Sentiment Analysis - Hotel Reviews

Johannes Benedikt Wichtlhuber[1520561], Marius Anton Hessenthaler[1563908], Sebastian Müller[1715417], Samuel Knaus[1715360], Andrej Katic[1715622], and Lennard Fabinski[1715337]

Data Mining I – Team 5

Abstract. We ran a sentiment analysis as a classification problem on the basis of 515 thousand hotel reviews. In the course of this project we combine different feature generating steps with a selection of machine learning models to find a suitable solution for the problem. We found that the fastText classifier achieves the best f1-Score of **0.62**.

1 Goals and Introduction

1.1 Motivation

The Warriott hotel chain has almost 7,000 luxury hotels around the world. During the COVID-19 crisis the Customer Service & Analytics Department sent a survey to their guests from the year before. This survey has more than 500,000 unstructured and textual replies which all need to be read by the Customer Service team. During the same time the Chairman Will Warriott becomes very worried about the future of his hotel chain and instructs Team 5 to give him a quick response about the customers reviews. From our Data Mining lecture Team 5 knows that a sentiment analysis is useful for deciding whether a statement is positive, neutral or negative. Therefore, we aim to train a classifier that predicts the sentiments of the reviews.

1.2 Structural Aspects of the Data

To train the sentiment analysis classifier the data set "515K Hotel Reviews Data in Europe" from Kaggle is used. It includes more than 515.000 review samples with textual reviews, numeric reviewer scores and many additional information like reviewer nationalities, dates, and geographical locations of 1493 luxury hotels from all over Europe [13]. From the data set we will only use the textual reviews (Positive Review and Negative Review) and the final score. By concatenating the Positive and Negative Review column we create our raw data. The reviewer score is the basis for the label.

Since we do not want to predict scores but the sentiment behind the textual data we bin the score in three different categories. Reviewers are giving relatively high scores for bad experiences at their stay. We see that the average score is 8.4, the 25% median 7.5, the 50% median 8.8 and the 75% median 9.6. The lowest score is only 2.5. We assume that only the scores of 8.5 to 10.0 are good

and have mostly nothing to complain about. Scores of 7.0 to 8.5 had some bad experiences but also liked some facts about the hotel. Scores below 5.0 had bad experiences and did not like anything or only very few things. We suspect that the unbalanced distribution in the reviewer score is only partially explained by the unbalanced data but also the fact that the reviewer score is not always proportional to the sentiment of the text. The shift of the review score ranges may be also accounted to the fact that the data set contains reviews of luxury hotels for which customers are especially critical. We therefore plan to bin the score into the labels [8.5 – 10.0] = good, [7.0 – 8.4] = ok, [6.9 – 0] = bad. Since the sentiment or class is defined by the reviewer score and not annotated we suspect that there might be some outliers in the data set which we could not filter due to the large size. We give some insights on outliers in the data set in our error analysis (see section 4.1).

The combined review text is on average 34 words long. The average word count of the negative part of reviews (negative length) with 18.5 is almost equal to the average word count of the positive part of the reviews (positive length) with 17.8. As expected the negative length decreases and the positive length increases with increasing reviewer score. However, with reviewer scores in the range of three to four the negative length is 47.4 and the positive length 8.2. Reviews with a score of 10 have a negative length of 12.4 and a positive length of 21.3 on average. We see that there is a large discrepancy in the word count of very positive and very negative reviews. The most interesting frequent words are: room (278521), staff (186561), hotel (180484), location (146094), breakfast (106325), good (104886), helpful (77305), friendly (76985), great (74064), bed (67492), clean (65341), nice (62584).

2 Preprocessing and Feature Generation

Preprocessing involves the application of standard methods for text mining and the binning based on the Reviewer Score that was described above. As data input the raw data set from Kaggle is used which contains text reviews divided into positive and negative reviews. We merged both columns into one and simply refer to this as "Reviews" in the following sections. The following methods are available and have been used depending on the feature generation method.

- **Tokenization** was applied by default. Any existing punctuation has been removed by the publisher of the set.
- **Stopword removal** reduces the length of the reviews and improves the data quality by removing meaningless words.
- **Stemming and Lemmatization** replaces the word with its stem to improve models that match similar words and consequently reduce the vocabulary size by combining words with the same stem.
- **N-gram** combines tokens to n-grams.
- **Balancing** Distributes target classes equally among the training set. We *downsample* the "good" class to the size of the "ok" class and *upsample* the "bad" class to the size of the "ok" class.

