Extracting location context from transcripts: a comparison of ELMo and TF-IDF

David Happel[1] , **Stavros Makrodimitris**[1] , **Arman Naseri Jahfari**[1] , **Tom Viering**[1] , **David Tax**[1] , **Marco Loog**[1] ,

[1]TU Delft

{D.V.Happel}@student.tudelft.nl, {S.Makrodimitris, A.NaseriJahfari, T.J.Viering, D.M.J.Tax, M.Loog}@tudelft.nl

Abstract

Using transcripts of the TV-series FRIENDS, this paper explores the problem of predicting the location in which a sentence was said. The research focuses on using feature extraction on the sentences, and training a logistic regression model on those features. Specifically looking at the differences in performance between using ELMo and TF-IDF for this feature extraction, achieving an accuracy rate of 58% and 67% respectively on a binary classification. The paper also explores the effect of several data cleaning techniques on the results.

1 Introduction

The study of Natural Language Processing can be traced all the way back to the the 1950's. Alan Turing proposed the "Turing test"(Turing, 2009), challenging an artificial intelligence to be indistinguishable from a human through language interaction. To this day, we are still trying to achieve this goal for the purpose of making it easier for people to interact with computers in an intuitive way. This, in turn, can have great implications in many areas. For example, it can be very beneficial for the care of people who are susceptible to loneliness, like elders or mentally disabled people. Firstly, by helping those people navigate modern technologies better to keep in contact with their loved ones. Secondly, sufficiently advanced computers could even become companions themselves. Recently, the Corona virus has further elevated that need, by forcing many people to be quarantined in the hospital and at home (Armitage & Nellums, 2020). Additionally, advanced language processing could help writers with creative writing; Or even aide professionals like historians or intelligence operators in interpreting textual sources.

The use of machine learning has given promising results in this field. Noticeably, in the last decade, where breakthroughs have been made in finding and representing the meaning of a word, word embedding. Starting with word2vec, developed by Google (Mikolov et al., 2013), which is a technology allowing a word to be represented as a weighted sum of abstract features, by looking at which words are often used together. This in turn, allows us to determine to what extend two words are related. Since then, word embedding performance has been improved and expanded to context awareness and sentence embedding, using techniques like latent variable generation (SIF)(Arora et al., 2016), bi-directional LSTM (ELMo) (Alammar, 2018; Taylor, 2019), and Transformers (BERT) (Alammar, 2018; Devlin et al., 2018).

For a computer to offer intuitive conversation, it needs to not only understand the literal definition of what is being said. It also needs to understand the context of the conversation. Information like who is speaking, and where the conversation is taking place. That is at the core of this research, extracting this context information from conversation. In our case, specifically the location. This research aims to contribute to this aspect by exploring to what extent it is possible to infer the location of a conversation using existing Language processing and classification techniques. I will be examining this in the context of the TV-series FRIENDS. For this show, transcripts with location data are freely available.

By combining sentence embedding with classification models like logistic regression (Jurafsky & Martin, 2008, Chapter 5), we can get concrete classifications from pieces of text. This can, for example, be used to classify a piece of text as spam (Alammar, 2018), or for author identification(Green & Sheppard, 2013; Stamatatos et al., 2000). I have however not been able to find the technique being used to get specific context information like location from a conversation transcript. Some research has been done using "Named Entities" to extract location from text (Lingad et al., 2013). This aims however to find a specific geographic location, while my research aims at finding a more contextual location, like a living room, or a restaurant. This is an important distinction to make.

This research looks only at extracting the location from individual sentences. It is obvious that many sentences do not contain any information about the location, and could have been said anywhere. Especially in the show FRIENDS, where many conversations about relationships, drama, jokes, etc. happen. Taking this into account, the main goal of this research is to understand how many sentences do contain references to the current location, and how to best extract this information. The expectation is that these techniques will only perform marginally better than random guessing, which I will consider the baseline for this research.

