A Deep Graphical Model for Spelling Correction

Stephan Raaijmakers [a]

[a] *TEXAR Data Science, Vijzelgracht 53C 1017 HP Amsterdam*

Abstract

We propose a deep graphical model for the correction of isolated spelling errors in text. The model is a deep autoencoder consisting of a stack of Restricted Boltzmann Machines, and learns to associate error-free strings represented as bags of character n-grams with bit strings. These bit strings can be used to find nearest neighbor matches of spelling errors with correct words. This is a novel application of a deep learning semantic hashing technique originally proposed for document retrieval. We demonstrate the effectiveness of our approach for two corpora of spelling errors, and propose a scalable correction procedure based on small sublexicons.

1 Introduction

In this paper, we investigate the use of deep graphical models for the purpose of context-free spelling correction in text: spelling correction in isolated words. Spelling correction is a well-studied subject that has been on the agenda of the NLP community for decades ([3]). Nowadays, new applications arise in the contexts of big data, search engines and user profiling. For instance, in product data cleansing ([4, 19]), where large quantities of manually entered product data need to be standardized, accurate detection and handling of misspellings is important. Search engines like Google proactively perform spelling correction on user queries ([5, 15, 17, 25]). Stylometric user profiling in forensic situations may benefit from accurate estimates of the amount and type of spelling errors a person makes, e.g. for authorship verification (e.g. [13, 23]). The approach we propose is an application of a recent neural network technique that was proposed by Salakhutdinov and Hinton ([20]) for document hashing and retrieval. This technique, *semantic hashing*, attempts to compress documents into compact bit strings, after which document retrieval becomes an efficient and accurate operation on binary hash codes. In Section 2, we outline this approach, and describe how it can be used for spelling correction, which we view as a form of document retrieval at word level. In Section 3, we describe a number of experiments on two corpora of spelling errors. Results are discussed in Section 4.

2 Deep graphical models

Deep graphical models have emerged in the machine learning community as hierarchically structured, powerful learning methods that perform 'deep learning': hidden layer information of subordinate layers is exploited as feature input by higher layers, the idea being that every layer computes features of its input (hence, higher levels compute meta-features, 'features of features'). This higher-order type of information is generally held to capture deep or semantic aspects of input data, and seems to correlate with the human brain architecture for cognitive processing ([1]). Deep learning has been applied to challenging problems in vision and speech and language processing (see e.g. [1, 11, 21]). Frequently, deep learning architectures are composed of stacks of Restricted Boltzmann Machines (RBM's): generative stochastic networks that learn probability distributions over inputs. Boltzmann Machines are stochastic neural networks with a layer of visual neurons (input units) and a layer of hidden neurons. Restricted Boltzmann Machines have no connections from visible units (inputs) to visible units, or from hidden units to hidden units, making them bipartite graphs (see Figure 1).

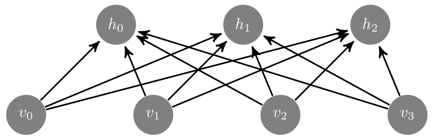

Figure 1: A Restricted Boltzmann Machine.

Often, the neurons are binary stochastic neurons than can be in two states: on or off. Whether a neuron is on depends (via a sigmoid function) on node-specific properties (like a bias), and the weighted connections the node has with the other neurons in the network, as well as the states of those other neurons.

The *energy* of a RBM at any time is given by $E(v,h) = h'Wv + b'v + c'h$, with W the weight matrix, and b and c bias vectors of the visible (v) and hidden (h) neurons. Training a Boltzmann machine means to feed input to the network through the visible neurons, and updating weights and biases with the purpose to minimize the energy of the network. Efficient training methods for RBM's have been proposed by [8, 9], based on alternating Gibbs sampling of the hidden units based on the visible units, and vice versa . In [20], an application of a stack of RBM's is presented for document hashing. The stack is trained to associate training documents with bit strings (binary codewords), which can be interpreted as memory addresses, and allow for finding similar test documents with fast and simple bit shift and Hamming distance operations. First, training proceeds in an unsupervised, greedy manner, mapping vector representations of documents to lower-dimensional real-valued vectors. After this stage, these output vectors are converted into binary vectors, which are used to reconstruct the original training documents. This is done by pairing the feedforward phase (generating bit strings) with a feedbackward phase that predicts the original training input from the binary vectors. Reconstruction errors are minimized with gradient-descent backpropagation in a supervised fine-tuning stage. Finally, the stack of finetuned RBM's can be used for the prediction of bit strings for arbitrary documents based on the 'training' on the unlabeled, original training documents. Systems of this type are called *autoencoders* ([10]): they learn how to compress data into low-dimensional codewords. In the next subsection, we describe the details of a similar architecture we use in our experiments to handle the problem of spelling correction in isolated words.