2.1 BOW TF/TF-IDF

We use *Tokenization*, *Stopword removal* and *Stemming* as preprocessing steps for the Bag of Words feature sets. We then apply the TfidfVectorizer from scikit-learn to this preprocessed set. The outcome is one Term Frequency (TF) feature set and one Term Frequency – Inverse Document Frequency (TF-IDF) feature set. Without pruning the feature dimensions are too high to train most of the classifiers in a reasonable amount of time. We decided to remove each word that is represented less than 0.5 percent in our reviews which resulted in an acceptable vocabulary of 561 terms.

2.2 Doc2Vec

Word2vec and Doc2vec are subsymbolic methods provided by the gensim library. We use them as a feature generating embeddings in this project. Both are used to represent texts as vectors. According to Lau et al. [10] Doc2Vec performs better than word2vec especially when processing large amounts of textural data. Therefore, doc2vec is selected as the feature generation method. We consider three options to produce our document embeddings:

1. Train a model with the available corpus
2. Use a pre-trained model
3. Use a pre-trained model and continue training with our own data set

Lau et al. [10] recommend approach 3 i.e. to further train the pre-trained model. Due to the lack of example implementations and suitable literature with recommendations for training we decide against the recommendation and try options 1 and 2 instead each resulting in one feature set. For the hyperparameters we use the proposed values from Lau et al [10]. Prior testing on a reduced data set of 10.000 reviews did not yield large deviations caused by different settings. With regards to the small number of available pre-trained models we choose the model trained on Wikipedia articles even though the domain on which it was trained is not optimal [10]. To optimize the results a list of all words used in the reviews of the data set is created and compared to the list of the pre-trained model. The result is a list of words called Out-Of-Vocabulary that appear in the reviews but not in the Wikipedia articles. Most of the words in this list are misspelled or certain numbers like woulnd, helpfull, wasnt, 30am etc.

Due to the limited research on the very specific task of hotel reviews, the settings and preprocessing steps for our own-trained model are related to the steps used in the pre-trained variant which only includes tokenization.

2.3 fastText

We use pre-trained word embeddings from fastText as feature representations of the review data. fastText trained word representations on Wikipedia include subword representations and therefore handle out-of-vocabulary words better. [17]. As an interface for fastText we use the SImple SenTence EmbeddeR library

to download data of the pre-trained embeddings and to generate fastText embedding vectors from the respective reviews [19]. For computation and storage reasons we decide to represent a single review as one vector. We use Tokenization and Lemmatization as preprcessing steps before generating features. Lemmatization might be helpful since it maps vocabulary into its inflected form which helps the classifier to process it as a single item. We generate one dataset with and one without stopword removal because stopwords may influence sentiments but also confuse the embedding generation [15].

3 Classifiers

In this section we describe the use of different machine learning classifiers. Inside the respective texts we provide a general explanation, first results as well as hyperparameter optimization steps and the final results. Problems and restrictions are covered when necessary. Due to the fact that we did not only have to identify the best classification algorithm but also committed to find suitable feature generated sets for the task, we decide to check all combinations. An overview of the performance of each Algorithm can be found in tables 1, 2 and 3. The results of the Sentiment Lexicons are not included in this table since they require their very own data mining process.

We split the data into a train set (70%), a validation set (15%) and a final test set (15%). Due to our unbalanced class distribution, we choose the macro f1-score as our comparative figure. In the second step, we optimize the different approaches with cross-validation (cv). After tuning we calculate the final score with the test set.

3.1 Sentiment Lexicon

The summed weights of predefined words over the texts allow thresholds or classifiers to predict the sentiment of a text. The sentiment analysis is conducted using the Opinion Lexicon by Hu & Liu in combination with thresholds and the Vader Lexicon as a classification method with the Vader Sentiment Analyzer [6, 11].
The Opinion Lexicon by Hu & Liu reaches a macro f1-score of **0.46** and acts as the first baseline. After trying out different thresholds the following determines the highest f1-score: if the positive words in the review outweigh the negative ones by 1 the sentiment is declared as *good*, if the negative and positive words are balanced the sentiment is declared as *ok* and otherwise the sentiment is declared as *bad*. The Vader Lexicon reaches an f1-score of **0.38**. It is sensitive to both the polarity and the intensity of sentiments expressed in the reviews. Based on the share of positive, neutral and negative sentiments a compound score for the review is computed. The best determined threshold for the compound is: a compound score higher than 0.725 is declared as *good*, lower than 0.55 is declared as *bad* and for the rest it is declared as *ok*.