2 Techniques

In order to classify textual data, first it needs to be transformed to a format that is more easily to process for a computer. Simply taking the numerical representations of the characters in sequence, and feeding that to a classifier would not be a practical option. Since the classification model would need to learn to interpret language from scratch. This is a very difficult task that requires a very large and complex model. Another problem when working with raw text is that the differences in the input lengths makes it hard for many models to interpret. Instead, I opt for a smarter method. Feature extraction. This is the process of taking a piece of text, and transforming it into list of numeric features. These features could be hand-crafted by humans or, as I will explore in this research, generated by an algorithm. Since this technique transforms a piece of text into numeric list of a set length (by making non-existent features 0), this also solves the problem differences in input length. A list of extracted features from text will also be called an embedding of the text. We still need a machine learning model to classify these embeddings. However, this model can now be a lot less complex, since it only needs to interpret a fixed length search space of continuous features. Before embedding, pre-processing can be done on the lines to remove irrelevant information.

Following this structure, the solutions I explore generally consist of three main parts: Pre-processing, embedding, and classification. Firstly, I will discuss some techniques for Feature extraction.

2.1 ELMo

In 2018, a new model for ”Deep contextualized word representations” was released (Peters et al., 2018). It provided lower error margins over a set of 6 diverse benchmarks. It works by stacking several bi-directional LSTM networks, each feeding into the next one. A LSTM network is a form of recurrent neural network that makes use of both short-term and long-term memory. For each token in a sentence, each layer attempts to predict the next and previous token in the sentence. The result of one layer is passed to the next one. In order to reduce the input space for the LSTM models, the tokens are not chosen to be words, but individual letters. Then, the resulting output vectors from the final, and hidden layers are combined together in a weighted sum. These weights are also learned parameters. Pre-trained versions of this model are available, trained on the 1 Billion Word Benchmark (Chelba et al., 2013). The parameters for the weighted summing for the output are optionally trainable for individual tasks.

2.2 TF-IDF

In contrast to the complex ELMo model attempting to capture contextualized representations of the sentences, TF-IDF is quite a simple method. It looks at the frequency in which words appear in a document, compared to other documents. Specifically, for each included word, it computes the product of two terms, the TF (Term Frequency) and the IDF (Inverse Document Frequency) (Wu et al., 2008) (“TF-IDF A Single-Page Tutorial - Information Retrieval and Text Mining”, n.d.). For the purpose of this research, these ”documents” are individual sentences. The individual components of the TF-IDF score are calculated as such:

TF (Term Frequency): The amount of times a word occurs in a sentence, normalized by dividing by the total amount of words in the sentence.

IDF (Inverse Document Frequency): The total amount of sentences, divided by the amount of sentences that contain the word. From this we take the logarithm.

TF-IDF: TF $\times$ IDF

TF-IDF on a sentence: A sparse matrix containing the sentence’s TF-IDF calculation for each word in the corpus. Each word not present in the sentence is set to 0.

2.3 Logistic Regression

For classification I use a Logistic Regression (Jurafsky & Martin, 2008, Chapter 5) model. Logistic Regression is a simple linear machine learning model. For each feature, it learns how much it contributes to the output label, storing this as a weight. When predicting, the model output simply represents a weighted sum of all the features. A possible problem with training self-learning models, is them conforming too specifically to the training data, and not learning to generalize (over-fitting). In order to prevent this, regularization can be used. It prevents the weights from becoming too large, and in turn, reduces how much the algorithm relies on very particular configurations of features. There is several forms of regularization. L1 regularization normalizes based on the Manhattan distance of the weights vector to the origin (the total sum of the weights). L2 regularization normalizes based on Euclidean distance (the length of the weights vector).

2.4 Data Cleaning

Textual data contains a lot of information that might not be relevant to our problem of finding the location. For example: punctuation, or numbers. These characters represent extra dimensions for the embedding algorithm and classifier to take into account, making it harder for the algorithm to learn what features are important. For this reason we clean the data.

Another technique we can perform is lemmatization (Prabhakaran, 2018). This transforms words into their base form. For verbs this means transforming it to the base verb. And for nouns this means making them singular. Removing or replacing pronouns might also be a part of this.

- Caring $\rightarrow$ Care
- Are $\rightarrow$ Be
- Cars $\rightarrow$ Car

It is also useful to remove lines that are very short. In the transcripts for example. Many lines are just very short confirmations or greetings like: ”yes!” or ”Hello”. We can be pretty confident that these do not contain any useful information regarding the location of the line. It could be beneficial to remove them from the data set to reduce noise. Taking into account that you will also have to remove short lines from actual unlabeled data when using the eventual model.

