Project 2

Dal Col Giada 0093122
Keller Dirk 4282264

04.11.2022

1 Introduction

Introduction. User-generated online reviews have become more and more popular, influencing consumers' decision about services and products. Consequently, they become targets of profit attempts: counterfeit reviews can be written for self-promotion or to undermine reputation of competitors. In the present work the focus is on negative deceptive opinion spam, fabricated unfavorable opinions created to seem authentic with the purpose of reflecting badly on rivals. While email spam or Web spam such as fictitious advertisements can be easily identified by human readers, the same does not hold for misleading reviews[1].

Related work. Previous work from Ott and colleagues (2012)[5] showed that human judges perform poorly on detection of deceptive opinions, almost by a chance. In addition, Ott and colleagues(2012) showed that two linear machine learning classifiers machine learning classifiers (Naive Bayes and Support Vector Machines with linear kernel) were able to improve substantially over human performance. Noteworthy, these substantial improvement over other techniques and human judgments were in part realized by creation of the first large-scale, gold-standard data set for deceptive opinion spam research. Yet, the question remains whether more flexible classifiers can achieve even higher predictive performance.

Objectives and scope. The present work address the main task of finding a well-suited model architecture for text classification to label opinions as truthful or deceptive. Using the data set of previous related works of Ott and colleagues [6], [5] containing a collection of 800 reviews of Chicago hotels, respectively 400 truthful and 400 deceptive reviews. After necessary pre-processing steps on the data, four classifiers are trained and compared, namely Classification Tree [1], Random Forest, Linear Regression and Multinomial Naive Bayes Classifier. According to the literature on machine learning models,the Random Forest

[1] Two variants of the decision tree, one using cost-complexity pruning and one using stopping rules

classifier is expected to improves over the simple Trees. Second, it is expected that the Naive Bayes achieves a similar performance as it was observed in Ott and colleagues (2012)[6]. In addition it is expected that the Naive Bayes model might be identified as the top performing model, as it has proven to perform in particular well in text classification problems. From the best model we retrieved the top 5 features that most support the predicted classification to perform a more qualitative analysis of the data structure.
In Section 2 we describe the dataset, the methodology used for the collection of the reviews and the pre-processing steps performed before training the models. In Section 3 we present the classifiers used, with focus on the hyper-parameter tuning. In Section 4 we described the comparisons between the trained models, looking in addition at difference between unigrams and bigrams case. Finally, in Section 5 we present the conclusion of our work.

2 Data set

In order to be able to distinguish fake from genuine reviews, a data set, feasible for supervised-learning from Ott and colleagues (2011) [6] was acquired. The data set includes 800 instances, consisting a collection of truthful ($n = 400$) and deceptive ($n = 400$) reviews for the 20 most popular hotels in the Chicago area, resulting in 20 reviews per hotel for both categories (hence $20 * 20 = 400$).

Collecting truthful reviews. With respect to the truthful reviews, the authors [5] mined all negative (1- or 2-star) truthful from six popular online review communities: Expedia, Hotels.com, Orbitz, Priceline, TripAdvisor, and Yelp.[2] From the total number of mined reviews a subset of reviews were excluded that did not satisfy the following constraints: (1) reviews with more than two stars, (2) non-English reviews, (3) reviews with fewer than 150 characters[3] and (4) reviews written by first-time authors[4]. Finally, to ensure an equal class distribution, a subset ($n = 400$) of the remaining truthful reviews is selected by sampling the truthful reviews according to a log-normal distribution fit to the lengths of our deceptive reviews [6].

