisie = {reference_precision:.3f}')
plt.title("Victoriaanse dataset eigen maskering macroprecisie uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("Macroprecisie")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, recall_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_recall, color='r', linestyle='--', label=f'Ref. recall = {reference_recall:.3f}')
plt.title("Victoriaanse dataset eigen maskering macrorecall uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("Macrorecall")
plt.legend()
plt.grid()
plt.show()
```

In de volgende code is COCA-maskering in combinatie met logistische regressie toegepast voor de Victoriaanse dataset. Hier zijn wederom weer grafieken van gemaakt:

```
import pandas as pd

data_victorian = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\18th_19th_Century_Victorian_Authors\\dataset\\dataset\\Gungor_2018_VictorianAuthorAttribution_data-train.csv", delimiter=",", error_bad_lines=False, header=0, encoding="latin-1")
# data wordt ingelezen, gescheiden op komma, met juiste encoding
data_victorian = data_victorian.sample(frac=0.25, random_state=1) #deel van data gebruiken om te kijken of het werkt? HIJ DOET HET!!! maar wel traag

from spellchecker import SpellChecker
import numpy as np
import matplotlib.pyplot as plt
 #Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

spel = SpellChecker()

#print(data_victorian)
#print(data_victorian.columns)

auteurs_groep = data_victorian.groupby('author')['text'].apply(list).to_dict() #dictionary met auteursgroepen

gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten
```

```
25
26 for auteur, teksten in auteurs_groep.items():
27     gecorrigeerde_teksten[auteur] = []
28     for tekst in teksten:
29         woorden = tekst.split()
30         gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [
       woord] for woord in woorden]
31         gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in
       gecorrigeerde_woorden]))
32
33 #printen om te checken
34 #for auteur, teksten in gecorrigeerde_teksten.items():
35 #    print(f"Auteur: {auteur}")
36 #    for tekst in teksten:
37 #        print(f"  Tekst: {tekst}")
38
39 frequentwords = pd.read_excel("C:\\Users\\Isabelle\\Documents\\BEP\\Maskeren\\wordFrequency.
       xlsx", sheet_name=1)
40 #lijst gebaseerd op COCA wordt ingeladen
41
42 #print(frequentwords.columns)
43
44 lemma_kolom = frequentwords['lemma']
45
46 #k = 1000 #deze parameter is met de hand aan te passen
47
48
49 k_values = range(0, 5001, 100)
50 f1_scores = []
51 precisie_scores = []
52 recall_scores = []
53
54 for k in k_values:
55     if k == 0:
56         f1_scores.append(0)  # Als k=0, sla over (geen features mogelijk)
57         precisie_scores.append(0)
58         recall_scores.append(0)
59         continue
60
61     W_k = lemma_kolom.head(k).tolist()
62
63     #print(W_k)
64
65     def vervang_woorden_tekst(string, W_k):
66         #woorden_string = ' '.join(string)
67         woorden = string.split()
68         nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in
       woorden]
69         return ' '.join(nieuwe_woorden)
70
71     final_teksten = {}
72
73     # TEST VOOR NIEUWE MANIER
74     for auteur, teksten in gecorrigeerde_teksten.items():
75         final_teksten[auteur] = []
76         for tekst in teksten:
77             final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))
78
79     #printen om te checken
80     #for auteur, teksten in final_teksten.items():
81     #    print(f"Auteur: {auteur}")
82     #    for tekst in teksten:
83     #        print(f"  Tekst: {tekst}")
84
85     rows = [{'auteur': auteur, 'tekst': tekst} for auteur, teksten in final_teksten.items()
       for tekst in teksten]
86     df = pd.DataFrame(rows)
87
88     x = df['tekst']
89     y = df['auteur']
90
```

```
    vectorizer = CountVectorizer(stop_words=None)
    X_tfidf = vectorizer.fit_transform(x)


    # Split dataset into training set and test set
    tekst_train, tekst_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    model = LogisticRegression()
    model.fit(tekst_train, author_train)

    author_pred = model.predict(tekst_test)

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    f1_scores.append(f1score)
    precisie_scores.append(precisiescore)
    recall_scores.append(recallscore)


reference_f1 = 0.9144047829092788
reference_precision = 0.9318696582749882
reference_recall = 0.9010333741375562


plt.figure(figsize=(10, 6))
plt.plot(k_values, f1_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_f1, color='m', linestyle='--', label=f'Ref. F1 = {reference_f1:.3f}')
plt.title("Victoriaanse dataset COCA-maskering F1-score uitgezet tegen k (logistische
    regressie)")
plt.xlabel("k")
plt.ylabel("F1-score")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, precisie_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_precision, color='m', linestyle='--', label=f'Ref. precisie = {
    reference_precision:.3f}')
plt.title("Victoriaanse dataset COCA-maskering macroprecisie uitgezet tegen k (logistische
    regressie)")
plt.xlabel("k")
plt.ylabel("Macroprecisie")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, recall_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_recall, color='m', linestyle='--', label=f'Ref. recall = {
    reference_recall:.3f}')
plt.title("Victoriaanse dataset COCA-maskering macrorecall uitgezet tegen k (logistische
    regressie)")
plt.xlabel("k")
plt.ylabel("Macrorecall")
plt.legend()
plt.grid()
plt.show()
```

In de laatste code voor de Victoriaanse dataset is eigen maskering in combinatie met logistische regressie toegepast en zijn hier grafieken van gemaakt:

```
import pandas as pd
import re

data_victorian = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\18
    th_19th_Century_Victorian_Authors\\dataset\\dataset\\
    Gungor_2018_VictorianAuthorAttribution_data-train.csv", delimiter=",", error_bad_lines=
    False, header=0, encoding="latin-1")
```

```
# data wordt ingelezen, gescheiden op komma, met juiste encoding
data_victorian = data_victorian.sample(frac=0.25, random_state=1) #deel van data gebruiken om te kijken of het werkt? HIJ DOET HET!!! maar wel traag

from spellchecker import SpellChecker
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

spel = SpellChecker()

#print(data_victorian)
#print(data_victorian.columns)

auteurs_groep = data_victorian.groupby('author')['text'].apply(list).to_dict() #dictionary met auteursgroepen

gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

for auteur, teksten in auteurs_groep.items():
    gecorrigeerde_teksten[auteur] = []
    for tekst in teksten:
        woorden = tekst.split()
        gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [woord] for woord in woorden]
        gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in gecorrigeerde_woorden]))

from collections import Counter

#k = 200
k_values = range(0, 5001, 100)
f1_scores = []
precisie_scores = []
recall_scores = []

for k in k_values:
    if k == 0:
        f1_scores.append(0)  # Als k=0, sla over (geen features mogelijk)
        precisie_scores.append(0)
        recall_scores.append(0)
        continue

    word_counter = Counter()

    for tweets in gecorrigeerde_teksten.values():
        for tweet in tweets:
            # Split de tweet in woorden en tel ze
            geenleestekens_words = re.sub(r'[^\w\s-]', '', tweet.lower())
            words = geenleestekens_words.split()
            words = [word.lstrip('-').rstrip('-') for word in words]
            word_counter.update(words)

    frequentwords = word_counter.most_common(k)

    # Print de gesorteerde lijst van woorden en hun frequentie
    #for word, count in frequentwords[:100]:
    #    print(f"{word} {count}")

    W_k = [word for word, _ in frequentwords]
    #print(W_k)

    def vervang_woorden_tekst(string, W_k):
    #woorden_string = ' '.join(string)
        woorden = string.split()
        nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in woorden]
```

```
        return ' '.join(nieuwe_woorden)

    final_teksten = {}

    # TEST VOOR NIEUWE MANIER
    for auteur, teksten in gecorrigeerde_teksten.items():
        final_teksten[auteur] = []
        for tekst in teksten:
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    #printen om te checken
    #for auteur, teksten in final_teksten.items():
    #    print(f"Auteur: {auteur}")
    #    for tekst in teksten:
    #        print(f" Tekst: {tekst}")

    rows = [{'auteur': auteur, 'tekst': tekst} for auteur, teksten in final_teksten.items()
    for tekst in teksten]
    df = pd.DataFrame(rows)

    x = df['tekst']
    y = df['auteur']

    vectorizer = CountVectorizer(stop_words=None)
    X_tfidf = vectorizer.fit_transform(x)


    # Split dataset into training set and test set
    tekst_train, tekst_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    model = LogisticRegression()
    model.fit(tekst_train, author_train)

    author_pred = model.predict(tekst_test)