3 Method

3.1 Data

In this research I explore and compare the applications of these techniques for solving the problem of guessing the location of a performed line in the show FRIENDS. For the purpose of simplicity, only two locations will be included in the experiments. Meaning that the classifier will only have to choose between two locations and making it a binary classification problem. This makes reading and interpreting the results more intuitive. While this research does not include results for more than two locations, in theory all these techniques could work for any amount of location labels with minor changes to the classifier. The two locations I chose to use are "Central Perk" and "Monica and Rachel's". These are two of the most prevalent locations in the show with both a roughly equal amount of sentences. The locations also represent two different contexts, a coffeehouse and an apartment respectively, which is likely to be reflected in the conversations. While these locations both appear roughly the same amount of times in the show, still the data is balanced by use of under-sampling the larger of the two groups. This happens after the cleaning step, since filtering out short sentences changes the amount of lines per location.

In order to properly evaluate the results, we split the data into training sets and testing sets using 5-fold cross-validation. This means the training and testing sets are of fractions 4/5 and 1/5 respectively.

Transcripts for the TV-series FRIENDS are provided by GitHub user puneeth019[1]. In order to properly perform calculations on these transcripts, they need to be split up into a list of sentences, each linked to the location they were said at. As previously mentioned. Only the sentences from the two chosen locations are included. Since the transcripts includes some inconsistencies and spelling errors regarding locations names, I include not only the locations that match literally to the selected locations. Instead I include the locations that match with Levenshtein similarity (Levenshtein, 1966) of at least 0.7 to "central perk" and "monica and rachels". Still, the data might contain some small errors.

Further details of the parsing are not very relevant for the purpose of this research, so they will be disregarded here. The code used is available in the source repository[2].

3.2 Setup

The setup consists of 5 main steps:

1. Parsing data
2. Cleaning
3. Embedding
4. Training Classifier
5. Evaluating Classifier

A visual representation of the data flow through these steps can be seen in Figure 1

Parts of the of the implementation of setup was inspired by the work of my peer researchers. Specifically, configuration management by Thomas van Tussenbroek[3], and parsing by Pia Keukeleire[4].

3.3 Cleaning

In this setup the cleaning is performed on both the training, and validation data. The following cleaning steps can be performed in their respective order:

1. Lemmatization
2. Remove numbers
3. Remove short sentences of less than n words.
4. Remove punctuation

By individually enabling/disabling steps 1, 2, and 4, we can evaluate the influence each step has on the accuracy.

I will look at the 3'rd step separately, by changing the minimum words a sentence needs to have. It is important to note that while only looking at very longer sentences can have a positive impact on the results, it also makes the resulting model only usable on very long sentences. So keeping the eventual use of this research in mind, I will only consider a minimum word length of up to 5 words. This means that for step 3: $1 \leq n \leq 5$.

Lemmatization

Lemmatization is done using the spaCy Language Processing model (Honnibal & Montani, 2017). This lemmatization also replaces pronouns with the placeholder "-PRON-". I opted to remove these placeholders from the data entirely as part of the lemmatization, since this yields the best results.

3.4 Embedding

ELMO

For the evaluation of using ELMo for embedding, I adapted code from Karan Purohit [5]. Using the pretrained ELMo V3 available through TensorFlow Hub [6]. As previously mentioned, it is possible to train the hyper-parameters used for summing the intermediate layers.

TF-IDF

For TF-IDF I use an existing implementation, the TfidfVectorizer from SKLearn (Pedregosa et al., 2011). For a list of sentences, it returns a very sparse matrix containing the calculated TF-IDF values for each word in the corpus, per sentence. The implementation offers for a built-in option to ignore stop words [7]. Stop words are words generally considered to not contain contextual information. However, obviously this depends on the specific context whether this is truly the case.

