Project Data Science - Text Classification

Conference recommendation for research papers

Primary Topic: IENLP

Course: 2021-1B – Group: 65 – Submission Date: 2021-01-26

Femke Boelens
University of Twente
f.a.boelens@student.utwente.nl

Rutger Mauritz
University of Twente
r.r.mauritz@student.utwente.nl

ABSTRACT

In this research we develop classification models for predicting a conference given a title of a research paper. We consider naive Bayes, logistic regression, support vector machines and a neural network as models. The first three models make use of a bag-of-words representation and the neural network uses a word embedding and takes the relational structure of words into account via an LSTM core. In addition, we investigate several methods of preprocessing, including stopword removal, lemmatization and stemming. Via numerical analysis we show that the neural network with LSTM layers outperforms the other classification models. In terms of preprocessing, although the performance differences are not significant, a combination of stopword removal and lemmatization yields the best results.

KEYWORDS

NLP, text classification, machine learning, LSTM, word embedding, bag-of-words, deep learning, neural network

1 INTRODUCTION

When a researcher publishes his or her paper, he or she has to publish and present it at a specific conference. It may be hard to decide on which of the conferences to choose from. This problem can be seen as a multi-class text classification problem, where the title of the paper is the text and the class is the conference. This brings us to the following research question:

> *How well can we use text classification models to predict a conference for a given research title and how does text preprocessing influence these results?*

We investigate how we can address this question by constructing text classification models and compare them to find out which classification model yields the best results. We consider two different types of classification models. On the one hand, we look at models that exploit the so-called *bag-of-words* principle together with *'TFIDF-ing'*. We look at three of such classification models, namely Naive Bayes, Logistic Regression and Support Vector Machines. On the other hand we look at a classification model that uses a *word embedding* and takes the context of words into account. This classification model is a neural network with *Long Short-Term Memory* architecture. On top of this, we look at different types of preprocessing: stopword-removal, stemming and lemmatization and numerically verify which combinations lead to the best results.

The remainder of this paper is organized as follows. In Section 2 we explain theory considering the bag-of-words/TFIDF-ing and word embedding principles. What is more, we briefly explain the theoretical background of the classification models that we use. In Section 3 we describe how we apply the models to our specific task and how we can compare them. Our experimental setup and its results are included in Section 4. Furthermore, we discuss our results in Section 5 and conclude this paper in Section 6.

2 BACKGROUND

In this section, we briefly explain the theoretical backgrounds of the main building blocks of the classification models that we use.

2.1 Stemming and Lemmatization

In text classification it is desirable that words with different inflections should be treated equally. An example is that the occurrence of the plural form of a word in a text is equally important as the singular form. Stemmers and lemmatizers address this challenge. A stemmer removes suffices from words and a lemmatizer tries to map a word to its canonical form. That is, the form of a word that one would find in a dictionary. [14]

2.2 Bag-of-words

The input text that needs to be classified in a text classification problem is often represented as a vector $\mathbf{s}$, which is a sequence of words. One way of representing a text so that it is suitable for classification is to construct a vector $\mathbf{x}$ containing for each word how often it occurs in the text. Hence, element x_i of this vector is the count of word i in text $\mathbf{s}$. One should note that the length of this vector equals the number of unique words in the overall text corpus where $\mathbf{s}$ is a part of. The vector does not include any relations between or the order of the words in the text and thus also ignores grammar and the structure of the text. This vector $\mathbf{x}$ is thus an unordered set of word counts. Therefore, it is often called a *bag-of-words* (BOW). [13] [9]

2.3 TFIDF-ing

As an extension of the BOW representation of text, we could use 'TFIDF-ing' [6] that is a specific way of giving weights to each term (word) in the corpus. Here, TFIDF stands for *term frequency–inverse document frequency* and is a statistic that in a particular way reflects the importance of a word to a document (paper title) in the text corpus. The TFIDF statistic downweights the importance of a word proportionally to the number of documents it occurs in. The idea behind this is that if a word appears in many documents, it is probably less important for the classifier as it does not contain much discriminatory information. The relation between the BOW and TFIDF approach is as follows. The TFIDF statistic for term t in document d is given by

$$\text{tf-idf}(t,d) = \text{tf}(t,d) \cdot \text{idf}(t),$$

where $\text{tf}(t,d)$ measures the term frequency of term t in document d (just as what BOW does) and $\text{idf}(t)$ is the inverse document frequency, which is inversely proportional to the document frequency $\text{df}(t)$, that measures the number of documents containing term t. By weighing the terms, TFIDF-ing has an obvious advantage over the regular BOW approach in the sense that it pays more attention to the value of words in terms of how discriminatory they are.