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    f1_scores.append(f1score)
    precisie_scores.append(precisiescore)
    recall_scores.append(recallscore)


reference_f1 = 0.9144047829092788
reference_precision = 0.9318696582749882
reference_recall = 0.9010333741375562

plt.figure(figsize=(10, 6))
plt.plot(k_values, f1_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_f1, color='m', linestyle='--', label=f'Ref. F1 = {reference_f1:.3f}')
plt.title("Victoriaanse dataset eigen maskering F1-score uitgezet tegen k (logistische
    regressie)")
plt.xlabel("k")
plt.ylabel("F1-score")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, precisie_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_precision, color='m', linestyle='--', label=f'Ref. precisie = {
    reference_precision:.3f}')
plt.title("Victoriaanse dataset eigen maskering macroprecisie uitgezet tegen k (logistische
    regressie)")
plt.xlabel("k")
plt.ylabel("Macroprecisie")
plt.legend()
plt.grid()
plt.show()

```

```
plt.figure(figsize=(10, 6))
plt.plot(k_values, recall_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_recall, color='m', linestyle='--', label=f'Ref. recall = {reference_recall:.3f}')
plt.title("Victoriaanse dataset eigen maskering macrorecall uitgezet tegen k (logistische regressie)")
plt.xlabel("k")
plt.ylabel("Macrorecall")
plt.legend()
plt.grid()
plt.show()
```

De onderstaande code is gebruikt om de gecombineerde grafieken voor de Victoriaanse dataset te maken:

```
from tabulate import tabulate
from sklearn import metrics
from spellchecker import SpellChecker
from sklearn import svm
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
import re
import emoji
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import svm
from sklearn import metrics
from collections import Counter
from tfIdfInheritVectorizer.feature_extraction.vectorizer import TfidfVectorizer


def creeer_gecorrigeerde_teksten(data_victorian):
    spel = SpellChecker()

    auteurs_groep = data_victorian.groupby('author')['text'].apply(list).to_dict() # dictionary met auteursgroepen

    gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

    for auteur, teksten in auteurs_groep.items():
        gecorrigeerde_teksten[auteur] = []
        for tekst in teksten:
            woorden = tekst.split()
            gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [woord] for woord in woorden]
            gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in gecorrigeerde_woorden]))
    return gecorrigeerde_teksten

def vervang_woorden_tekst(string, W_k):
    #woorden_string = ' '.join(string)
    woorden = string.split()
    nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in woorden]
    return ' '.join(nieuwe_woorden)


def svm_coca(k, gecorrigeerde_teksten, lemma_kolom):
    if k == 0:
        return 0,0,0

    W_k = lemma_kolom.head(k).tolist()

    final_teksten = {}

    for auteur, teksten in gecorrigeerde_teksten.items():
```

```
53          final_teksten[auteur] = []
54          for tekst in teksten:
55              final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))
56
57      rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
        for tweet in tweets]
58      df = pd.DataFrame(rows)
59
60      x = df['tweet']
61      y = df['auteur']
62
63      vectorizer = TfidfVectorizer()
64      X_tfidf = vectorizer.fit_transform(x)
65
66      # Split dataset into training set and test set
67      tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
        test_size=0.3,random_state=109) # 70% training and 30% test
68
69      #Create a svm Classifier
70      clf = svm.SVC(kernel='linear') # Linear Kernel
71
72      #Train the model using the training sets
73      clf.fit(tweet_train, author_train)
74
75      #Predict the response for test dataset
76      author_pred = clf.predict(tweet_test)
77
78      f1score = metrics.f1_score(author_test, author_pred, average='macro')
79      precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
80      recallscore = metrics.recall_score(author_test, author_pred, average='macro')
81      return f1score, precisiescore, recallscore
82
83  def logreg_coca(k, gecorrigeerde_teksten, lemma_kolom):
84      if k == 0:
85          return 0,0,0
86
87      W_k = lemma_kolom.head(k).tolist()
88
89      final_teksten = {}
90
91      # TEST VOOR NIEUWE MANIER
92      for auteur, teksten in gecorrigeerde_teksten.items():
93          final_teksten[auteur] = []
94          for tekst in teksten:
95              final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))
96
97      rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
        for tweet in tweets]
98      df = pd.DataFrame(rows)
99
100     x = df['tweet']
101     y = df['auteur']
102
103     vectorizer = CountVectorizer(stop_words=None)
104     X_tfidf = vectorizer.fit_transform(x)
105
106     # Split dataset into training set and test set
107     tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
        test_size=0.3,random_state=109) # 70% training and 30% test
108
109     model = LogisticRegression()
110     model.fit(tweet_train, author_train)
111
112     author_pred = model.predict(tweet_test)
113
114     f1score = metrics.f1_score(author_test, author_pred, average='macro')
115     precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
116     recallscore = metrics.recall_score(author_test, author_pred, average='macro')
117     return f1score,precisiescore,recallscore
118
119 def svm_eigen(k, gecorrigeerde_teksten):
```

```
    if k == 0:
        return 0,0,0

    word_counter = Counter()

    for tweets in gecorrigeerde_teksten.values():
        for tweet in tweets:
            # Split de tweet in woorden en tel ze
            geenleestekens_words = re.sub(r'[^\w\s-]', '', tweet.lower())
            words = geenleestekens_words.split()
            words = [word.lstrip('-').rstrip('-') for word in words]
            word_counter.update(words)

    frequentwords = word_counter.most_common(k)
    W_k = [word for word, _ in frequentwords]

    final_teksten = {}

    for auteur, teksten in gecorrigeerde_teksten.items():
        final_teksten[auteur] = []
        for tekst in teksten:
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
    for tweet in tweets]
    df = pd.DataFrame(rows)

    x = df['tweet']
    y = df['auteur']

    vectorizer = TfidfVectorizer()
    X_tfidf = vectorizer.fit_transform(x)

    # Split dataset into training set and test set
    tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    #Create a svm Classifier
    clf = svm.SVC(kernel='linear') # Linear Kernel

    #Train the model using the training sets
    clf.fit(tweet_train, author_train)

    #Predict the response for test dataset
    author_pred = clf.predict(tweet_test)

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    return f1score,precisiescore,recallscore

def logreg_eigen(k, gecorrigeerde_teksten):
    if k == 0:
        return 0,0,0

    word_counter = Counter()

    for tweets in gecorrigeerde_teksten.values():
        for tweet in tweets:
            # Split de tweet in woorden en tel ze
            geenleestekens_words = re.sub(r'[^\w\s-]', '', tweet.lower())
            words = geenleestekens_words.split()
            words = [word.lstrip('-').rstrip('-') for word in words]
            word_counter.update(words)

    frequentwords = word_counter.most_common(k)

    W_k = [word for word, _ in frequentwords]

    final_teksten = {}

```

```
    # TEST VOOR NIEUWE MANIER
    for auteur, teksten in gecorrigeerde_teksten.items():
        final_teksten[auteur] = []
        for tekst in teksten:
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
    for tweet in tweets]
    df = pd.DataFrame(rows)

    x = df['tweet']
    y = df['auteur']

    vectorizer = CountVectorizer(stop_words=None)
    X_tfidf = vectorizer.fit_transform(x)

    # Split dataset into training set and test set
    tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    model = LogisticRegression()
    model.fit(tweet_train, author_train)

    author_pred = model.predict(tweet_test)

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    return f1score,precisiescore,recallscore

# Gebruiken we bij COCA Maskering
def svm_coca_resultaten(k_values,gecorrigeerde_teksten, frequentwords):

    lemma_kolom = frequentwords['lemma']

    f1_scores = []
    precisie_scores = []
    recall_scores = []

    for k in k_values:
        f1_score,precisie_score,recall_score = svm_coca(k, gecorrigeerde_teksten, lemma_kolom
    )
        f1_scores.append(f1_score)
        precisie_scores.append(precisie_score)
        recall_scores.append(recall_score)
    return f1_scores, precisie_scores, recall_scores


def logreg_coca_resultaten(k_values,gecorrigeerde_teksten, frequentwords):
    lemma_kolom = frequentwords['lemma']


    f1_scores = []
    precisie_scores = []
    recall_scores = []

    for k in k_values:
        f1_score,precisie_score,recall_score = logreg_coca(k, gecorrigeerde_teksten,
    lemma_kolom)
        f1_scores.append(f1_score)
        precisie_scores.append(precisie_score)
        recall_scores.append(recall_score)
    return f1_scores, precisie_scores, recall_scores

def svm_eigen_resultaten(k_values,gecorrigeerde_teksten):
    f1_scores = []
    precisie_scores = []
    recall_scores = []

    for k in k_values:
        f1_score,precisie_score,recall_score = svm_eigen(k, gecorrigeerde_teksten)
```