3.2 Naive Bayes

The scikit-learn library is used for Naive Bayes (NB). It offers different implementations which work based on the Bayes Theorem: *Gaussian NB*, *Multinomial NB*, *Complement NB* and *Bernoulli NB*. In the tests the Bernoulli implementation shows no difference between using TF and TF-IDF feature sets since it only supports binary term occurrences and non-zero values are interpreted as 1 [16].

We use the Gaussian NB implementation as the second baseline. To optimize the results and to find out which of the four implementations works best on the given classification problem we test out each of them on the BoW feature sets. Multinomial NB in combination with TF and a balanced set results in a f1-score of **0.57**. The balanced TF-IDF set achieves the same result on Multinomial NB.

Finally, the NB implementations are applied to two BoW feature sets with 53 words and two with 1550 words in order to find out how the size of dimension influences our results. The low dimensional set achieves a f1-score of **0.50**. It drops by 0.07 compared to the result above which indicates that words with important sentiment were cut off. The large dimension set in contrast achieves a f1-score of **0.58**. Because the score only increases by 0.01, we assume that the reference BoW set with 561 words makes a good trade-off between size and computational performance with the means available to us.

3.3 Logistic Regression

Logistic regression is an adapted machine learning design that exploits a logistic function to model a binary dependent variable. Since we want to classify more than two classes, a multi-nominal logistic regression is used. Thus a linear model is applied to predict probabilities for every class, which are used by a softmax function to present the most possible class. [5] Our best logistic regression model reaches 0.59 in f1-score which is an above average score. For optimization we determine a grid search with the following hyperparameters and a 3-fold grid search:

- penalty= l1,l2, *solver= liblinear*, saga, C= 0.01, 0.1, 1, 10, 100
- penalty= none, l2, *solver= newton-cg*, lbfgs, sag, C=0.01, 0.1, 1, 10, 100
- penalty= l1,l2, *solver= liblinear*, saga, C= 100, 150, 200, 300, 500
- penalty= none, l2, *solver= newton-cg*, lbfgs, sag, C= 100, 150, 200, 300, 500

The split in four different test settings is necessary because of the lack of computational power and the fact that some solvers do not work with certain penalty methods. The optimized parameters: 'C': 1, 'penalty': 'l2', 'solver': 'lbfgs' could not improve the default parameters with a score of **0.59**.

3.4 SVM: SVC & Linear SVC

Support vector machines (SVM) originated in the idea of searching hyperplanes in vector spaces in order to separate classes by the widest margin. SVMs are potentially better in dealing with high dimensional data like text embeddings

since they are able to identify relevant features and are less likely to overfit. [7,9]. The standard SVM implementation was designed to operate for binary classification tasks only but there are special architectures like One-versus-One(OvO) or One-versus-Rest(OvR) to bypass this problem [3]. We use two model implementations SVC and LinearSVC of the scikit-learn python library. SVC and LinearSVC are fundamentally different because LinearSVC allows to optimize only squared hinge loss instead of the actual hinge loss which SVC optimizes mandatorily. We choose the LinearSVC with the different optimization criterion because it allows to work faster on larger data sets compared to SVC which also influences the classification. We first evaluate which feature representations works best for default SVC and LinearSVC classifiers for each classifier once with balancing and once without. Based on the feature set selection we optimize parameters for both Doc2Vec and BOW TF for the SVC with reduced data sets in a 4-fold-cv parameter grid search. The parameter grid consisting of C [0.01, 0.1, 1, 10], kernel ["rbf", "poly"], degree [1,2,3,5], gamma [0.1, 1, 10], decision function shape ["ovr", "ovo"] was defined with the help of [12,14]. The optimzed model for Doc2Vec and BOW TFdoes not increase the default models' macro f1 scores. We then try to enhance our model with Linear SVCs which we can run on the full dataset. We increased the score with a sufficiently large parameter grid search, where we sought to optimize the most important parameters: tolerance(0.0001,0.0001,0.001,0.01), C(0.1,5,10) and compared two multi class architectures(cramer-singer and OvR). Our optimized models worked well with BoW and pre-traind Doc2Vec features, but failed with fastText.

3.5 fastText Text Classifier

The fastText text classifier is a standard implementation from facebook's fastText library [1] [8]. The classifier generates fastText document vectors by averaging n-gram/word vectors. Finally, the classifier conducts a multinominal logistic regression. After the automatic holdout tuning process the model achieves **0.62** in the f1-score making it the best score in this task. The most important parameters during our tuning are learning rate and the dimension of the respective fastText embeddings. Balancing the set results in a decrease of overall accuracy but an increase of the f1-score in the neutral reviews. Overall, balancing does not help to increase the macro f1-score [15].