2.1 Model architecture for spelling correction

Our proposal is based on the idea of treating words as 'documents', and spelling correction as a form of document retrieval: retrieving the best matching correct spelling for a given input. Our hypothesis is that semantic hashing-based document matching can be applied here too, even if the documents are tiny compared to 'normal' documents. In our approach, bit strings are learned from training words (correct spellings), and a corresponding dictionary of bit strings paired with correct spellings is derived. For queries, the model makes a bit string prediction, which is then matched against the bit string dictionary derived from the training data. The bit string with closest Hamming distance to the predicted bit string produces the proposed spelling for the query word. This approach is illustrated in Figure 2. We expand words to quasi-documents using a bag of character n-grams representation. An example is

- (bigrams) `spelling`: `_s,sp,pe,el,li,in,ng,g_`

Using character n-grams allows for approximate matching of strings, since partial matches are possible. Character n-grams have been found to be strong features for many applications in text analytics, such as polarity classification of sentiment in documents (e.g. [18]). The bag-representation discards order, and usually contains frequencies of elements (like words, or, in our case, character n-grams)[1]. Like [20], we

[1]The unordered character n-gram representation may lead to discriminative difficulties in the rare case of two words that share the same n-grams, but differ from each other in the position of some n-grams. Adding position indexes to n-grams may prove beneficiary

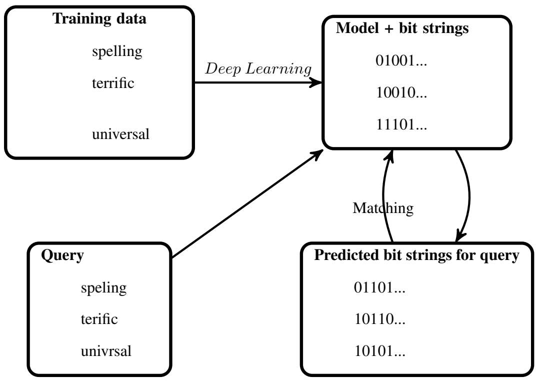

Figure 2: System setup. A model for the prediction of bit strings is learned from correctly spelled words. The closest match of learned bit strings with predictions yields the correction of the input.

used a uniform model architecture in our experiments, with the following characteristics. We stack three primary RBM's with binary hiddden units, which have a $(N\ (input) \times N/2\ (hidden)) - (N/2\ (input) \times N/2\ (hidden)) - (N/2\ (input) \times 128\ (output))$ structure. Here, N is the dimension of the input layer (the number of features), which we set to be $\min(N_f, 1000)$, with N_f the amount of different features in the training data. We generate bit strings of length 128, a value that was determined on the basis of some experiments on a fraction of the training data. These RBM's are connected to their inverse counterparts, in order to reconstruct the original training data from the bit string predictions. The graphic structure of the total, *unrolled* model is depicted in Figure 3.

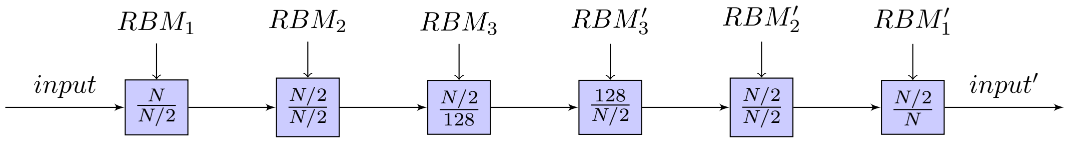

Figure 3: Model architecture. Three RBM's are pre-trained, and then reversed ('unrolled').