```
        f1_scores.append(f1_score)
        precisie_scores.append(precisie_score)
        recall_scores.append(recall_score)
    return f1_scores, precisie_scores, recall_scores

def logreg_eigen_resultaten(k_values,gecorrigeerde_teksten):
    f1_scores = []
    precisie_scores = []
    recall_scores = []

    for k in k_values:
        f1_score,precisie_score,recall_score = logreg_eigen(k, gecorrigeerde_teksten)
        f1_scores.append(f1_score)
        precisie_scores.append(precisie_score)
        recall_scores.append(recall_score)
    return f1_scores, precisie_scores, recall_scores

frequentwords = pd.read_excel("C:\\Users\\Isabelle\\Documents\\BEP\\Maskeren\\wordFrequency.
    xlsx", sheet_name=1)
data_victorian = pd.read_csv("C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\18
    th_19th_Century_Victorian_Authors\\dataset\\dataset\\
    Gungor_2018_VictorianAuthorAttribution_data-train.csv", delimiter=",", error_bad_lines=
    False, header=0, encoding="latin-1")
data_victorian = data_victorian.sample(frac=0.25, random_state=1) #deel van data gebruiken om
     te kijken of het werkt? HIJ DOET HET!!! maar wel traag
k_values = range(0, 5001, 100)

gecorrigeerde_teksten = creeer_gecorrigeerde_teksten(data_victorian)
svm_coca_f1, svm_coca_precisie, svm_coca_recall = svm_coca_resultaten(k_values,
    gecorrigeerde_teksten, frequentwords)
logreg_coca_f1, logreg_coca_precisie, logreg_coca_recall = logreg_coca_resultaten(k_values,
    gecorrigeerde_teksten, frequentwords)
svm_eigen_f1, svm_eigen_precisie, svm_eigen_recall = svm_eigen_resultaten(k_values,
    gecorrigeerde_teksten)
logreg_eigen_f1, logreg_eigen_precisie, logreg_eigen_recall = logreg_eigen_resultaten(
    k_values,gecorrigeerde_teksten)

reference_f1_svm = 0.8160282931172088
reference_precision_svm = 0.932525676251512
reference_recall_svm = 0.7652173468005774


reference_f1_logreg = 0.9144047829092788
reference_precision_logreg = 0.9318696582749882
reference_recall_logreg = 0.9010333741375562

plt.figure(figsize=(10, 6))
plt.plot(k_values, svm_coca_f1, marker='o', linestyle='-', color='b')
plt.plot(k_values, logreg_coca_f1, marker='o', linestyle='-', color='g')
plt.plot(k_values, svm_eigen_f1, marker='o', linestyle='-', color='r')
plt.plot(k_values, logreg_eigen_f1, marker='o', linestyle='-', color='purple')

plt.axhline(y=reference_f1_svm, color='black', linestyle='--', label=f'Ref. F1 SVM = {
    reference_f1_svm:.3f}')
plt.axhline(y=reference_f1_logreg, color='brown', linestyle='--', label=f'Ref. F1 LOGREG = {
    reference_f1_logreg:.3f}')
plt.title("Victoriaanse dataset F1-score uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("F1-score")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, svm_coca_precisie, marker='o', linestyle='-', color='b')
plt.plot(k_values, logreg_coca_precisie, marker='o', linestyle='-', color='g')
plt.plot(k_values, svm_eigen_precisie, marker='o', linestyle='-', color='r')
plt.plot(k_values, logreg_eigen_precisie, marker='o', linestyle='-', color='purple')

plt.axhline(y=reference_precision_svm, color='black', linestyle='--', label=f'Ref. precisie
    SVM = {reference_precision_svm:.3f}')
```

```
plt.axhline(y=reference_precision_logreg, color='brown', linestyle='--', label=f'Ref. precisie LOGREG = {reference_precision_logreg:.3f}')
plt.title("Victoriaanse dataset macroprecisie uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("Macroprecisie")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, svm_coca_recall, marker='o', linestyle='-', color='b')
plt.plot(k_values, logreg_coca_recall, marker='o', linestyle='-', color='g')
plt.plot(k_values, svm_eigen_recall, marker='o', linestyle='-', color='r')
plt.plot(k_values, logreg_eigen_recall, marker='o', linestyle='-', color='purple')

plt.axhline(y=reference_recall_svm, color='black', linestyle='--', label=f'Ref. recall SVM = {reference_recall_svm:.3f}')
plt.axhline(y=reference_recall_logreg, color='brown', linestyle='--', label=f'Ref. recall LOGREG = {reference_recall_logreg:.3f}')
plt.title("Victoriaanse dataset macrorecall uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("Macrorecall")
plt.legend()
plt.grid()
plt.show()
```

C

Pythoncodes Federalist Papers dataset

De eerste, onderstaande, code is gebruikt om de Federalist Papers dataset schoon te maken en SVM toe te passen zonder maskering:

```
import os
import pandas as pd

folder_path = 'C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Federalist papers\\
    FedPapersCorpus\\FedPapersCorpustrain'

texts = []

# Loop door elk bestand in de map
for filename in os.listdir(folder_path):
    if filename.endswith('.txt'):
        # Open en lees het bestand
        with open(os.path.join(folder_path, filename), 'r', encoding='utf-8') as file:
            content = file.read()
            # Zoek naar de positie van "author:" en behoud alleen de tekst vanaf dat punt
            author_index = content.find("Author:")
            if author_index != -1:
                content_from_author = content[author_index:]  # Behoud alles vanaf "author:"
                texts.append(content_from_author)  # Voeg toe aan de lijst
            else:
                texts.append("")  # Voeg een lege string toe als "author:" niet gevonden is



# print om te checken
#for i, text in enumerate(texts, 1):
#    print(f"Bestand {i}: {text[:100]}...")

# Dictionary om auteurs en hun teksten op te slaan
auteurs_dict = {}


for text in texts:
    author = ""
    lines_without_author = []
    regels = text.split("\n")
    for regel in regels:
        #print("Regel:", regel)
        if author != "":
            lines_without_author.append(regel)
        if regel.strip().lower().startswith("author:"):
            author = regel.split(":")[1].strip()
            #print("author:",author)
    text = '\n'.join(lines_without_author)
    #print(text)
    if " and " in author:
```

```
        for split_auteur in author.split(" and "):
            if auteurs_dict.get(split_auteur) == None:
                auteurs_dict[split_auteur] = []
                auteurs_dict[split_auteur].append(text)
            else:
                auteurs_dict[split_auteur].append(text)
    else:
        if auteurs_dict.get(author) == None:
            auteurs_dict[author] = []
            auteurs_dict[author].append(text)
        else:
            auteurs_dict[author].append(text)

#printen om te checken
#for auteur, teksten in auteurs_dict.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

from spellchecker import SpellChecker

spel = SpellChecker()

gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

for auteur, teksten in auteurs_dict.items():
    gecorrigeerde_teksten[auteur] = []
    for tekst in teksten:
        regels = tekst.split("\n")
        for regel in regels:
            woorden = regel.split()
            gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [woord] for woord in woorden]
            gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in gecorrigeerde_woorden]))

#printen om te checken
#for auteur, teksten in gecorrigeerde_teksten.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

rows = [{'auteur': auteur, 'tekst': tekst} for auteur, teksten in gecorrigeerde_teksten.items() for tekst in teksten]
df = pd.DataFrame(rows)

# Import train_test_split function
from sklearn.model_selection import train_test_split

x = df['tekst']
y = df['auteur']

from tfIdfInheritVectorizer.feature_extraction.vectorizer import TfidfVectorizer

vectorizer = TfidfVectorizer()
X_tfidf = vectorizer.fit_transform(x)


# Split dataset into training set and test set
tekst_train, tekst_test, author_train, author_test = train_test_split(X_tfidf,y, test_size=0.3,random_state=109) # 70% training and 30% test

#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(tekst_train, author_train)

```

```
#Predict the response for test dataset
author_pred = clf.predict(tekst_test)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy: how often is the classifier correct?
print("Macroprecisie:",metrics.precision_score(author_test, author_pred, average='macro')) #
    bij average kan je micro, macro of weighted instellen
print("Gevoeligheid:",metrics.recall_score(author_test, author_pred, average='macro')) #ook
    hier bij average micro, macro of weighted kiezen
print("F1-score:",metrics.f1_score(author_test, author_pred, average='macro'))
```

De tweede code is gebruikt om de Federalist Papers dataset schoon te maken en logistische regressie toe te passen:

```
import os
import pandas as pd

folder_path = 'C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Federalist papers\\
    FedPapersCorpus\\FedPapersCorpustrain'

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

texts = []

# Loop door elk bestand in de map
for filename in os.listdir(folder_path):
    if filename.endswith('.txt'):
        # Open en lees het bestand
        with open(os.path.join(folder_path, filename), 'r', encoding='utf-8') as file:
            content = file.read()
            # Zoek naar de positie van "author:" en behoud alleen de tekst vanaf dat punt
            author_index = content.find("Author:")
            if author_index != -1:
                content_from_author = content[author_index:]  # Behoud alles vanaf "author:"
                texts.append(content_from_author)  # Voeg toe aan de lijst
            else:
                texts.append("")  # Voeg een lege string toe als "author:" niet gevonden is



# print om te checken
#for i, text in enumerate(texts, 1):
#    print(f"Bestand {i}: {text[:100]}...")

# Dictionary om auteurs en hun teksten op te slaan
auteurs_dict = {}


for text in texts:
    author = ""
    lines_without_author = []
    regels = text.split("\n")
    for regel in regels:
        #print("Regel:", regel)
        if author != "":
            lines_without_author.append(regel)
        if regel.strip().lower().startswith("author:"):
            author = regel.split(":")[1].strip()
            #print("author:",author)
    text = '\n'.join(lines_without_author)
    #print(text)
    if " and " in author:
        for split_auteur in author.split(" and "):
            if auteurs_dict.get(split_auteur) == None:
                auteurs_dict[split_auteur] = []
                auteurs_dict[split_auteur].append(text)
```

```
            else:
                auteurs_dict[split_auteur].append(text)
    else:
        if auteurs_dict.get(author) == None:
            auteurs_dict[author] = []
            auteurs_dict[author].append(text)
        else:
            auteurs_dict[author].append(text)

#printen om te checken
#for auteur, teksten in auteurs_dict.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

from spellchecker import SpellChecker

spel = SpellChecker()

gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

for auteur, teksten in auteurs_dict.items():
    gecorrigeerde_teksten[auteur] = []
    for tekst in teksten:
        regels = tekst.split("\n")
        for regel in regels:
            woorden = regel.split()
            gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [woord] for woord in woorden]
            gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in gecorrigeerde_woorden]))

#printen om te checken
#for auteur, teksten in gecorrigeerde_teksten.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

rows = [{'auteur': auteur, 'tekst': tekst} for auteur, teksten in gecorrigeerde_teksten.items() for tekst in teksten]
df = pd.DataFrame(rows)

x = df['tekst']
y = df['auteur']


vectorizer = CountVectorizer(stop_words=None)
X_tfidf = vectorizer.fit_transform(x)


# Split dataset into training set and test set
tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y, test_size=0.3,random_state=109) # 70% training and 30% test

model = LogisticRegression()
model.fit(tweet_train, author_train)

author_pred = model.predict(tweet_test)
print(classification_report(author_test, author_pred))


#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy: how often is the classifier correct?
#print("Accuraatheid:",metrics.accuracy_score(author_test, author_pred))
print("Macroprecisie:",metrics.precision_score(author_test, author_pred, average='macro')) # bij average kan je micro, macro of weighted instellen
print("Gevoeligheid:",metrics.recall_score(author_test, author_pred, average='macro')) #ook hier bij average micro, macro of weighted kiezen
print("F1-score:",metrics.f1_score(author_test, author_pred, average='macro'))
```

In de volgende code is COCA-maskering in combinatie met SVM toegepast op de Federalist Papers dataset en zijn hier grafieken van gemaakt:

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

folder_path = 'C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Federalist papers\\
    FedPapersCorpus\\FedPapersCorpustrain'

texts = []

# Loop door elk bestand in de map
for filename in os.listdir(folder_path):
    if filename.endswith('.txt'):
        # Open en lees het bestand
        with open(os.path.join(folder_path, filename), 'r', encoding='utf-8') as file:
            content = file.read()
            # Zoek naar de positie van "author:" en behoud alleen de tekst vanaf dat punt
            author_index = content.find("Author:")
            if author_index != -1:
                content_from_author = content[author_index:]  # Behoud alles vanaf "author:"
                texts.append(content_from_author)  # Voeg toe aan de lijst
            else:
                texts.append("")  # Voeg een lege string toe als "author:" niet gevonden is



# print om te checken
#for i, text in enumerate(texts, 1):
#    print(f"Bestand {i}: {text[:100]}...")

# Dictionary om auteurs en hun teksten op te slaan
auteurs_dict = {}


for text in texts:
    author = ""
    lines_without_author = []
    regels = text.split("\n")
    for regel in regels:
        #print("Regel:", regel)
        if author != "":
            lines_without_author.append(regel)
        if regel.strip().lower().startswith("author:"):
            author = regel.split(":")[1].strip()
            #print("author:",author)
    text = '\n'.join(lines_without_author)
    #print(text)
    if " and " in author:
        for split_auteur in author.split(" and "):
            if auteurs_dict.get(split_auteur) == None:
                auteurs_dict[split_auteur] = []
                auteurs_dict[split_auteur].append(text)
            else:
                auteurs_dict[split_auteur].append(text)
    else:
        if auteurs_dict.get(author) == None:
            auteurs_dict[author] = []
            auteurs_dict[author].append(text)
        else:
            auteurs_dict[author].append(text)

#printen om te checken
#for auteur, teksten in auteurs_dict.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

from spellchecker import SpellChecker
```

```
68
69 spel = SpellChecker()
70
71 gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten
72
73 for auteur, teksten in auteurs_dict.items():
74     gecorrigeerde_teksten[auteur] = []
75     for tekst in teksten:
76         regels = tekst.split("\n")
77         for regel in regels:
78             woorden = regel.split()
79             gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else
       [woord] for woord in woorden]
80             gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in
     gecorrigeerde_woorden]))
81
82 frequentwords = pd.read_excel("C:\\Users\\Isabelle\\Documents\\BEP\\Maskeren\\wordFrequency.
     xlsx", sheet_name=1)
83 #lijst gebaseerd op COCA wordt ingeladen
84
85 #print(frequentwords.columns)
86
87 lemma_kolom = frequentwords['lemma']
88
89 #k = 100 #deze parameter is met de hand aan te passen
90
91 k_values = range(0, 5001, 100)
92 f1_scores = []
93 precisie_scores = []
94 recall_scores = []
95
96 for k in k_values:
97     if k == 0:
98         f1_scores.append(0)  # Als k=0, sla over (geen features mogelijk)
99         precisie_scores.append(0)
100        recall_scores.append(0)
101        continue
102
103    W_k = lemma_kolom.head(k).tolist()
104
105    #print(W_k)
106
107    def vervang_woorden_tekst(string, W_k):
108        #woorden_string = ' '.join(string)
109        woorden = string.split()
110        nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in
     woorden]
111        return ' '.join(nieuwe_woorden)
112
113    final_teksten = {}
114
115    # TEST VOOR NIEUWE MANIER
116    for auteur, teksten in gecorrigeerde_teksten.items():
117        final_teksten[auteur] = []
118        for tekst in teksten:
119            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))
120
121    #printen om te checken
122    #for auteur, teksten in final_teksten.items():
123    #    print(f"Auteur: {auteur}")
124    #    for tekst in teksten:
125    #        print(f"  Tekst: {tekst}")
126
127    rows = [{'auteur': auteur, 'tekst': tekst} for auteur, teksten in final_teksten.items()
     for tekst in teksten]
128    df = pd.DataFrame(rows)
129
130    # Import train_test_split function
131    from sklearn.model_selection import train_test_split
132
133    x = df['tekst']
```

```
    y = df['auteur']

    from tfIdfInheritVectorizer.feature_extraction.vectorizer import TfidfVectorizer

    vectorizer = TfidfVectorizer()
    X_tfidf = vectorizer.fit_transform(x)


    # Split dataset into training set and test set
    tekst_train, tekst_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    #Import svm model
    from sklearn import svm

    #Create a svm Classifier
    clf = svm.SVC(kernel='linear') # Linear Kernel

    #Train the model using the training sets
    clf.fit(tekst_train, author_train)

    #Predict the response for test dataset
    author_pred = clf.predict(tekst_test)

    #Import scikit-learn metrics module for accuracy calculation
    from sklearn import metrics

    # Model Accuracy: how often is the classifier correct?
    #print("Accuraatheid:",metrics.accuracy_score(author_test, author_pred))
    #print("Precisie:",metrics.precision_score(author_test, author_pred, average='macro')) #
    bij average kan je micro, macro of weighted instellen
    #print("Gevoeligheid:",metrics.recall_score(author_test, author_pred, average='macro')) #
    ook hier bij average micro, macro of weighted kiezen

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    f1_scores.append(f1score)
    precisie_scores.append(precisiescore)
    recall_scores.append(recallscore)

reference_f1 = 0.38804359376179787
reference_precision = 0.6795722449369963
reference_recall = 0.39179935160285817

plt.figure(figsize=(10, 6))
plt.plot(k_values, f1_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_f1, color='r', linestyle='--', label=f'Ref. F1 = {reference_f1:.3f}')
plt.title("Federalist Papers dataset COCA-maskering F1-score uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("F1-score")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, precisie_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_precision, color='r', linestyle='--', label=f'Ref. precisie = {
    reference_precision:.3f}')
plt.title("Federalist Papers dataset COCA-maskering macroprecisie uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("Macroprecisie")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, recall_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_recall, color='r', linestyle='--', label=f'Ref. recall = {
    reference_recall:.3f}')
plt.title("Federalist Papers dataset COCA-maskering macrorecall uitgezet tegen k")
```

```
plt.xlabel(”k”)
plt.ylabel(”Macrorecall”)
plt.legend()
plt.grid()
plt.show()
```