2.4 Word embedding

As mentioned above, the BOW approach does not take into account the relation that a word has to the sentence it belongs to. This is the main weakness of BOW, as the context of a word is often very important. An approach that does take the context into account, is the approach that uses *word embedding*. Word embedding is a technique in Natural Language Processing that maps words/phrases from the vocabulary to a vector of real numbers. These vectors are calculated based on the probability distribution for each word to appear before or after another. In other words, the vectors representing words that often appear in the same context in the corpus will lie close together in the vector space. One way of creating a word embedding is by using the *Word2Vec* [15] approach from Google. Word2Vec either uses the *continuous back of words* (CBOW) or *skip-gram* architecture. To put it simply, CBOW tries to predict a word for a given context, whereas the skip-gram approach tries to predict the context given the word. Both these architectures consist of a two-layer neural network that are trained to reconstruct contexts of words. The input to such a neural network is a text corpus whereas the output is a vector space such that each word in the text corpus is mapped to a unique vector in that space. This word embedding can then be used to map words sequences to a sequence of embedded vectors, that can then be fed to e.g. a neural network model, see Section 2.8.

2.5 Naive Bayes classifier

The *Naive Bayes* (NB)[13] classifier is based on the idea of Bayasian Inference. The classifier is a probabilistic classifier in the sense that the predicted class $\hat{y}$ is the class among all classes for which the posterior probability of the class given the text is the highest. To find the posterior probabilities $P(y = k|\mathbf{x})$ for all classes $k \in \{1, 2, \dots, K\}$, Bayes' Theorem is used. This theorem also includes prior probabilities and the likelihood of the data in a class. It is stated as follows.

$$P(y = k|\mathbf{x}) = \frac{P(\mathbf{x}|y = k)P(y = k)}{P(\mathbf{x})}.$$

Since the NB classifier uses the likelihood $P(\mathbf{x}|y = k)$ of the data lying the classes in its calculations, it is called a generative model. The classifier is naive because of the naive Bayes assumption that the features/attributes that represent each text are independent of each other. Furthermore, the NB classifier is a linear classifier.

2.6 Logistic Regression classifier

The *Logistic Regression* (LR) [2] classifier learns a set of weight vectors $\{\mathbf{w}_1, \dots, \mathbf{w}_K\}$ and a biases $\{b_1, \dots, b_k\}$ for each class $k \in \{1, 2, \dots, K\}$ in order to find the evidence $z = \mathbf{w}_k\mathbf{x} + b_k$ for each class k, where $\mathbf{x}$ is a feature vector representing a text. In order to map $\mathbf{x}$ to a posterior probability $P(y = k|\mathbf{x})$, the softmax function [3] is used:

$$P(y = k|\mathbf{x}) = \frac{e^{\mathbf{w}_k \cdot \mathbf{x}}}{\sum_{j=1}^{K} e^{\mathbf{w}_j \cdot \mathbf{x}}}.$$

LR assigns label $\hat{y}$ to vector $\mathbf{x}$ when its corresponding posterior probability $P(y = \hat{y}|\mathbf{x})$ is maximal. The LR classifier is a discriminative model, as it directly learns the probability of a class given a text and thus learns which features of texts are the best to discriminate between different classes. [13]

2.7 Support Vector Machine classifier

The *Support Vector Machine* (SVM) [5] tries to find a decision boundary in the input space such that the margin between training examples from different classes is maximized. The training examples that lie on this boundary are called the support vectors. What is more, a *soft-margin* SVM allows for some training data points lying on the wrong side of this boundary. The function for the decision boundary is of the form

$$D(\mathbf{x}) = \mathbf{w}\mathbf{x} + b,$$

where $\mathbf{x}$ is the feature vector representation of the input text. The SVM learns the weights $\mathbf{w}$ and the bias b via the following optimization problem over the training data $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$:

$$\begin{aligned} &\min_{\mathbf{w},b,\mathbf{u}} \frac{1}{2}||\mathbf{w}||^2 + \gamma \mathbb{1}^T\mathbf{u} \\ &\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - u_i \text{ for } i = 1, \dots, N \\ &\quad \mathbf{u} \geq \mathbf{0} \end{aligned}$$

Here $\mathbf{u}$ represents the slackness variables for the soft-margin setting and γ is a hyper-parameter of the method. After optimization, the sign of $D(\mathbf{x})$ indicates the predicted class of $\mathbf{x}$.