The feature representation of the input layer then consists of a vector of $N\ \log(1 + c_f)$ values, with c_f the raw frequency of feature f in the bag of character n-grams derived for an input word. In our setup, the three first Boltzmann machines are pretrained for 500 epochs. Subsequently, the chain is unrolled (i.e. put in reverse mode), and the training data is reconstructed ($input'$) from the predicted bit strings. This is followed by a gradient descent backpropagation phase (100 epochs) that feeds back the measured reconstruction error (the error between original training data and reconstructed, predicted training data) into the chain, and optimizes the various weights. Finally, after backprogation, the three trained and optimized RBM's are stored as the final model. Predicted values by the model are discretized to binary values by simply thresholding them (values greater than or equal to .1 becoming 1, following [20], who mention that RBM's have asymmetric energy functions that tend to produce values closer to zero than to 1). During training, the learning rate was set to .1, and no momentum nor weight decay was used. A momentum of .9 was used for backpropagation. No systematic grid search was attempted due to the rather unfavorable time complexity of the training procedure.[2] Our approach differs in some respects from the approach by Salakhutdinov and Hinton [20]. Unlike their setup, which uses RBM's based on Constrained Poisson Models, we used standard Restricted Boltzmann Machines. In addition, we did not apply any optimization techniques to enhance the binarization of the output predictions. In [20], deterministic Gaussian noise was added to the inputs received by every node in the chain, in order to produce better binary codes. We merely relied on backpropagation

in such cases, with the position indices being string parts of the n-grams, e.g. ab_1, bc_2.

[2]According to [20], deep graphical models are somewhat insensitive to perturbations of hyperparameters.

for bit string optimization after the initial bit string predictions were made.

3 Experiments

We tested the deep graphical model on two publicly available corpora of spelling errors: a corpus of spelling errors in Tweets[3], and a Wikipedia corpus of edit errors[4]. The Tweet corpus consists of 39,172 spelling errors, their correction, and the edit steps to be performed to transform the error to the correct word form (replacement, insertion, deletion). The corpus has 2,466 different correct word types, which implies on average 15.9 errors per word. The Wikipeda corpus consists of 2,455 errors coupled to correct word forms (a total of 1,922 different correct word types), which means on average 1.3 errors per word. We performed two types of experiments. The first experiment is a small lexicon experiment, in which we built for separate portions of errors a model of the corresponding ground truth corrections. For both corpora, we took 10 equal-sized test data splits consisting of spelling errors, and the corresponding ground truth, built models for the ground truth words, and tested the models on the 10 test splits. We indexed every word for the presence of character bigrams and trigrams, on the basis of a feature lexicon of character n-grams derived from the training data. This lexicon consists of the 1,000 top character n-grams ranked by inverse document frequency, treating the bag of character n-grams of a word as a 'document'. Next, in order to assess the impact of lexicon size on correction accuracy, we built a model of a larger amount of ground truth data, and tested the model on the same test data splits we used in the first experiment. We used the first 5,000 words of the Twitter corpus, and the first 500 words of the Wikipedia corpus for training, after which we tested the model on the first 5 test splits of the corresponding test data. For both corpora, this means a considerable increase in lexicon size. To speed up training of the model, we limited the feature lexicon to the 500 top-ranking character n-grams. Details of the corpus size, and the training and test splits are listed in Table 1.

Corpus	Size	Size of training splits	Size of test splits	Splits used	Lexicon size
Experiment 1					
Twitter	39,172	1,000	1,000	1-10 (the first 10)	1,000
Wikipedia	2,455	100	100	1-10 (the first 10)	1,000
Experiment 2					
Twitter	39,172	5,000	1,000	1-5 (the first 5)	500
Wikipedia	2,455	500	100	1-5 (the first 5)	500

Table 1: Characteristics of the data used in the experiments.

We report correction accuracy for different minimal word lengths, varying from 4, 6, 8 to 10 characters. Since we use character n-grams as features, we hypothesize that our approach works better for relatively longer words, which produce larger bags of character n-grams. The deep graphical model was implemented in Python, with the Oger toolbox[5]. As a baseline, we used the well-known Unix `aspell` spelling correction program. While this comparison is not entirely sound (for one thing, we did not feed `aspell` with the same lexicon data as the RBM), it shows the baseline performance on our data with a standard, widely used and off-the-shelf large-vocabulary tool.