[1]https://github.com/puneeth019/FRIENDS

[2]https://github.com/David-Happel/scene-location-NLP

[3]https://github.com/thomasvant/Character-classification

[4]https://github.com/pi-zz-a/Text-Generation

[5]https://medium.com/saarthi-ai/elmo-for-contextual-word-embedding-for-text-classification-24c9693b0045

[6]https://tfhub.dev/google/elmo/3

[7]https://scikit-learn.org/stable/modules/feature_extraction.html#stop-words

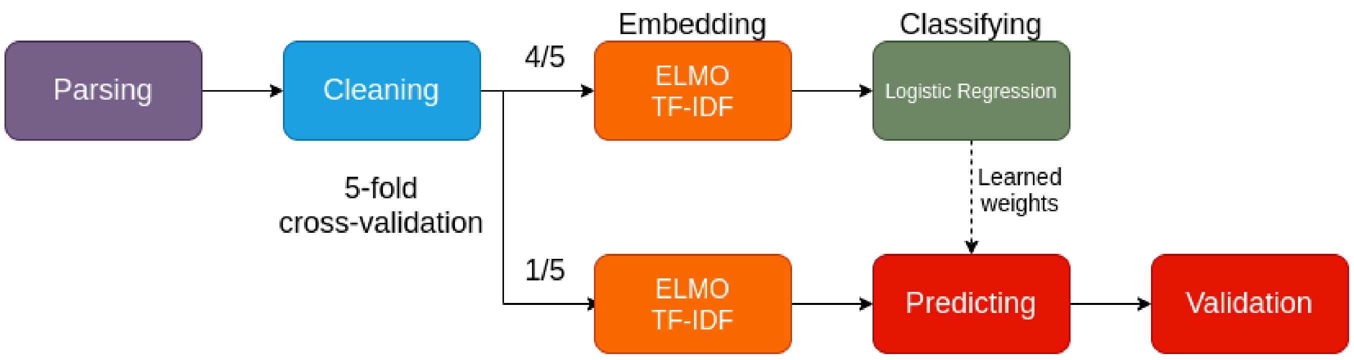

Figure 1: Conceptual Experimentation setup data flow.

Classification
For the classification I use SKLearn's logistic regression implementation (Pedregosa et al., 2011). I use its default setting for regularization type, which is L2 regularization. This regularization has a parameter C, the inverse of the regularization coefficient. For both ELMo and TF-IDF, I use SKLearn's grid-search to determine the best values for the C parameter. Using 5-fold cross-validation and optimizing for accuracy. The maximum iterations parameter is also included in the grid search.

4 Experimental Results

4.1 Parameter Grid-Search

By performing grid-search on the parameters needed for logistic-regression, we can find the most effective values for them. The grid-search was performed on the parameters: C, and the max iterations. The maximum iterations parameter is not included in the figures since for any value over 100, this did not impact the performance significantly. Consequently, for the displayed results the max iterations is set at 200. For the following experiments we perform the following cleaning steps: remove numbers, remove sentences shorter than 2 words, and remove punctuation. Lemmatization is not performed.

ELMo
The results of grid search on ELMo for feature extraction can be seen in Figure 2. The best results of 58.2% accuracy were obtained with a value of C = 10. However C = 100 provides very similar results far within the margin of variance across runs. The latter has been used in further experimentation. Furthermore, enabling the option for the hyper-parameters to be fine-tuned, did not improve performance. Consequently, I have opted to use the ”default” output instead.

TF-IDF
Similar to before, using grid-search when training the Logistic Regression model on the TF-IDF generated data gives us the best value for C: Figure 3. Here, the best results of 64.1% accuracy were obtained with value C = 10000. This value is used in further experimentation. Ignoring stop words did not improve the results, and was not used for any of the experiments featured in this report.

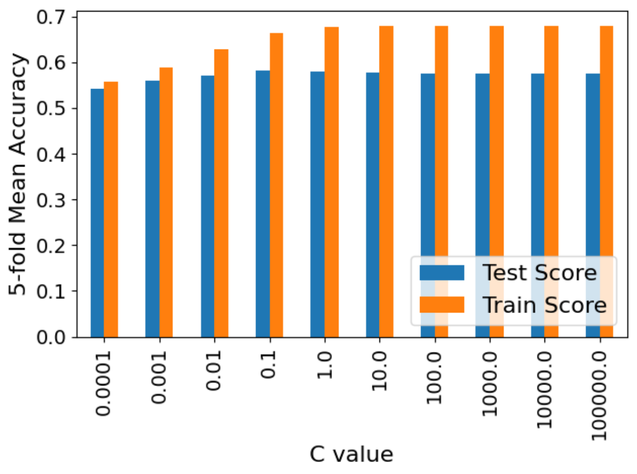

Figure 2: Grid search for C parameter for logistic regression on ELMo embeddings using 5-fold validation. X-axis: Value for parameter C, the inverse of the regularization coefficient. Y-axis: Mean accuracy on the validation set (blue) and the testing set (orange).