2.8 LSTM classifier

As noted earlier, the context of a words matters for its meaning in a sentence. In other words, a sentence can be seen as a sequence where the position of a word in that sequence matters. A deep learning model that can work with sequences is the so-called *Recurrent Neural Network* (RNN) [16]. An RNN consists of certain states that contain information from the past sequence and the new inputted data at that time. An example of such a RNN architecture is given in Figure 1. The architecture is built such that it can predict the future from the past sequence. However, it is also possible to at the same time use the future to predict the past. This is called a Bi-directional RNN. Note that in Figure 1, at each time t the external signal x_t is a word from the sentence. In other words, the number of states in the RNN equals the number of words in the sentence (which can be of varying length). Because of the problem of vanishing or exploding gradients in such an RNN, a modification called *Long Short-term Memory* (LSTM) [12] is proposed that solves this problem. Because of its ability to capture past and future information from a sequence and thus capturing context in a sentence, an LSTM is an ideal model for text classification.

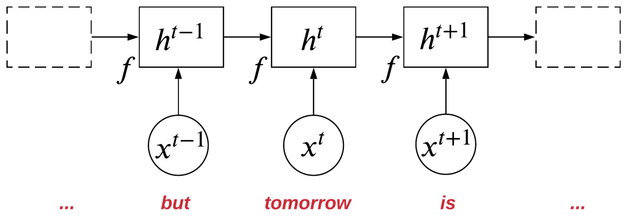

Figure 1: LSTM architecture example

3 APPROACH

In this section we describe how can use the above theory to construct classification models that are able to predict a conference for a given title of a research paper. As mentioned earlier, we analyze an NB, LR and SVM model that use the BOW text representation together with the TFIDF transformation as an input. In addition, we use an LSTM model that uses a word embedding text representation as input. We also look at different ways of preprocessing the text before applying the classification models. We close this section by explaining how we can validate and compare the performances of the different models.

3.1 Preprocessing

We apply different methods of preprocessing to the text before feeding it to the classifier. For all text, we transform the text to lowercase characters. Besides this step, we also look at several methods of preprocessing, being stopword removal, lemmatization [1] and stemming [4]. For each different classifier model, we look at its performance for all reasonable combinations of preprocessing.

3.2 Training and applying classification models

The training process for each of classifier models can broadly be summarized as follows. Given training data $\mathcal{T} = \{(\mathbf{s}_i, y_i)\}_{i=1}^N$ consisting of paper titles and corresponding conferences where they were published, we feed the paper titles (after certain preprocessing, see Section 3.2.1 and 3.2.2) to the classifier model h. Then for each paper title $\mathbf{s}_i$, h predicts a label $h(\mathbf{s}_i) = \hat{y}_i$. Predicted label $\hat{y}_i$ is then compared with the ground truth label y_i. Via minimization of the classifier-specific loss function $L(\hat{y}, y)$, the parameters of the classifier are updated to obtain better performance on the training set. Once the classifier model h has been trained, we apply it to the unseen test data so that we can determine the test performance.

Besides this general scheme, each of the two classifier classes requires some extra steps and details. Those will be explained in Section 3.2.1 and 3.2.2 below.

3.2.1 BOW: NB, LR & SVM. As mentioned earlier, for both the NB, LR and SVM models we use the BOW representation of the text as input to these models. On top of that, we also use TFIDF-ing as term-weighing scheme. Given the training titles $\{\mathbf{s}_i\}_{i=1}^N$ we first learn the vocabulary of this text corpus. This then allows us to convert each title $\mathbf{s}_i$ to a BOW representation $\mathbf{x}_i$ after which we apply TFIDF-ing to obtain $\tilde{\mathbf{x}}_i$, containing the TFIDF features for $\mathbf{s}_i$. These feature vectors are then passed to the multi-class classifier h (either NB, LR or SVM) that calculates an output $h(\tilde{\mathbf{x}}_i) = \hat{y}_i$ and compares it to the ground truth label y_i. When applying the trained model h to the test set, one should note that the same learned vocabulary and TFIDF-scheme is used to transform the test examples (which is thus the vocabulary and TFIDF-scheme of the training set).