4 Results and discussion

Results are presented in Figure 4 and are summarized in Table 2. The results demonstrate effective error correction capability for words longer than 6 characters, and show an overall performance well above baseline,

[3]http://luululu.com/tweet/, published by Eiji Aramaki.
[4]http://www.dcs.bbk.ac.uk/ ROGER/corpora.html
[5]http://reservoir-computing.org/organic/engine

with the exception of the extended lexicon for Wikipedia. Yet, it is clear that, in general, correction accuracy linearly decreases with the size of the lexicon. The effect is stronger for the Wikipedia corpus than for the Twitter corpus. In fact, for the latter, the use of the large lexicon becomes an advantage for long words of 9 characters and more. Especially for shorter words in both corpora, the increase in lexicon size seems to lead to less accurate models. While we cannot rule out effects of non-optimal hyperparameters, we propose to

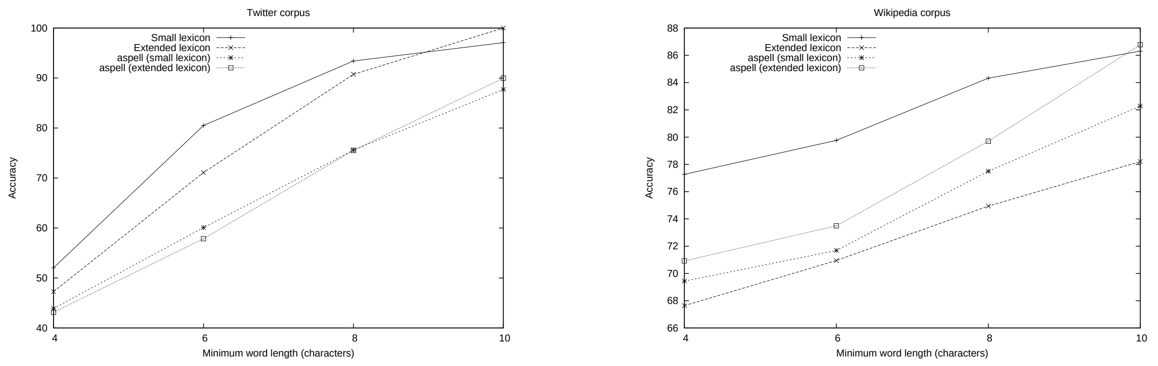

Figure 4: Accuracy results for the Twitter and Wikipedia corpora, for the small and extended lexicons. The `aspell` program was used as a baseline on the test data for both experiments; it uses its own dictionary.

Experiment 1 - small lexicons				
	≥ 4	≥ 6	≥ 8	≥ 10
Twitter	52	80.5	93.4	97.1
Baseline	43.9	60.1	75.6	87.7
Wikipedia	77.3	79.8	84.3	86.3
Baseline	69.4	71.7	77.5	82.3

Experiment 2 - extended lexicons				
	≥ 4	≥ 6	≥ 8	≥ 10
Twitter	47.3	71.1	90.7	100
Baseline	43.1	57.9	75.5	89.9
Wikipedia	67.6	70.9	74.9	78.2
Baseline	70.9	73.5	79.7	86.8

Table 2: Accuracy results for the Twitter and Wikipedia corpus, under both experimental conditions, and for different minimum word lengths (4-10).

remedy this problem by using an array of models derived from relatively small sublexicons. Input strings can efficiently be matched against these sublexicons, under the hypothesis that out-of-vocabulary words receive bit strings with higher average Hamming distance to the bit strings of the training words in the lexicon than in-vocabulary words. This is an expression of the reconstruction error of the model for 'foreign' data. The less optimal the fit of the model to the test data, the higher the Hamming distance between the predicted bit strings with the learned bit strings for the training data becomes. The proposed procedure basically performs a nearest neighbor search over several lexicons in parallel, and the dictionary with the lowest average Hamming distance between training bit strings and predicted bit strings produces the correction.[6] Formally,

[6]The size of the separate sublexicons is an extra parameter that can be further tuned during training.