4.2 Cleaning

The previous results were obtained using the following of the before mentioned cleaning steps in their respective order, because of the configurations relatively good performance with both embedding algorithms:

1. Lemmatization: No
2. Remove numbers: Yes
3. Remove sentences of 2 words or shorter: Yes
4. Remove punctuation: Yes

In the following section I will show the effect the mentioned cleaning steps have on the accuracy scores when using either ELMo or TF-IDF. Firstly, looking at the effect of removing punctuation, removing numbers, and lemmatization. This is done by adapting the above standard configuration and individually enabling/disabling these techniques. Using 5-fold cross-validation. The results can be seen in Table 1.

The previous results also make use of removing sentences of 2 word or shorter. The minimum amount of words is also a parameter that we can change. In a similar manner as before, the impact that this has can be seen in Table 2. Of course,

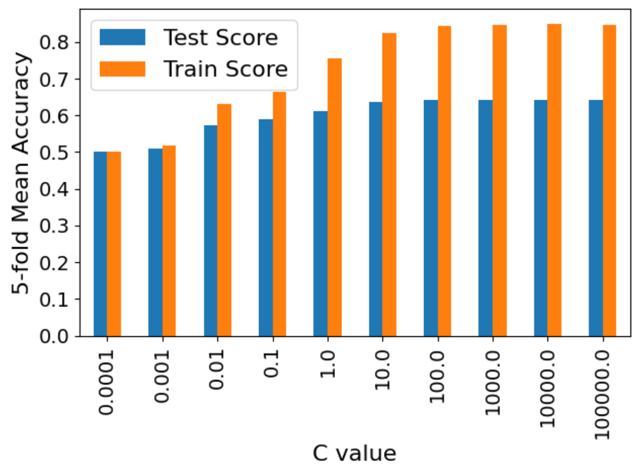

Figure 3: Grid search for C parameter for logistic regression on TF-IDF embeddings using 5-fold validation. X-axis: Value for parameter C, the inverse of the regularization coefficient. Y-axis: Mean accuracy on the validation set (blue) and the testing set (orange).

Embedding	Lemmatization	Rem. Numbers	Rem. Punctuation	Accuracy	STD
ELMo	✗	√	√	58.3%	0.7
ELMo	√	√	√	58.1%	0.5
ELMo	✗	✗	√	58.0%	0.6
EMLo	✗	√	✗	57.9%	1.2
TF-iDF	✗	√	√	64.4%	1.1
TF-iDF	√	√	√	62.9%	0.9
TF-iDF	✗	✗	√	64.9%	0.6
TF-iDF	✗	√	✗	64.0%	0.6

Table 1: The accuracy results obtained with different cleaning configurations when embedding using ELM0 and TF-IDF respectively. Accuracy: The mean accuracy of 5-fold cross-validation. STD: standard deviation of the different validation results.

when removing sentences, the amount of data data available changes. Table 3 shows the size of the testing and training sets size per minimum word count.

4.3 TF-IDF insights

While the features from the ELMo embeddings do not have a clean and intuitive meaning to them, features from TF-IDF are linked to words. An advantage to this is that we can peek at the logistic regression's reasoning by looking at the weights associated with those TF-IDF features. The words of the TF-IDF features most associated with either location are displayed in Table 4.

5 Responsible Research

This research was performed as part of the course CSE3000 at Delft University of Technology. To accommodate full reproducibility, all code that was used to generate the results presented in this report is available on GitHub[8] and all relevant parameters used to obtain the results are disclosed. While the results of the techniques in this research are not immediately alarming, more advanced iterations of this research could raise privacy concerns. It is easy to imagine how a computer accurately guessing the location of a conversation transcript could be used in the field of surveillance. This is something to be mindful of when working on natural language processing techniques.