3.2.2 Word embedding: LSTM. The LSTM model consists of several parts that we explain briefly in terms of their place in the pipeline.

First, the model learns a vocabulary based on the training data $\mathcal{T}$. This dictionary is then used to convert every paper title $\mathbf{x}_i$ in both the training and the test data to a sequence $\tilde{\mathbf{x}}_i$ of integers, where each integer represents a word. Each of these sequences is padded at the end with zeros so that all sequences have a fixed length d. An example of this process is given in Figure 2.

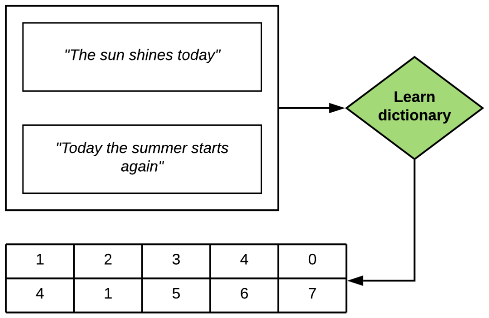

Figure 2: Text to sequence with $d = 5$. Here the vocabulary is learned such that the first word in the corpus is assigned a 1, the second word a 2, etc.

Once we have the padded sequences, we use the sequences from the training data to learn an embedding via the Word2Vec model from Google [15]. For this we use the skip-gram approach that learns an embedding via trying to predict a context given a word. For a specified embedding size d_e (the length of each embedding vector), each word k in the learned dictionary is assigned a vector $\mathbf{e}_k \in \mathbb{R}^{d_e}$. This then results in an embedding matrix E that has the embedding vectors on the rows such that row E_{k*} corresponds to the embedding $\mathbf{e}_k$ of the k-th word from the dictionary.

For training, the sequences $\{\tilde{\mathbf{x}}_i\}_{i=1}^N$ are then fed to a neural network. The network optimizes over this training data by optimizing its parameters via backpropagation of some error function L. The architecture of this network is as follows:

- The first layer is the so called embedding layer. What this layer does is that for a given sequence $\tilde{\mathbf{x}}_i$ of length d, the integers in this sequence are used to access the rows of the embedding matrix E so that a sequence of d embedding vectors $\mathbf{e}_j$ is returned. This is depicted in Figure 3.
- This sequence of embedding vectors is then passed on to a bi-directional LSTM, followed by a dense neural network. The last layer is a Softmax layer [3] such that the output has a dimensionality equal to the number of categories in the training data with each element being between 0 and 1 and the sum of the elements being equal to 1. The output thus represents a probability distribution over the categories

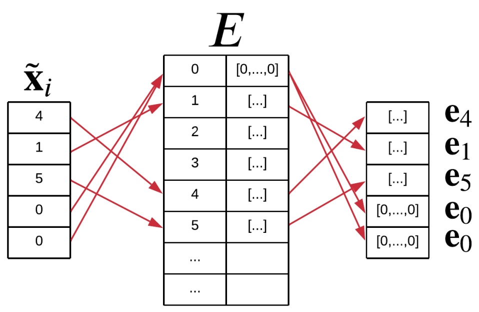

Figure 3: Embedding layer

and for a given input $\tilde{\mathbf{x}}_i$, the output $\hat{y}_i$ is chosen to be the category that is assigned the most probability mass.

3.3 Comparing classification models

In order to compare the classification models together with different combinations of preprocessing methods that we analyze, we perform k-fold cross validation using the training data set. As a metric to measure the performance of the classification, we use the micro-average precision, recall and F1 measure. Because we are dealing with a multi-class problem, these measures turn out to be the same. We provide an explanation why this is the case; In case of having 5 classes, the formulas for micro-average precision and recall are

$$\text{Precision} = \frac{TP_1 + TP_2 + TP_3 + TP_4 + TP_5}{TP_1 + TP_2 + TP_3 + TP_4 + TP_5 + FP_1 + FP_2 + FP_3 + FP_4 + FP_5},$$

$$\text{Recall} = \frac{TP_1 + TP_2 + TP_3 + TP_4 + TP_5}{TP_1 + TP_2 + TP_3 + TP_4 + TP_5 + FN_1 + FN_2 + FN_3 + FN_4 + FN_5}.$$