we compute $C(x)$, the correction of input word x as follows:

$$C(x) = \arg\min_{y \in L} \left(\arg\min_{L} H(\hat{p}_L(x), \hat{p}_L(y)) \right) \tag{1}$$

Here, $H(\hat{p}_L(x), \hat{p}_L(y))$ is the Hamming distance between the bit strings predicted by the model for lexicon L, for words x and y (where y is a correct word in the dictionary L). We tested the viability of this procedure on the Wikipedia corpus for the 10 splits of Experiment 1, and we were able to infer for all test splits the correct lexicon for lookup. For every test split, we produced predictions with the 10 training data models, only one of which contained the correct ground truth (training partition i for test partition i). For every word in the test data, we let these models produce a bit string. We then found the nearest neighbor bit string in the corresponding training bit strings, and measured the Hamming distance between the predicted bit string and this nearest neighbor. For every test split, we averaged the distances. The resulting 10×10 distance matrix is presented in Figure 5, which indeed shows (on the diagonal) minimal distance between the test data and the correct models. This means that the model with the minimal average Hamming distance between predictions and learned bit strings can be identified as the preferred model for correction. From a

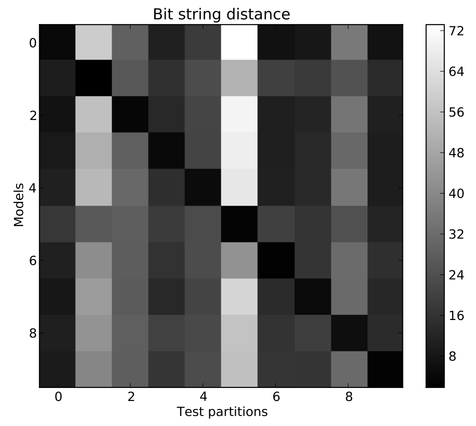

Figure 5: Average Hamming distance between bit strings produced for the 10 test data splits (starting at 0), and the bit strings learned for the training data.

practical point of view, these relatively small models can be jointly loaded in memory, and spelling error correction can be done efficiently in parallel.

5 Related work

For a traditional topic like spelling correction, many solutions have been proposed, ranging from either knowledge-intensive (rule-based, dictionary-based) to statistical, data-driven and machine learning based methods, and mixed rule-based/data-driven solutions ([16]). In [12], an overview of noisy channel methods that can be used for spelling correction based on character-based language models is provided. Models based on transformations attempt to apply inverse string edit operations in order to map noisy strings back to clean versions[7]. A modern substring matching algorithm that uses a similar approach is [22]. The well-known GNU `aspell` program is based on these transformations as well, in combination with a procedure mapping words to soundalike strings ('metaphones') that abstract away to a certain extent from orthography. A context-sensitive approach to spelling correction was presented in [6], using the Winnow machine learning algorithm. In [2, 14], early neural network approaches to spelling error correction were presented. In [24], a

[7]See http://norvig.com/spell-correct.html for a good and practical explanation.

hashing approach to the spelling correction of person names is presented. The authors formulate the problem of finding optimal bit strings as an Eigenvalue problem, to be solved analytically, and, using character bigrams, are able to improve significantly on a number of baselines. Their approach differs from ours in that we explicitly use machine learning techniques to learn optimal bit strings with self-optimization, and that we exploit an inherent characteristic of the learning paradigm (the reconstruction error of the trained autoencoder) to optimize lexicon selection for error correction. Character-level spelling correction based on error correction codes in the context of a brain-computer interface is described in [7].