[8]https://github.com/David-Happel/scene-location-NLP

Embedding	Minimum Words	Accuracy	STD
ELMo	None	57.6%	0.9
ELMo	2	58.4%	0.4
ELMo	3	58.2%	1.1
EMLo	4	58.3%	1.1
EMLo	5	58.8%	0.5
TF-iDF	None	63.0%	0.4
TF-iDF	2	64.9%	1.3
TF-iDF	3	64.7%	0.7
TF-iDF	4	65.1%	0.6
TF-iDF	5	66.2%	1.1

Table 2: The accuracy results obtained when filtering the data for different minimum sentence lengths when embedding using ELM0 and TF-IDF respectively. Accuracy: The mean accuracy of 5-fold cross-validation. STD: standard deviation of the different validation results.

Minimum Words	None	2	3	4	5
Test-set	13561	11748	10673	9692	8822
Train-set	3391	2938	2669	2424	2206

Table 3: Size of training and testing sets for each minimum word count.

6 Discussion

6.1 ELMo vs TF-IDF

The results show that when using elmo for embedding, it is possible to get 58.8% accuracy. When using TF-IDF, the accuracy can get up to 66.2%. TF-IDF clearly performs better.

Looking at Figures 2 and 3. For both ELMo and TF-IDF the mean accuracy on the training set is higher than on the testing set. From this, it is possible to conclude that the model is over-fitting to some extend. Increasing the regularization decreases the relative difference between the training and validation accuracy considerably. However, this does not significantly result in a better overall performance on the validation set. Only for ELMo, does the higher regularization slightly improve accuracy before hurting it, the best result being at C = 10. For TF-IDF, extremely low regularization gives the best results.

Very high regularization hurts performance considerably, which is to be expected since high regularization reduces the learned weights towards the origin, reducing learning capacity. For ELMo and TF-IDF, Performance start to suffer at $C \leq 0.01$ and $C \leq 10.0$ respectively

6.2 Cleaning

The results from Table 1 show that the two embedding techniques are affected differently by the different cleaning steps. Both suffer from the use of Lemmatization. When using ELMo, removing numbers and punctuation get the best accuracy. While TF-IDF does best when removing punctuation but not numbers. Why not removing numbers works

Monica and Rachel's	Central Perk
paulo	jealous
airport	spontaneous
earrings	jill
snapped	affects
dues	teeth
breast	felt
kitchen	shutting
squad	sudden
tasted	dollar
nuhuh	imim
accent	yell
andandand	russ
poem	needed

Table 4: Words corresponding to the TF-IDF scores most associated with locations ”Monica and Rachel's” and ”Central Perk”.

best when using TF-IDF is not entirely clear. It could be that money amounts are mentioned more often in the Lunchroom environment than at the apartment.

It is clear from Table 2 that both methods do generally benefit from removing short sentences, especially TF-IDF. ELMo also slightly improves when with longer sentences, however not nearly by as much. This makes sense since with TF-IDF each distinct word will be a feature. Sentences with very few words will have very few features that are non-zero, activating only a very small part of the classifier. Short sentences also simply contain less information, explaining the better performance for both techniques. In real world applications the benefits of slightly more accurate results on longer sentences should be weighed against the reduced usability on short sentences when choosing a minimum sentence length.

Interestingly, both techniques experience a small drop in performance after eliminating 2-word sentences. I have no obvious explanation for this. Possibly there is a set of 2-word sentences with clear correlation to a location, enough to counter the general trend.

6.3 TF-IDF Feature Interpretation

While some of the words in Table 4 do not make immediate sense, it is reassuring to see the word ”kitchen” being associated with the apartment, and ”dollar” with the lunchroom. Some other of the words represent names or expressions, the latter likely do not appear in the data-set very often, and probably do not bear any real significant link to locations in real world applications. Identifying these words could help improve data cleaning in future research. It is, however, important to realize that a correlation the machine learning algorithm has learned, which does not seem intuitive to a human, could still be a valid. An additional consideration is that, while these are the words with the highest weights associated to them, each sentence only contains a very limited number of words. So many of the classifications will be purely decided by features with smaller weights.

This makes apparent a considerable limitation of the current method. The lines are being shuffled completely randomly, not keeping track of what lines belong together in conversation. It might be that specific words that are mentioned often in only one conversation artificially inflate the accuracy. Since some of the lines in that conversation could end up in the testing set, these words then do not contain any real information about the location. TF-IDF already somewhat deals with this since the IDF part takes into account how often that word is mentioned, limiting the TF-IDF value of infrequent words. However in a difficult task like this, this effect could still have a significant impact. One issue to deal with this, could be to completely disregard words that appear only a limited number of times.