The confusion matrix of our classification model has elements C_{ij}, where j is the predicted class and i is the true class. For this confusion matrix, the micro-average precision and recall are given by

$$\text{Precision} = \frac{\sum_{i=1}^{5} C_{ii}}{\sum_{i=1}^{5} \sum_{j=1}^{5} C_{ij}} = \text{Recall}.$$

Hence, if taken the micro-average of precision and recall, the resulting metrics are equal. Moreover, the F1 measure is calculated as

$$\text{F1 measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

If precision and recall are equal, say they equal A, we obtain

$$\text{F1 measure} = \frac{2A^2}{2A} = A.$$

Hence, when taking the micro-average, precision, recall and F1 measure are equal. [11]

4 EXPERIMENTS

In this section we present the results of our experiments and mention the setup that we used. For the code implementations mentioned in this section, we refer to the GitHub repository [1].

[1]https://github.com/RRMauritz/TCNER

4.1 Data Set

The data set considered in this project is Conference Proceedings data and contains titles of research papers[17]. For each research paper, the conference at which the paper is presented is given and thus can be seen as a label. The conferences that are represented in this data set are INFOCOM, ISCAS, SIGGRAPH, VLDB and WWW. The data set contains 21,643 training instances and 3,373 test instances.

4.2 Setup

In this subsection we provide necessary details on how our approach is implemented.

For the preprocessing phase, we use the NLTK Python package[7]. For the stopword removal, we use the English language stopwords and for lemmatization and stemming we use the built-in functions of this package.

For the BOW representation we use the TFIDF vectorizer of the sklearn library[8], which is equivalent to applying a count vectorizer and then perform a TFIDF transform. We set the maximum number of features to 10,000 and extract both unigrams and bigrams.

For the classifiers that use the BOW representation we also use the sklearn library. For the NB classifier we use the built-in multinomial naive bayes function. Furthermore, for the LR classifier we also use a built-in function, together with the LBFGS solver. Finally, for the SVM classifier we use the SGD classifier, together with a hinge loss function and L2 regularization.

For the word embedding procedure, we use the tokenizer of NLTK for fitting a vocabulary on the training data. The padding is done such that each sequence is of length equal to the length of the largest title in the training data set. In order to create the word embeddings, we use Google's Word2Vec model [15] with an embedding size of 300 and a maximum distance between the current and predicted word of 7. As training algorithm, we use skip-gram as this leads in general to more accurate results for infrequent words [10] and we perform 30 iterations. The architecture and parameters of the neural network that we use for the classification are given in Table 1.

Table 1: Architecture and parameters of the neural network.

Network layers	
Embedding layer	
Bidirectional LSTM layer	dropout = 0.4
Bidirectional LSTM layer	dropout = 0.4
Dense layer + ReLU	output: 64 units, L2 regularization
Dense layer + Softmax	output: 5 units
Parameters	
Optimization Method	ADAM
Batch Size	64
Epochs	30
Loss Criterion	Sparse Categorical Cross Entropy

In order to compare the models together with the different combinations of preprocessing methods, we perform k-fold cross validation with k equal to 10 on the training set. We use the sklearn library for this. Finally, we apply the model and (combination of) preprocessing that results in the highest micro-average precision/recall/F1 measure on the test set and report again its micro-average precision/recall/F1 measure.

4.3 Results

As explained in Section 3.3, we first perform 10-fold cross validation on the training set in order to approximate the performance of the classification models, together with different combinations of preprocessing. In Table 2 we report the micro-average precision/recall/F1 measure.

Table 2: 10-fold CV performance estimate (micro-average precision/recall/F1) of the classification models for different combinations of preprocessing.

Preprocessing	NB	LR	SVM	LSTM
None	0.840	0.832	0.790	0.863
Stopword	0.849	0.834	0.798	0.861
Stopword & Lemmatization	0.853	0.835	0.801	0.866
Stopword & Stemming	0.849	0.837	0.801	0.860
Lemmatization	0.844	0.836	0.796	0.866
Stemming	0.843	0.834	0.796	0.865

The highest scores are achieved by the LSTM with either stopword removal and lemmatization or only lemmatization as preprocessing. We therefore apply the LSTM with these preprocessings on the test set. The results can be found in Table 3.

Table 3: Test set performance (micro-average precision/recall/F1) of the LSTM model for two combinations of preprocessing.