In de volgende code is eigen maskering in combinatie met SVM toegepast op de Federalist Papers dataset en zijn hier grafieken van gemaakt:

```
import os
import pandas as pd
import re
import numpy as np
import matplotlib.pyplot as plt

folder_path = 'C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Federalist papers\\
    FedPapersCorpus\\FedPapersCorpustrain'

texts = []

# Loop door elk bestand in de map
for filename in os.listdir(folder_path):
    if filename.endswith('.txt'):
        # Open en lees het bestand
        with open(os.path.join(folder_path, filename), 'r', encoding='utf-8') as file:
            content = file.read()
            # Zoek naar de positie van ”author:” en behoud alleen de tekst vanaf dat punt
            author_index = content.find(”Author:”)
            if author_index != -1:
                content_from_author = content[author_index:]  # Behoud alles vanaf ”author:”
                texts.append(content_from_author)  # Voeg toe aan de lijst
            else:
                texts.append(””)  # Voeg een lege string toe als ”author:” niet gevonden is



# print om te checken
#for i, text in enumerate(texts, 1):
#    print(f”Bestand {i}: {text[:100]}...”)

# Dictionary om auteurs en hun teksten op te slaan
auteurs_dict = {}


for text in texts:
    author = ””
    lines_without_author = []
    regels = text.split(”\n”)
    for regel in regels:
        #print(”Regel:”, regel)
        if author != ””:
            lines_without_author.append(regel)
        if regel.strip().lower().startswith(”author:”):
            author = regel.split(”:”)[1].strip()
            #print(”author:”,author)
    text = '\n'.join(lines_without_author)
    #print(text)
    if ” and ” in author:
        for split_auteur in author.split(” and ”):
            if auteurs_dict.get(split_auteur) == None:
                auteurs_dict[split_auteur] = []
                auteurs_dict[split_auteur].append(text)
            else:
                auteurs_dict[split_auteur].append(text)
    else:
        if auteurs_dict.get(author) == None:
            auteurs_dict[author] = []
            auteurs_dict[author].append(text)
        else:
            auteurs_dict[author].append(text)

```

```
#printen om te checken
#for auteur, teksten in auteurs_dict.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

from spellchecker import SpellChecker

spel = SpellChecker()

gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

for auteur, teksten in auteurs_dict.items():
    gecorrigeerde_teksten[auteur] = []
    for tekst in teksten:
        regels = tekst.split("\n")
        for regel in regels:
            woorden = regel.split()
            gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else [woord] for woord in woorden]
            gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in gecorrigeerde_woorden]))

from collections import Counter

#k = 200

k_values = range(0, 5001, 100)
f1_scores = []
precisie_scores = []
recall_scores = []

for k in k_values:
    if k == 0:
        f1_scores.append(0)  # Als k=0, sla over (geen features mogelijk)
        precisie_scores.append(0)
        recall_scores.append(0)
        continue

    word_counter = Counter()

    for teksten in gecorrigeerde_teksten.values():
        for tekst in teksten:
            # Split de tweet in woorden en tel ze
            geenleestekens_words = re.sub(r'[^\w\s-]', '', tekst.lower())
            words = geenleestekens_words.split()
            words = [word.lstrip('-').rstrip('-') for word in words]
            word_counter.update(words)

    frequentwords = word_counter.most_common(k)

    # Print de gesorteerde lijst van woorden en hun frequentie
    #for word, count in frequentwords[:100]:
    #    print(f"{word} {count}")

    W_k = [word for word, _ in frequentwords]
    #print(W_k)

    def vervang_woorden_tekst(string, W_k):
        #woorden_string = ' '.join(string)
        woorden = string.split()
        nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in woorden]
        return ' '.join(nieuwe_woorden)

    final_teksten = {}

    # TEST VOOR NIEUWE MANIER
    for auteur, teksten in gecorrigeerde_teksten.items():
        final_teksten[auteur] = []
        for tekst in teksten:
```

```
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    #printen om te checken
    #for auteur, teksten in final_teksten.items():
    #    print(f"Auteur: {auteur}")
    #    for tekst in teksten:
    #        print(f"  Tekst: {tekst}")

    rows = [{'auteur': auteur, 'tekst': tekst} for auteur, teksten in final_teksten.items()
    for tekst in teksten]
    df = pd.DataFrame(rows)

    # Import train_test_split function
    from sklearn.model_selection import train_test_split

    x = df['tekst']
    y = df['auteur']

    from tfIdfInheritVectorizer.feature_extraction.vectorizer import TfidfVectorizer

    vectorizer = TfidfVectorizer()
    X_tfidf = vectorizer.fit_transform(x)


    # Split dataset into training set and test set
    tekst_train, tekst_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    #Import svm model
    from sklearn import svm

    #Create a svm Classifier
    clf = svm.SVC(kernel='linear') # Linear Kernel

    #Train the model using the training sets
    clf.fit(tekst_train, author_train)

    #Predict the response for test dataset
    author_pred = clf.predict(tekst_test)

    #Import scikit-learn metrics module for accuracy calculation
    from sklearn import metrics

    # Model Accuracy: how often is the classifier correct?
    #print("Accuraatheid:",metrics.accuracy_score(author_test, author_pred))
    #print("Precisie:",metrics.precision_score(author_test, author_pred, average='macro')) #
    bij average kan je micro, macro of weighted instellen
    #print("Gevoeligheid:",metrics.recall_score(author_test, author_pred, average='macro')) #
    ook hier bij average micro, macro of weighted kiezen

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    f1_scores.append(f1score)
    precisie_scores.append(precisiescore)
    recall_scores.append(recallscore)

reference_f1 = 0.38804359376179787
reference_precision = 0.6795722449369963
reference_recall = 0.39179935160285817

plt.figure(figsize=(10, 6))
plt.plot(k_values, f1_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_f1, color='r', linestyle='--', label=f'Ref. F1 = {reference_f1:.3f}')
plt.title("Federalist Papers dataset eigen maskering F1-score uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("F1-score")
plt.legend()
plt.grid()
plt.show()

```

```
plt.figure(figsize=(10, 6))
plt.plot(k_values, precisie_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_precision, color='r', linestyle='--', label=f'Ref. precisie = {reference_precision:.3f}')
plt.title("Federalist Papers dataset eigen maskering macroprecisie uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("Macroprecisie")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, recall_scores, marker='o', linestyle='-', color='b')
plt.axhline(y=reference_recall, color='r', linestyle='--', label=f'Ref. recall = {reference_recall:.3f}')
plt.title("Federalist Papers dataset eigen maskering macrorecall uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("Macrorecall")
plt.legend()
plt.grid()
plt.show()
```

In de volgende code is COCA-maskering in combinatie met logistische regressie toegepast op de Federalist Papers dataset en zijn hier grafieken van gemaakt:

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

folder_path = 'C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Federalist papers\\FedPapersCorpus\\FedPapersCorpustrain'

texts = []

# Loop door elk bestand in de map
for filename in os.listdir(folder_path):
    if filename.endswith('.txt'):
        # Open en lees het bestand
        with open(os.path.join(folder_path, filename), 'r', encoding='utf-8') as file:
            content = file.read()
            # Zoek naar de positie van "author:" en behoud alleen de tekst vanaf dat punt
            author_index = content.find("Author:")
            if author_index != -1:
                content_from_author = content[author_index:]  # Behoud alles vanaf "author:"
                texts.append(content_from_author)  # Voeg toe aan de lijst
            else:
                texts.append("")  # Voeg een lege string toe als "author:" niet gevonden is