3.6 Random Forest

The Random Forest classifier is a class of ensemble methods developed specifically for decision tree classifiers. This classifier combines predictions which are made by multiple decision trees where each tree is generated based on the values of an independent set of random vectors. [20] The following conclusions result from the application of the random forest classifier on the data set. The best result with the default settings is achieved with BOW TF-IDF and a balanced training dataset. To evaluate the result with optimized hyperparameters we implement a randomized search to identify relevant parameters. Afterwards the

final optimization is based on a grid search with a stratified 6-folds cv [5]. The following hyperparameters and values combinations are identified and chosen to be tested:

- n_estimators: 20, 40, 60; max_depth: 70, 100, 150; min_samples_split: 2,4,6
- n_estimators: 50, 100, 150; max_depth: 100, 150, 200; min_samples_split: 2,4
- n_estimators: 150, 200, 300; max_depth: 200, 300, 400; min_samples_split: 2,4

With the optimized hyperparameters, max_depth= 300, min_samples_split= 4, n_estimators=300 the Random Forest achieves a f1-score of 0.67 as a result of the cv from the grid search. This model was able to produce a final f1-score of **0.58** on the test set.

3.7 Ensemble Learning

A second variation of ensemble learning is implemented which in contrast to Random Forest combines different classification algorithms in order to exploit weaknesses and strength of a diverse set of models. In combination the classifiers may supplement each other and may correct errors made by single classifiers. [5] At first, the following classification algorithms are taken into the ensemble: Random Forest, Logistic Regression, K-nearest neighbor & Gaussian Naive Bayes.

The first selection of models scores 0.55 on the BoW feature sets which compared to the baseline and other unoptimized classifiers gives a hint that further optimization may be rewarding.

For optimization we use our prior knowledge of optimal parameters to feed our ensemble learning model with already enhanced models. We drop bad performing ML-methods to increase our results. We also add new models to the ensemble. Because LinearSVC does not provide prediction probabilities we used hard voting which performs usually a bit worse than soft voting according to Geron [5]. In the end the Ensemble reaches a f1-score of **0,59**.

3.8 Binary classification with threshold for neutral reviews

In most of our classifiers the main problem is the prediction of the neutral class, for example a "good"/ "bad" classification the LinearSVC model with the balanced full training data reaches a macro f1-score of 0.80. Based on these findings we train a binary "god"/ "bad" SVC classifier. For the prediction we evaluate the class probabilities and predict only positive or negative with a probability above a threshold and otherwise predict the neutral class. We use this prediction method for the best performing classifiers SVC, Random Forest, Logistic Regression and Ensemble each with upsampled train sets. For each classifier we use the default parameters since we suspect that the optimized parameter on the three class classification were not transferable to the binary classification. For each classifier we then optimize the probability threshold. For SVC, Logistic Regression and the Ensemble the binary classification does not increase the score. For Random Forest the score increases to **0.61**.

Table 1. Best f1-Scores of ML Algorithms with **doc2vec** feature sets.

	Pre Traind		Own Traind	
ML-Algo	Unbalanced	Balanced	Unbalanced	Balanced
Gaussian NB	0.40	0.41	0.24	0.32
SVC	0.55*	0.58	0.59	0.33
LinearSVC	0.53*	0.57*	0.24*	0.32*
RF	0.43	0.48	0.29	0.32
Logistic Regression	0.55	0.57	0.24	0.32
Ensemble	0.47	0.51	0.26	0.33

Table 2. Best f1-Scores of ML Algorithms with **BoW feature sets**.

	TF		TF-IDF	
ML-Algo	Unbalanced	Balanced	Unbalanced	Balanced
Gaussian NB	**0.52**	**0.52**	**0.52**	**0.52**
Multinomial NB	0.47	**0.57**	0.49	**0.57**
SVC	0.55*	**0.59***	0.55	**0.59**
LinearSVC	0.56*	0.57*	0.55*	0.57*
RF	0.53	0.55	0.52	**0.61***
Logistic Regression	0.58	**0.59**	0.57	**0.59***
Ensemble	0.55	**0.59***	0.52	0.55

Table 3. Best f1-Scores of ML Algorithms with fastText feature sets.