There is another solution that this research was not able to touch upon however. Looking outside of the scope of a single sentence. When classifying individual sentences the accuracy does not exceed 70% and might even hit a hard limit related to the limited amount of information contained in a sentence. However, we can also keep the sentences grouped by their conversation. And when predicting, we average the predictions of all sentences in a complete conversation together. Then we assign the predicted location of the complete conversation, to all sentences in the location. We avoid the problem of topics in conversations artificially increasing the accuracy on the testing set. Since sentences are kept grouped by conversation, and a conversation is never split up between the test and training set. At the same time, we can likely get a higher accuracy. Since the average of many guesses is will likely be more accurate than the individual guesses separately, as described in the law of large numbers(Hsu & Robbins, 1947). The effects of predicting for entire conversation is something that I encourage future research to explore.

6.4 Fitting TF-IDF

Before the sentences can be vectorized using TF-IDF, first the model is fitted on the training data. For this fitting it calculates the IDF (Inverse Document Frequency) part of the equation for each word. After fitting, when calculating the actual embeddings for all data, words that have not been seen during the fitting step are not included. For the results in this paper TF-IDF was only fitted on the training data. However the labels are not needed for this step. This makes it is not unthinkable to also fit TF-IDF on the testing data, as is done by McClure in the book ”TensorFlow machine learning cookbook” (McClure, 2017, Chapter 7). This generates more complete features for the sentences and likely increase eventual accuracy. This increased accuracy does come at a cost though, it means that for each batch of new unlabeled data, both TF-IDF and the classifier would need to be re-fitted. Whether this is useful to do if the model is to be used in production would depend entirely on the circumstances. For once, this would only create significant difference if the unlabeled data would come in sizeable batches compared to the training data. Additionally, if the unlabeled data includes data that is not representative of the rest of the data-set, this ”bad” data will likely worsen the models performance on the other unlabeled data.

6.5 Larger Classifier Model

Another consideration is the fact that higher regularization in the classifier does not significantly increase performance for either embedding algorithm. This could be an indicator that the Logistic Regression model has hit its capacity of the amount of information that can be encoded into its weights. This, in turn, might mean that using a more intricate classifier model, for example by adding an extra hidden layer, could lead to a more intricate generalization and better results. This was briefly investigated in this research, but not pursued to the end in the interest of the scope of the research.

7 Conclusion

The results presented in this paper have shown that it is possible to infer which one of two locations a line in the TV-series FRIENDS was performed at upwards of 66% of the time when using TF-IDF with logistic regression. While this accuracy is only 16% above randomly guessing, it exceeds the initial expectations. This research has also shown that the use of the relatively simple statistical TF-IDF method performs better at this task than the complex context aware ELMo embeddings, which yielded an accuracy of up to 59%.

References

Alammar, J. (2018). The illustrated bert, elmo, and co. (how nlp cracked transfer learning). https://jalammar.github.io/illustrated-bert/

Armitage, R., & Nellums, L. B. (2020). Covid-19 and the consequences of isolating the elderly. *The Lancet Public Health*, *5*(5), e256.

Arora, S., Liang, Y., & Ma, T. (2016). A simple but tough-to-beat baseline for sentence embeddings.

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., & Robinson, T. (2013). One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Green, R. M., & Sheppard, J. W. (2013). Comparing frequency-and style-based features for twitter author identification, In *The twenty-sixth international flairs conference*.

Honnibal, M., & Montani, I. (2017). Spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*.

Hsu, P.-L., & Robbins, H. (1947). Complete convergence and the law of large numbers. *Proceedings of the National Academy of Sciences of the United States of America*, *33*(2), 25.

Jurafsky, D., & Martin, J. H. (2008). Speech and language processing: An introduction to speech recognition, computational linguistics and natural language processing. *Upper Saddle River, NJ: Prentice Hall*.

Levenshtein, V. I. (1966). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, *10*, 707.

Lingad, J., Karimi, S., & Yin, J. (2013). Location extraction from disaster-related microblogs, In *Proceedings of the 22nd international conference on world wide web*.