# print om te checken
#for i, text in enumerate(texts, 1):
#    print(f"Bestand {i}: {text[:100]}...")

# Dictionary om auteurs en hun teksten op te slaan
auteurs_dict = {}


for text in texts:
    author = ""
    lines_without_author = []
    regels = text.split("\n")
    for regel in regels:
        #print("Regel:", regel)
        if author != "":
```

```
            lines_without_author.append(regel)
        if regel.strip().lower().startswith("author:"):
            author = regel.split(":")[1].strip()
            #print("author:",author)
    text = '\n'.join(lines_without_author)
    #print(text)
    if " and " in author:
        for split_auteur in author.split(" and "):
            if auteurs_dict.get(split_auteur) == None:
                auteurs_dict[split_auteur] = []
                auteurs_dict[split_auteur].append(text)
            else:
                auteurs_dict[split_auteur].append(text)
    else:
        if auteurs_dict.get(author) == None:
            auteurs_dict[author] = []
            auteurs_dict[author].append(text)
        else:
            auteurs_dict[author].append(text)

#printen om te checken
#for auteur, teksten in auteurs_dict.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

from spellchecker import SpellChecker

spel = SpellChecker()

gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

for auteur, teksten in auteurs_dict.items():
    gecorrigeerde_teksten[auteur] = []
    for tekst in teksten:
        regels = tekst.split("\n")
        for regel in regels:
            woorden = regel.split()
            gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else
     [woord] for woord in woorden]
            gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in
    gecorrigeerde_woorden]))

frequentwords = pd.read_excel("C:\\Users\\Isabelle\\Documents\\BEP\\Maskeren\\wordFrequency.
    xlsx", sheet_name=1)
#lijst gebaseerd op COCA wordt ingeladen

#print(frequentwords.columns)

lemma_kolom = frequentwords['lemma']

#k = 100 #deze parameter is met de hand aan te passen

k_values = range(0, 5001, 100)
f1_scores = []
precisie_scores = []
recall_scores = []

for k in k_values:
    if k == 0:
        f1_scores.append(0)  # Als k=0, sla over (geen features mogelijk)
        precisie_scores.append(0)
        recall_scores.append(0)
        continue

    W_k = lemma_kolom.head(k).tolist()

    #print(W_k)

    def vervang_woorden_tekst(string, W_k):
        #woorden_string = ' '.join(string)
```

```
        woorden = string.split()
        nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in woorden]
        return ' '.join(nieuwe_woorden)

    final_teksten = {}

    # TEST VOOR NIEUWE MANIER
    for auteur, teksten in gecorrigeerde_teksten.items():
        final_teksten[auteur] = []
        for tekst in teksten:
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    #printen om te checken
    #for auteur, teksten in final_teksten.items():
    #    print(f"Auteur: {auteur}")
    #    for tekst in teksten:
    #        print(f"  Tekst: {tekst}")

    rows = [{'auteur': auteur, 'tekst': tekst} for auteur, teksten in final_teksten.items() for tekst in teksten]
    df = pd.DataFrame(rows)

    x = df['tekst']
    y = df['auteur']


    vectorizer = CountVectorizer(stop_words=None)
    X_tfidf = vectorizer.fit_transform(x)


    # Split dataset into training set and test set
    tekst_train, tekst_test, author_train, author_test = train_test_split(X_tfidf,y, test_size=0.3,random_state=109) # 70% training and 30% test

    model = LogisticRegression()
    model.fit(tekst_train, author_train)

    author_pred = model.predict(tekst_test)

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    f1_scores.append(f1score)
    precisie_scores.append(precisiescore)
    recall_scores.append(recallscore)


reference_f1 = 0.446152884786702
reference_precision = 0.5824081866740823
reference_recall = 0.4304060811887429

plt.figure(figsize=(10, 6))
plt.plot(k_values, f1_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_f1, color='m', linestyle='--', label=f'Ref. F1 = {reference_f1:.3f}')
plt.title("Federalist Papers dataset COCA-maskering F1-score uitgezet tegen k (logistische regressie)")
plt.xlabel("k")
plt.ylabel("F1-score")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, precisie_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_precision, color='m', linestyle='--', label=f'Ref. precisie = {reference_precision:.3f}')
plt.title("Federalist Papers dataset COCA-maskering macroprecisie uitgezet tegen k (logistische regressie)")
plt.xlabel("k")
plt.ylabel("Macroprecisie")
```

```
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, recall_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_recall, color='m', linestyle='--', label=f'Ref. recall = {
    reference_recall:.3f}')
plt.title("Federalist Papers dataset COCA-maskering macrorecall uitgezet tegen k (logistische
     regressie)")
plt.xlabel("k")
plt.ylabel("Macrorecall")
plt.legend()
plt.grid()
plt.show()
```

In de laatste, onderstaande, code van de Federalist Papers dataset wordt eigen maskering in combinatie met logistische regressie toegepast en worden hier grafieken van gemaakt:

```
import os
import pandas as pd
import re
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report


folder_path = 'C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Federalist papers\\
    FedPapersCorpus\\FedPapersCorpustrain'

texts = []

# Loop door elk bestand in de map
for filename in os.listdir(folder_path):
    if filename.endswith('.txt'):
        # Open en lees het bestand
        with open(os.path.join(folder_path, filename), 'r', encoding='utf-8') as file:
            content = file.read()
            # Zoek naar de positie van "author:" en behoud alleen de tekst vanaf dat punt
            author_index = content.find("Author:")
            if author_index != -1:
                content_from_author = content[author_index:]  # Behoud alles vanaf "author:"
                texts.append(content_from_author)  # Voeg toe aan de lijst
            else:
                texts.append("")  # Voeg een lege string toe als "author:" niet gevonden is



# print om te checken
#for i, text in enumerate(texts, 1):
#    print(f"Bestand {i}: {text[:100]}...")

# Dictionary om auteurs en hun teksten op te slaan
auteurs_dict = {}


for text in texts:
    author = ""
    lines_without_author = []
    regels = text.split("\n")
    for regel in regels:
        #print("Regel:", regel)
        if author != "":
            lines_without_author.append(regel)
        if regel.strip().lower().startswith("author:"):
            author = regel.split(":")[1].strip()
            #print("author:",author)
```

```
    text = '\n'.join(lines_without_author)
    #print(text)
    if " and " in author:
        for split_auteur in author.split(" and "):
            if auteurs_dict.get(split_auteur) == None:
                auteurs_dict[split_auteur] = []
                auteurs_dict[split_auteur].append(text)
            else:
                auteurs_dict[split_auteur].append(text)
    else:
        if auteurs_dict.get(author) == None:
            auteurs_dict[author] = []
            auteurs_dict[author].append(text)
        else:
            auteurs_dict[author].append(text)

#printen om te checken
#for auteur, teksten in auteurs_dict.items():
#    print(f"Auteur: {auteur}")
#    for tekst in teksten:
#        print(f"  Tekst: {tekst}")

from spellchecker import SpellChecker

spel = SpellChecker()

gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

for auteur, teksten in auteurs_dict.items():
    gecorrigeerde_teksten[auteur] = []
    for tekst in teksten:
        regels = tekst.split("\n")
        for regel in regels:
            woorden = regel.split()
            gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord]) else
 [woord] for woord in woorden]
            gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in
    gecorrigeerde_woorden]))

from collections import Counter

#k = 200

k_values = range(0, 5001, 100)
f1_scores = []
precisie_scores = []
recall_scores = []

for k in k_values:
    if k == 0:
        f1_scores.append(0)  # Als k=0, sla over (geen features mogelijk)
        precisie_scores.append(0)
        recall_scores.append(0)
        continue

    word_counter = Counter()

    for teksten in gecorrigeerde_teksten.values():
        for tekst in teksten:
            # Split de tweet in woorden en tel ze
            geenleestekens_words = re.sub(r'[^\w\s-]', '', tekst.lower())
            words = geenleestekens_words.split()
            words = [word.lstrip('-').rstrip('-') for word in words]
            word_counter.update(words)

    frequentwords = word_counter.most_common(k)

    # Print de gesorteerde lijst van woorden en hun frequentie
    #for word, count in frequentwords[:100]:
    #    print(f"{word} {count}")

```