6 Conclusions

We have demonstrated the effectiveness of deep autoencoders for spelling correction on two publicly available corpora of spelling errors in isolated words. To the best of our knowledge, this is a novel application of the semantic hashing technique proposed by [20]. Overall, character n-gram representations of strings produced high accuracy results, especially for longer words. This representation generates 'quasi-documents' from words, and makes semantic hashing techniques originally meant for documents amenable to the level of words. Increased lexicon size was found to be of negative influence on correction accuracy. We proposed the use of small sublexicons as a remedy, relying on the reconstruction error of the autoencoder to identify the suitable lexicons for correction of a certain input. In a separate experiment, we were able to identify for the 10 Wikipedia test data splits of our small lexicon experiment the sublexicon that minimized the reconstruction error of the test data. Our approach is knowledge-free, and only demands a list of correctly spelled words. It can be extended to contextual spelling correction, by using windowed character n-gram representations of text, which we plan for future work. While we are not aware of comparable results on the two corpora we used, we hope our research contributes to further benchmarking of spelling correction methods applied to these datasets. The data splits we used are easily reconstructed from our specifications. Future work wil also address scalable hyperparameter optimization. Due to the prohibitive time complexity of training the models, efficient methods are needed here. While it is in theory possible we hit a coincidental local maximum in our choice of parameter settings, we observed in line with [20] some indifference of the models with respect to perturbations of certain parameters. A systematic hyperparameter search will deepen our understanding of this phenomenon, and may lead to further improvement of our results.

References

[1] Yoshua Bengio. Learning deep architectures for AI. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009.

[2] V. Cherkassky and N. Vassilas. Backpropagation networks for spelling correction. *Neural Net.*, 1:166–173, 1989.

[3] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, March 1964.

[4] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, Inc, 2003.

[5] Huizhong Duan and Bo-June (Paul) Hsu. Online spelling correction for query completion. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 117–126, New York, NY, USA, 2011. ACM.

[6] Andrew R. Golding and Dan Roth. A Winnow-based approach to context-sensitive spelling correction. *Mach. Learn.*, 34(1-3):107–130, February 1999.

[7] N. Jeremy Hill, Jason Farquhar, Suzanna Martens, Felix Bießmann, and Bernhard Schölkopf. Effects of stimulus type and of error-correcting code design on BCI speller performance. In *NIPS*, pages 665–672, 2008.

[8] G. Hinton. A practical guide to training restricted Boltzmann machines. *Momentum*, 9:1, 2010.

[9] G.E. Hinton, S. Osindero, and Y.W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[10] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[11] Geoffrey E. Hinton. To recognize shapes first learn to generate images. In *Computational Neuroscience: Theoretical Insights into Brain Function*. Elsevier, 2007.

[12] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[13] M. Koppel and J. Schler. Exploiting stylistic idiosyncrasies for authorship attribution. In *Proceedings of IJCAI*, volume 3, pages 69–72, 2003.

[14] Mark Lewellen. Neural network recognition of spelling errors. In *COLING-ACL*, pages 1490–1492, 1998.

[15] Yanen Li, Huizhong Duan, and ChengXiang Zhai. A generalized hidden Markov model with discriminative training for query spelling correction. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '12, pages 611–620, New York, NY, USA, 2012. ACM.

[16] Lidia Mangu and Eric Brill. Automatic rule acquisition for spelling correction. In *In Proceedings of the 14th International Conference on Machine Learning*, pages 734–741. Morgan Kaufmann, 1997.

[17] Bruno Martins and Mrio J. Silva. Spelling correction for search engine queries. In *Proceedings of EsTAL-04, Espaa for Natural Language Processing*, 2004.

[18] Stephan Raaijmakers and Wessel Kraaij. A shallow approach to subjectivity classification. In Eytan Adar, Matthew Hurst, Tim Finin, Natalie S. Glance, Nicolas Nicolov, and Belle L. Tseng, editors, *ICWSM*. The AAAI Press, 2008.

[19] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23:2000, 2000.

[20] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, July 2009.

[21] Sabato Marco Siniscalchi, Dong Yu, Li Deng, and Chin-Hui Lee. Exploiting deep neural networks for detection-based speech recognition. *Neurocomput.*, 106:148–157, April 2013.

[22] Jason Soo. A non-learning approach to spelling correction in web queries. In *Proceedings of the 22nd international conference on World Wide Web companion*, WWW '13 Companion, pages 101–102, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[23] Efstathios Stamatatos. A survey of modern authorship attribution methods. *J. Am. Soc. Inf. Sci. Technol.*, 60(3):538–556, March 2009.

[24] Raghavendra Udupa and Shaishav Kumar. Hashing-based approaches to spelling correction of personal names. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1256–1265, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[25] W. John Wilbur, Won Kim, and Natalie Xie. Spelling correction in the PubMed search engine. *Inf. Retr.*, 9(5):543–564, 2006.