McClure, N. (2017). *Tensorflow machine learning cookbook*. Packt Publishing Ltd.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality, In *Advances in neural information processing systems*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikitlearn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations, In *Proc. of naacl*.

Prabhakaran, S. (2018). Lemmatization approaches with examples in python. https://www.machinelearningplus.com/nlp/lemmatization-examples-python/

Stamatatos, E., Fakotakis, N., & Kokkinakis, G. (2000). Automatic text categorization in terms of genre and author. *Computational linguistics*, *26*(4), 471–495.

Taylor, J. (2019). Elmo: Contextual language embedding. https://towardsdatascience.com/elmo-contextual-language-embedding-335de2268604

Tf-idf a single-page tutorial - information retrieval and text mining. (n.d.). http://www.tfidf.com/

Turing, A. M. (2009). Computing machinery and intelligence, In *Parsing the turing test*. Springer.

Wu, H. C., Luk, R. W. P., Wong, K. F., & Kwok, K. L. (2008). Interpreting tf-idf term weights as making relevance decisions. *ACM Transactions on Information Systems (TOIS)*, *26*(3), 1–37.

A Coefficient Weights

To provide some insights into the models decisions, I extracted the weights for each feature from the logistic regression model. These feature weights, or coefficients, are ordered from low to high and displayed in Figure 4 for the ELMo embeddings and Figure 5 for TF-IDF embeddings. A high coefficient means the feature points to the location ”Central Perk” while a low coefficient mean the feature is associated with the location ”Monica and Rachel’s” apartment.

For both embedding techniques, the coefficients resemble a normal distribution, where most features are around the center and few features have the biggest impact on the result. These coefficient distributions are also quite balanced. The median coefficient of the ELMo and TF-IDF features are 0.008 and 1.371 respectively. One big difference that can be seen between the two techniques, is that the ELMo features mostly have coefficients between -1 and 1, while those of TF-IDF range between -40 and 40. This can be explained by the difference of the regularization parameter C. As discussed previously, the C parameter is set to 100 for ELMo and 10000 for TF-IDF. This means that ELMo uses higher regularization, and big coefficient values are punished more.

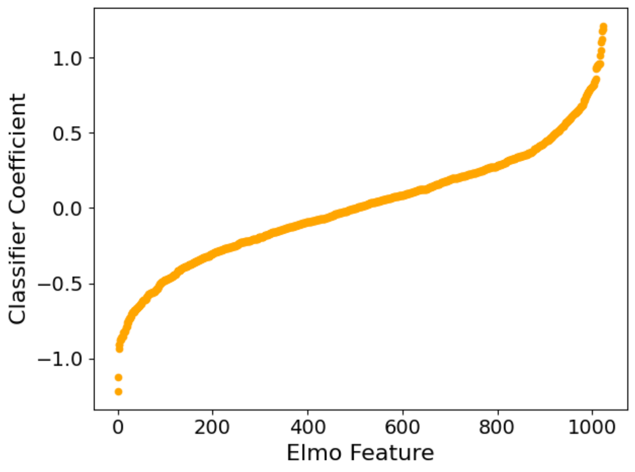

Figure 4: The feature coefficients from logistic regression trained on the ELMo embeddings, ordered from low to high.

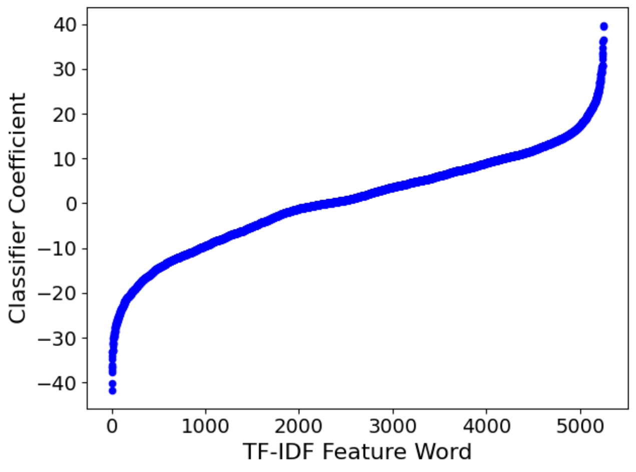

Figure 5: The feature coefficients from logistic regression trained on the TF-IDF embeddings, ordered from low to high.