```
    W_k = [word for word, _ in frequentwords]
    #print(W_k)

    def vervang_woorden_tekst(string, W_k):
        #woorden_string = ' '.join(string)
        woorden = string.split()
        nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in
    woorden]
        return ' '.join(nieuwe_woorden)

    final_teksten = {}

    # TEST VOOR NIEUWE MANIER
    for auteur, teksten in gecorrigeerde_teksten.items():
        final_teksten[auteur] = []
        for tekst in teksten:
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    #printen om te checken
    #for auteur, teksten in final_teksten.items():
    #    print(f"Auteur: {auteur}")
    #    for tekst in teksten:
    #        print(f"  Tekst: {tekst}")

    rows = [{'auteur': auteur, 'tekst': tekst} for auteur, teksten in final_teksten.items()
    for tekst in teksten]
    df = pd.DataFrame(rows)

    x = df['tekst']
    y = df['auteur']

    vectorizer = CountVectorizer(stop_words=None)
    X_tfidf = vectorizer.fit_transform(x)


    # Split dataset into training set and test set
    tekst_train, tekst_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    model = LogisticRegression()
    model.fit(tekst_train, author_train)

    author_pred = model.predict(tekst_test)

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    f1_scores.append(f1score)
    precisie_scores.append(precisiescore)
    recall_scores.append(recallscore)


reference_f1 = 0.446152884786702
reference_precision = 0.5824081866740823
reference_recall = 0.4304060811887429


plt.figure(figsize=(10, 6))
plt.plot(k_values, f1_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_f1, color='m', linestyle='--', label=f'Ref. F1 = {reference_f1:.3f}')
plt.title("Federalist Papers dataset eigen maskering F1-score uitgezet tegen k (logistische
    regressie)")
plt.xlabel("k")
plt.ylabel("F1-score")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, precisie_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_precision, color='m', linestyle='--', label=f'Ref. precisie = {
```

```
    reference_precision:.3f}')
plt.title("Federalist Papers dataset eigen maskering macroprecisie uitgezet tegen k (
    logistische regressie)")
plt.xlabel("k")
plt.ylabel("Macroprecisie")
plt.legend()
plt.grid()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, recall_scores, marker='o', linestyle='-', color='g')
plt.axhline(y=reference_recall, color='m', linestyle='--', label=f'Ref. recall = {
    reference_recall:.3f}')
plt.title("Federalist Papers dataset eigen maskering macrorecall uitgezet tegen k (
    logistische regressie)")
plt.xlabel("k")
plt.ylabel("Macrorecall")
plt.legend()
plt.grid()
plt.show()
```

De onderstaande code is gebruikt om de gecombineerde grafieken voor de Federalist Papers dataset te maken:

```
from tabulate import tabulate
from sklearn import metrics
from spellchecker import SpellChecker
from sklearn import svm
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
import re
import emoji
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import svm
from sklearn import metrics
from collections import Counter
from tfIdfInheritVectorizer.feature_extraction.vectorizer import TfidfVectorizer


def bestanden_inlezen(folder_path):
    texts = []

    # Loop door elk bestand in de map
    for filename in os.listdir(folder_path):
        if filename.endswith('.txt'):
            # Open en lees het bestand
            with open(os.path.join(folder_path, filename), 'r', encoding='utf-8') as file:
                content = file.read()
                # Zoek naar de positie van "author:" en behoud alleen de tekst vanaf dat punt
                author_index = content.find("Author:")
                if author_index != -1:
                    content_from_author = content[author_index:]  # Behoud alles vanaf "
    author:"
                    texts.append(content_from_author)  # Voeg toe aan de lijst
                else:
                    texts.append("")  # Voeg een lege string toe als "author:" niet gevonden
    is



    # print om te checken
    #for i, text in enumerate(texts, 1):
    #    print(f"Bestand {i}: {text[:100]}...")

```

```
    # Dictionary om auteurs en hun teksten op te slaan
    auteurs_dict = {}


    for text in texts:
        author = ""
        lines_without_author = []
        regels = text.split("\n")
        for regel in regels:
            #print("Regel:", regel)
            if author != "":
                lines_without_author.append(regel)
            if regel.strip().lower().startswith("author:"):
                author = regel.split(":")[1].strip()
                #print("author:",author)
        text = '\n'.join(lines_without_author)
        #print(text)
        if " and " in author:
            for split_auteur in author.split(" and "):
                if auteurs_dict.get(split_auteur) == None:
                    auteurs_dict[split_auteur] = []
                    auteurs_dict[split_auteur].append(text)
                else:
                    auteurs_dict[split_auteur].append(text)
        else:
            if auteurs_dict.get(author) == None:
                auteurs_dict[author] = []
                auteurs_dict[author].append(text)
            else:
                auteurs_dict[author].append(text)
    return auteurs_dict


def creeer_gecorrigeerde_teksten(auteurs_dict):
    spel = SpellChecker()

    gecorrigeerde_teksten = {} #nieuwe dictionary met verbeterde teksten

    for auteur, teksten in auteurs_dict.items():
        gecorrigeerde_teksten[auteur] = []
        for tekst in teksten:
            regels = tekst.split("\n")
            for regel in regels:
                woorden = regel.split()
                gecorrigeerde_woorden = [spel.candidates(woord) if not spel.unknown([woord])
    else [woord] for woord in woorden]
                gecorrigeerde_teksten[auteur].append(" ".join([next(iter(w), w) for w in
    gecorrigeerde_woorden]))
    return gecorrigeerde_teksten

def vervang_woorden_tekst(string, W_k):
    #woorden_string = ' '.join(string)
    woorden = string.split()
    nieuwe_woorden = [woord if woord.lower().strip('.,!?') in W_k else '*' for woord in
    woorden]
    return ' '.join(nieuwe_woorden)


def svm_coca(k, gecorrigeerde_teksten, lemma_kolom):
    if k == 0:
        return 0,0,0

    W_k = lemma_kolom.head(k).tolist()

    final_teksten = {}

    for auteur, teksten in gecorrigeerde_teksten.items():
        final_teksten[auteur] = []
        for tekst in teksten:
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

```

```
    rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
    for tweet in tweets]
    df = pd.DataFrame(rows)

    x = df['tweet']
    y = df['auteur']

    vectorizer = TfidfVectorizer()
    X_tfidf = vectorizer.fit_transform(x)

    # Split dataset into training set and test set
    tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    #Create a svm Classifier
    clf = svm.SVC(kernel='linear') # Linear Kernel

    #Train the model using the training sets
    clf.fit(tweet_train, author_train)

    #Predict the response for test dataset
    author_pred = clf.predict(tweet_test)

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    return f1score, precisiescore, recallscore

def logreg_coca(k, gecorrigeerde_teksten, lemma_kolom):
    if k == 0:
        return 0,0,0

    W_k = lemma_kolom.head(k).tolist()

    final_teksten = {}

    # TEST VOOR NIEUWE MANIER
    for auteur, teksten in gecorrigeerde_teksten.items():
        final_teksten[auteur] = []
        for tekst in teksten:
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
    for tweet in tweets]
    df = pd.DataFrame(rows)

    x = df['tweet']
    y = df['auteur']

    vectorizer = CountVectorizer(stop_words=None)
    X_tfidf = vectorizer.fit_transform(x)

    # Split dataset into training set and test set
    tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    model = LogisticRegression()
    model.fit(tweet_train, author_train)

    author_pred = model.predict(tweet_test)

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    return f1score,precisiescore,recallscore

def svm_eigen(k, gecorrigeerde_teksten):
    if k == 0:
        return 0,0,0

    word_counter = Counter()
```

```
181
182     for tweets in gecorrigeerde_teksten.values():
183         for tweet in tweets:
184             # Split de tweet in woorden en tel ze
185             geenleestekens_words = re.sub(r'[^\w\s-]', '', tweet.lower())
186             words = geenleestekens_words.split()
187             words = [word.lstrip('-').rstrip('-') for word in words]
188             word_counter.update(words)
189
190     frequentwords = word_counter.most_common(k)
191     W_k = [word for word, _ in frequentwords]
192
193     final_teksten = {}
194
195     for auteur, teksten in gecorrigeerde_teksten.items():
196         final_teksten[auteur] = []
197         for tekst in teksten:
198             final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))
199
200     rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
        for tweet in tweets]
201     df = pd.DataFrame(rows)
202
203     x = df['tweet']
204     y = df['auteur']
205
206     vectorizer = TfidfVectorizer()
207     X_tfidf = vectorizer.fit_transform(x)
208
209     # Split dataset into training set and test set
210     tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
        test_size=0.3,random_state=109) # 70% training and 30% test
211
212     #Create a svm Classifier
213     clf = svm.SVC(kernel='linear') # Linear Kernel
214
215     #Train the model using the training sets
216     clf.fit(tweet_train, author_train)
217
218     #Predict the response for test dataset
219     author_pred = clf.predict(tweet_test)
220
221     f1score = metrics.f1_score(author_test, author_pred, average='macro')
222     precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
223     recallscore = metrics.recall_score(author_test, author_pred, average='macro')
224     return f1score,precisiescore,recallscore
225
226 def logreg_eigen(k, gecorrigeerde_teksten):
227     if k == 0:
228         return 0,0,0
229
230     word_counter = Counter()
231
232     for tweets in gecorrigeerde_teksten.values():
233         for tweet in tweets:
234             # Split de tweet in woorden en tel ze
235             geenleestekens_words = re.sub(r'[^\w\s-]', '', tweet.lower())
236             words = geenleestekens_words.split()
237             words = [word.lstrip('-').rstrip('-') for word in words]
238             word_counter.update(words)
239
240     frequentwords = word_counter.most_common(k)
241
242     W_k = [word for word, _ in frequentwords]
243
244     final_teksten = {}
245
246     # TEST VOOR NIEUWE MANIER
247     for auteur, teksten in gecorrigeerde_teksten.items():
248         final_teksten[auteur] = []
249         for tekst in teksten:
```