	With stopwords		Without stopwords	
ML-Algo	Unbalanced	Balanced	Unbalanced	Balanced
Gaussian NB	0.37	0.36	0.35	0.34
SVC	0.25	0.42	0.25	0.42
LinearSVC	0.32	0.42	0.33	0.42
RF	0.40	0.43	0.40	0.44
FastText Classifier	**0.62**	**0.62**	0.59	0.48
Logistic Regression	0.33	0.44	0.33	0.44
Ensemble	0.42	0.40	0.42	0.39

(*) Achieved with optimized algorithms. Best results are **bold**.

4 Discussion and Conclusion

4.1 Error Analysis

We conduct an error analysis on falsely classified instances on one of our best classifiers Random Forest with binary classification chapter 3.8. We look at more than 100 instances and determine the three most important classes.
Misclassifications: We categorize those instances as misclassified where the target variable matches the reviewer's text therefore can only be explained by performance issues of the classifier. **25,48%** of our reviews fit in this category

and are therefore subject to improvement (e.g "Nice Pool, friendly staff." True: good, Label: bad). We take a closer look into this and determine the top four reasons why misclassifications occured: *1)* unexplainable reasons with 55% of all misclassifications. *2)* Expressive words that may have confused the classifier with 26% (e.g "Very good hotel. No reasons to *complain*" True: good, Label: ok). *3)* Very short reviews that did not give much insight into sentiment with 11%. *4)* Other causes with 8%.
Corner Cases: Instances that only partially match their label are classified as corner cases. We found that this is the case for **29,13%** of our reviews.
Outliers: Instances that do not match their label at all are classified as outliers and are not subject to improvement. This is the case for **45,38%** of the instances. Oftentimes customers outline special aspects of their stay instead of judging the overall service. We experience that luxury hotels are often rated by customers with very high standards. Since the set was generated by many international guests, different feedback cultures might have played a role as well. While some guests only outline negative aspects of the hotel others outline positive aspects of their stay as well. The high occurrence of outliers proves the very challenging nature of the problem.

4.2 Comparison

Baseline Experiments: The *sentiment lexicon* fell short of expectations with a 0.46 f1-score in combination with the Opinion Lexicon. The Opinion Lexicon outperforms the Vader Lexicon by around 0.08. We expected the Vader Lexicon to beat the Opinion Lexicon since it takes polarity and the intensity of sentiments into account. The *Naive Bayes* implementation achieves medium results around 0.52 with the BoW presentations but lacks sufficiency with word embedding features. The Multinomial NB approach achieves better results on the sets. We already suspected from Gall et al. [4] that a multinominal distribution works well in combination with term frequency vectors.

Looking to the more advanced methods, Logistic Regression works comparatively well on BoW embeddings (0.59) and on pre-trained Doc2Vec vectors (0.57). In general, **Random Forest** performs better on BoW embeddings. The optimized version achieves good results with 0.61 in f1-score on the test sample. **Linear SVCs** reached 0.57. Our top classifier is the **fastText standard text classifier** which achieves 0.62 in f1-score through the use of state of the art models. The accuracy of this classifier is 69%.

Besides sampling different classifiers and feature generations we also conduct experiments with **alternative architectures**. We introduce the architecture described in section 3.8 because frequent misclassification of the "ok" class. This architectural feat is only possible with classifiers that return the confidence of their choices. We were able to improve the score of the Random Forest classifier to 0.61 in f1-score. We also exploit advantages on diverse classifiers with ensemble

learning. By this way we were able to increase the f1-score to 0.59 which made it on the top ranks as our third best model.

Overall, balancing the distribution of our training set improved our results by a good margin. The biggest jump in performance can be noticed with the Multinomial NB classifier which increased by 0.1 in f1-score. This can be accounted to the fact that "ok" with 134913 samples and "bad" with 86851 samples are underrepresented compared to "good" with 293974 samples. Also, the f1-scores of most classifiers were worse for "ok" and "bad" already without upsampling and still with upsampling, which make us suspect that classifying those is more difficult.

Let's turn our attention to our feature vectors for a moment. Although the fastText text classifier obtained the best score overall, TF and TF-IDF turned out to be the most useful feature generating steps since BoW models usually achieve the best results among their respective classifier combinations. Although pre-trained Doc2Vec sets produced comparable results, Doc2Vec vectors computed with the training data were disappointing. We suspect that our corpus is too small to compete with pre-trained versions. Further, the word embeddings may not be able to play their context advantage over simple BoW methods because of the difference of the very specific domain of the task and the data the pre-trained word embeddings were trained on. Sentiment classification may also be mostly done with certain adjectives in the review which can be represented by the BoW vectors easily and does not need semantic representation.