Preprocessing	LSTM
Stopword & Lemmatization	0.822
Lemmatization	0.807

5 DISCUSSION

The results from Table 2 tell us several things. First of all we notice that for each combination of preprocessing, the LSTM model with word embedding outperforms the other classifiers, be it not very significantly. This tells us that it is indeed of some added value to take into account the relational structure of words within the sentences. What is more, we notice that the performance of the NB model - despite its simplicity and strong assumptions - is quite close to that of the LSTM model. It thus seems that using the relational structure of words does not lead to a very significant increase of the model performance. The reason for this can be explained by the nature of the text, namely the fact that we deal with titles of papers in our research. When a researcher creates a title for his/her research paper, he/she will try to put as many information as possible into the title of the paper. Hence, the title is often more like a set of different words (nouns) rather than a sentence with many inter-sentence relationships, which may explain why models that use bag-of-words representation or a word embedding can have similar performances.

Another important observation is that if we compare the CV performance estimates for the LSTM model from Table 2 with that of the test set performance given in Table 3, we see that the test set performance is significantly lower. This is an indication for overfitting which should be addressed by applying even more/better regularization techniques than the one that we applied (i.e. dropout & L2 regularization). This brings us to another point of discussion. For all models, we did not investigate different values of all hyper-parameters extensively as this was not within the scope of our research. Doing so may improve the generalization performance of the models.

The last observation is that it seems that the different methods of preprocessing do not lead to significantly different results. A possible explanation is again the nature of the texts. In titles of research papers words often occur in the same form and hence, preprocessing does not change a lot in these sentences. This is in especially true for the most important words of a research, which are often placed as nouns in the title.

6 CONCLUSION

The main goal of this paper was to find a text classification model that can recommend a conference to a researcher, given the title of his/her new article. We did this by comparing different text classification models, namely on the one hand the naive Bayes, logistic and support vector machines classifiers that make use of the so-called bag-of-words text representation with TFIDF-ing, and on the other hand a neural network classifier with LSTM layers that uses so-called word embedding as input. In addition, we investigated different methods of preprocessing, including stopword removal, stemming and lemmatization. For the comparison we used the micro-average precision, recall and F1 measure as performance indicators and we performed this comparison using k-fold cross validation on the training set.

Based on the results, we conclude that the neural network using LSTM layers together with lemmatization and stopword removal or only lemmatization yields the best performance. However, we also saw that the performance differences were not very significant, especially in terms of preprocessing methods. In addition, the performance results of the NB classifier and the LSTM classifier are close.

As a suggestion for further research, we recommend to investigate hyper-parameter tuning for the different models. Furthermore, more thorough quantitative and qualitative investigation can be done on the (lacking) effect of preprocessing methods on the performance.

REFERENCES

[1] [n.d.]. Lemmatisation. https://en.wikipedia.org/wiki/Lemmatisation. Accessed: 2021-01-25.
[2] [n.d.]. Logistic regression. https://en.wikipedia.org/wiki/Logistic_regression. Accessed: 2021-01-25.
[3] [n.d.]. Softmax function. https://en.wikipedia.org/wiki/Softmax_function. Accessed: 2021-01-25.
[4] [n.d.]. Stemming. https://en.wikipedia.org/wiki/Stemming. Accessed: 2021-01-25.
[5] [n.d.]. Support-vector machine. https://en.wikipedia.org/wiki/Support-vector_machine. Accessed: 2021-01-25.
[6] [n.d.]. tf-idf. https://en.wikipedia.org/wiki/Tf-idf. Accessed: 2021-01-25.

[7] Steven Bird, Edward Loper, and Ewan Klein. 2009. *Natural Language Processing with Python.* O'Reilly Media Inc.

[8] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning.* 108–122.

[9] Jacob Eisenstein. 2018. Natural language processing.

[10] Google. 2013. *Word2Vec.* https://code.google.com/archive/p/word2vec/

[11] Margherita Grandini, Enrico Bagli, and Giorgio Visani. 2020. Metrics for Multi-Class Classification: an Overview. (08 2020).

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[13] Daniel Jurafsky and James Martin. 2008. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Vol. 2.

[14] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval.* Cambridge University Press.

[15] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[16] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533–536.

[17] Fabrizio Sebastiani. 2002. Machine Learning in Automated Text Categorization. *ACM Comput. Surv.* 34, 1 (March 2002), 1–47. https://doi.org/10.1145/505282.505283