```
            final_teksten[auteur].append(vervang_woorden_tekst(tekst, W_k))

    rows = [{'auteur': auteur, 'tweet': tweet} for auteur, tweets in final_teksten.items()
    for tweet in tweets]
    df = pd.DataFrame(rows)

    x = df['tweet']
    y = df['auteur']

    vectorizer = CountVectorizer(stop_words=None)
    X_tfidf = vectorizer.fit_transform(x)

    # Split dataset into training set and test set
    tweet_train, tweet_test, author_train, author_test = train_test_split(X_tfidf,y,
    test_size=0.3,random_state=109) # 70% training and 30% test

    model = LogisticRegression()
    model.fit(tweet_train, author_train)

    author_pred = model.predict(tweet_test)

    f1score = metrics.f1_score(author_test, author_pred, average='macro')
    precisiescore = metrics.precision_score(author_test, author_pred, average='macro')
    recallscore = metrics.recall_score(author_test, author_pred, average='macro')
    return f1score,precisiescore,recallscore

# Gebruiken we bij COCA Maskering
def svm_coca_resultaten(k_values,gecorrigeerde_teksten, frequentwords):

    lemma_kolom = frequentwords['lemma']

    f1_scores = []
    precisie_scores = []
    recall_scores = []

    for k in k_values:
        f1_score,precisie_score,recall_score = svm_coca(k, gecorrigeerde_teksten, lemma_kolom
    )
        f1_scores.append(f1_score)
        precisie_scores.append(precisie_score)
        recall_scores.append(recall_score)
    return f1_scores, precisie_scores, recall_scores


def logreg_coca_resultaten(k_values,gecorrigeerde_teksten, frequentwords):
    lemma_kolom = frequentwords['lemma']


    f1_scores = []
    precisie_scores = []
    recall_scores = []

    for k in k_values:
        f1_score,precisie_score,recall_score = logreg_coca(k, gecorrigeerde_teksten,
    lemma_kolom)
        f1_scores.append(f1_score)
        precisie_scores.append(precisie_score)
        recall_scores.append(recall_score)
    return f1_scores, precisie_scores, recall_scores

def svm_eigen_resultaten(k_values,gecorrigeerde_teksten):
    f1_scores = []
    precisie_scores = []
    recall_scores = []

    for k in k_values:
        f1_score,precisie_score,recall_score = svm_eigen(k, gecorrigeerde_teksten)
        f1_scores.append(f1_score)
        precisie_scores.append(precisie_score)
        recall_scores.append(recall_score)
    return f1_scores, precisie_scores, recall_scores
```

```
317
318 def logreg_eigen_resultaten(k_values,gecorrigeerde_teksten):
319     f1_scores = []
320     precisie_scores = []
321     recall_scores = []
322
323     for k in k_values:
324         f1_score,precisie_score,recall_score = logreg_eigen(k, gecorrigeerde_teksten)
325         f1_scores.append(f1_score)
326         precisie_scores.append(precisie_score)
327         recall_scores.append(recall_score)
328     return f1_scores, precisie_scores, recall_scores
329
330
331 folder_path = 'C:\\Users\\Isabelle\\Documents\\BEP\\Datasets\\Federalist papers\\
        FedPapersCorpus\\FedPapersCorpustrain'
332 frequentwords = pd.read_excel("C:\\Users\\Isabelle\\Documents\\BEP\\Maskeren\\wordFrequency.
        xlsx", sheet_name=1)
333 k_values = range(0, 5001, 100)
334
335 auteurs_dict = bestanden_inlezen(folder_path)
336 gecorrigeerde_teksten = creeer_gecorrigeerde_teksten(auteurs_dict)
337 svm_coca_f1, svm_coca_precisie, svm_coca_recall = svm_coca_resultaten(k_values,
        gecorrigeerde_teksten, frequentwords)
338 logreg_coca_f1, logreg_coca_precisie, logreg_coca_recall = logreg_coca_resultaten(k_values,
        gecorrigeerde_teksten, frequentwords)
339 svm_eigen_f1, svm_eigen_precisie, svm_eigen_recall = svm_eigen_resultaten(k_values,
        gecorrigeerde_teksten)
340 logreg_eigen_f1, logreg_eigen_precisie, logreg_eigen_recall = logreg_eigen_resultaten(
        k_values,gecorrigeerde_teksten)
341
342 reference_f1_svm = 0.38804359376179787
343 reference_precision_svm = 0.6795722449369963
344 reference_recall_svm = 0.39179935160285817
345
346 reference_f1_logreg = 0.446152884786702
347 reference_precision_logreg = 0.5824081866740823
348 reference_recall_logreg = 0.4304060811887429
349
350 plt.figure(figsize=(10, 6))
351 plt.plot(k_values, svm_coca_f1, marker='o', linestyle='-', color='b')
352 plt.plot(k_values, logreg_coca_f1, marker='o', linestyle='-', color='g')
353 plt.plot(k_values, svm_eigen_f1, marker='o', linestyle='-', color='r')
354 plt.plot(k_values, logreg_eigen_f1, marker='o', linestyle='-', color='purple')
355
356 plt.axhline(y=reference_f1_svm, color='black', linestyle='--', label=f'Ref. F1 SVM = {
        reference_f1_svm:.3f}')
357 plt.axhline(y=reference_f1_logreg, color='brown', linestyle='--', label=f'Ref. F1 LOGREG = {
        reference_f1_logreg:.3f}')
358 plt.title("Federalist Papers dataset F1-score uitgezet tegen k")
359 plt.xlabel("k")
360 plt.ylabel("F1-score")
361 plt.legend()
362 plt.grid()
363 plt.show()
364
365 plt.figure(figsize=(10, 6))
366 plt.plot(k_values, svm_coca_precisie, marker='o', linestyle='-', color='b')
367 plt.plot(k_values, logreg_coca_precisie, marker='o', linestyle='-', color='g')
368 plt.plot(k_values, svm_eigen_precisie, marker='o', linestyle='-', color='r')
369 plt.plot(k_values, logreg_eigen_precisie, marker='o', linestyle='-', color='purple')
370
371 plt.axhline(y=reference_precision_svm, color='black', linestyle='--', label=f'Ref. precisie
        SVM = {reference_precision_svm:.3f}')
372 plt.axhline(y=reference_precision_logreg, color='brown', linestyle='--', label=f'Ref.
        precisie LOGREG = {reference_precision_logreg:.3f}')
373 plt.title("Federalist Papers dataset macroprecisie uitgezet tegen k")
374 plt.xlabel("k")
375 plt.ylabel("Macroprecisie")
376 plt.legend()
377 plt.grid()
```

```
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(k_values, svm_coca_recall, marker='o', linestyle='-', color='b')
plt.plot(k_values, logreg_coca_recall, marker='o', linestyle='-', color='g')
plt.plot(k_values, svm_eigen_recall, marker='o', linestyle='-', color='r')
plt.plot(k_values, logreg_eigen_recall, marker='o', linestyle='-', color='purple')

plt.axhline(y=reference_recall_svm, color='black', linestyle='--', label=f'Ref. recall SVM =
    {reference_recall_svm:.3f}')
plt.axhline(y=reference_recall_logreg, color='brown', linestyle='--', label=f'Ref. recall
    LOGREG = {reference_recall_logreg:.3f}')
plt.title("Federalist Papers dataset macrorecall uitgezet tegen k")
plt.xlabel("k")
plt.ylabel("Macrorecall")
plt.legend()
plt.grid()
plt.show()
```