Since we obtained our raw data from Kaggle, we found a range of other submitted kernels. Other kernels mostly classified only on two classes: Negative binned scores from [0-5] and Negative scores from]5-10] [18]. This task is for obvious reasons much easier than the task we conducted, since we also introduced a neutral class from [7.0-8.5]. Other Kernels (83% f1-score) on the other hand just took the negative_reviews column and the positive_reviews column and labeled them accordingly as positive and negative. Unlike us, who concatenated both review columns and binned scores as a target variable [2]. In fact we did not find any kernel that conducted the same task on the same data. Most kernels had simplified versions of our problem and therefore scored higher. But there are other kernels that conducted similar tasks with similar classes like a tweet sentiment analysis with a 79% accuracy based on 1500k tweets. We reached 69% accuracy with the same method, but with only 500k instances [15].

In conclusion we suggest to use our best classifier "fastText text classifier" to obtain scores from Hotel Reviews. We found that preprocessing and balancing help to improve scores. Overall, TF and TF-IDF were oftentimes the choice when it comes to feature vectors. Architectural tricks that focus on the neutral class produce good results too. The applicants should be aware of the many outliers in Hotel Reviews.

References

1. Text classification · fasttext, https://fasttext.cc/docs/en/supervised-tutorial.html
2. Bertolino, D.: Ranking hotel reviews using sentiment analysis (Mar 2020), https://www.kaggle.com/dbertolino/ranking-hotel-reviews-using-sentiment-analysis/notebook
3. Bishop, C.M.: Pattern recognition and machine learning. springer (2006)
4. Gall, R., Lobo, S., Davis, V.: Implementing 3 naive bayes classifiers in scikit-learn (May 2018), https://hub.packtpub.com/implementing-3-naive-bayes-classifiers-in-scikit-learn/
5. Géron, A.: Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow. dpunkt, https://www.oreilly.com/library/view/praxiseinstieg-machine-learning/9781492065838/
6. Hutto, C.J., Gilbert, E.: Vader: A parsimonious rule-based model for sentiment analysis of social media text. In: Eighth international AAAI conference on weblogs and social media (2014)
7. Joachims, T.: Text categorization with support vector machines: Learning with many relevant features. In: European conference on machine learning. pp. 137–142. Springer (1998)
8. Joulin, A., Grave, E., Bojanowski, P., Mikolov, T.: Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759 (2016)
9. Kotsiantis, S.B., Zaharakis, I., Pintelas, P.: Supervised machine learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering **160**, 3–24 (2007)
10. Lau, J.H., Baldwin, T.: An empirical evaluation of doc2vec with practical insights into document embedding generation. arXiv preprint arXiv:1607.05368 (2016)
11. Liu, B., Hu, M.: Opinion mining, sentiment analysis, and opinion spam detection (May 2004), https://www.cs.uic.edu/ liub/FBS/sentiment-analysis.htmllexicon
12. Liu, C.: Svm hyper-parameter tuning using gridsearchcv (Feb 2020), https://towardsdatascience.com/svm-hyper-parameter-tuning-using-gridsearchcv-49c0bc55ce29
13. Liu, J.: 515k hotel reviews data in europe (Aug 2017), https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe
14. Madan, R.: Hyperparameter tuning an svm a demonstration using hyperparameter tuning/cross validation on... (Nov 2019), https://medium.com/analytics-vidhya/hyperparameter-tuning-an-svm-a-demonstration-using-hyperparameter-tuning-cross-validation-on-96b05db54e5b
15. Malafosse, C.: Fasttext sentiment analysis for tweets: A straightforward guide. (Oct 2019), https://towardsdatascience.com/fasttext-sentiment-analysis-for-tweets-a-straightforward-guide-9a8c070449a2
16. McCallum, A., Nigam, K., et al.: A comparison of event models for naive bayes text classification. In: AAAI-98 workshop on learning for text categorization. vol. 752, pp. 41–48. Citeseer (1998)
17. Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., Joulin, A.: Advances in pre-training distributed word representations. arXiv preprint arXiv:1712.09405 (2017)
18. Oheix, J.: Sentiment analysis with hotel reviews (Dec 2018), https://www.kaggle.com/jonathanoheix/sentiment-analysis-with-hotel-reviews
19. Takeshita, S.: Simple sentence embedder, https://github.com/tofunlp/sister
20. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining: Pearson New International Edition -. Pearson Education Limited, Harlow (2013)