Collecting deceptive reviews. The deceptive opinions for the same 20 hotels were gathered using Amazon Mechanical Turk (AMT). Each review originates from a single, distinct individual that was instructed to "assume that they work for the hotel's marketing department, and to pretend that their boss wants them to write a fake review (as if they were a customer) to be posted on a travel review website". Additionally, the review needs to be sound, realistic

[2] Although, reviews mined travel communities contain reasonably small deception rates, the ground truth status of the truthful reviews can be compromised [5]

[3] submission is considered unreasonably short if it contains fewer than 150 characters

[4] opinions of first-time authors are more likely to contain opinion spam, which would compromise the validity of the truthful reviews [6]

and portray the hotel in a negative light. Any submission found to be of insufficient quality (e.g., written for the wrong hotel, unintelligible, unreasonably short, plagiarized, etc.) was rejected [6].

Training and test set. The data set is split into a training set, consisting of 640 reviews and a test set of the remaining 160 reviews. Note that 20-fold cross validation is used for the hyperparameter tuning to increase the number of training's instances for for each fit of a model. The class labels are assigned to "0" for deceptive reviews and "1" for truthful reviews.

2.1 Pre-processing the data

Features. From these reviews, individual unique words (unigrams) were extracted, forming the features that contribute to the prediction of the class. The number of features extracted and the type of features included is determined by the choices made in the data prepossessing as well as by the in-build feature selection of the specific algorithm (e.g. L1 regularization in the logistic regression). In addition to the unigram feature set, another data set was constructed that includes both bigrams and unigrams, as a combined feature set of uni- and bigrams allows to construct and select more relevant features for the classifiers. In addition, the choice to include bigrams can address the in practice often violated feature independence assumption of the Naive Bayes classifier, inflating the calculation of the class probabilities, which in turn can affect the prediction of the class. This assumption does not hold for text classification problems, because words in natural language are often correlated with each other. In particular, in the English language, semantically unique concept, such as the name of a hotel addressed in the review, are often represented as an ensemble of two or more words (e.g. 'Hilton hotel' or 'front desk' but not 'doorman'). Including bigrams in the data set, allows to treat the correlated words as single feature, potentially reducing the overestimation of the class probabilities and, hence, potentially improve the classification performance metrics.

Standardization. Crucially, to achieve good classification performance an extensive text pre-processing and feature selection on the data was performed. The aim is to include only relevant features that are able to distinguish between deceptive and truthful reviews, allowing the improvement of the performances of all models. In the first step all word forms and documents were standardized with respect to certain linguistic characteristics. First, capital letters were lowered to small letters, punctuation and numbers were excluded from each document ('review'). Punctuation considers characters such as, commas, marks or other separators characters for words. Since, separation characters are often not informative features to distinguish between class labels, this preprocessing step will not only standardize the input, but will also reduce the number of (unique) features to improve generalization. Another important Standardization step is lemmatization - lemmatization is transforming a given word to its *base form*,

that is an abstract representation, subsuming all the formal lexical variations which may apply. In contrast to stemming, lemmatization is an even stronger semantic standardization, which extends beyond simple word reduction by considering a language's full vocabulary to apply a morphological analysis to words, aiming to remove inflectional endings only and to return the base or dictionary form of a word - the lemma. For example, this might involve core-word extraction ($meeting \rightarrow meet$), tense conversion ($was \rightarrow be$) and plural to singular conversion ($mice \rightarrow mouse$). After, lemmatization, the different morphological forms are compressed to 6338 base forms for the unigram features and 47604 for including bigrams (from initially 6955 and 49391).

Pruning and normalization of (in-)frequent words. Moreover, each natural language corpus, follows a (power law) distribution - known as Zipf's law [8]: a few words (close class words) are very common, but the large majority of words (open class words) are rare in the data. Thus, including raw counts or even basic frequencies is a problematic approach in vectorize word embedding, as frequent words in the corpus are expected to have high counts in any document (e.g. review), regardless of whether they occur more often with one class or the other. Hence, neither very frequent nor very infrequent words are informative features for a machine learning model. Therefore, the implications of Zipf's Law for a corpus-based model is, regardless of the corpus size, there will be a lot of infrequent (and zero-frequency!) words that compromise a classifier's generalisation ability. To circumvent this problem, the features that do not satisfy the following conditions were excluded from the set of features:

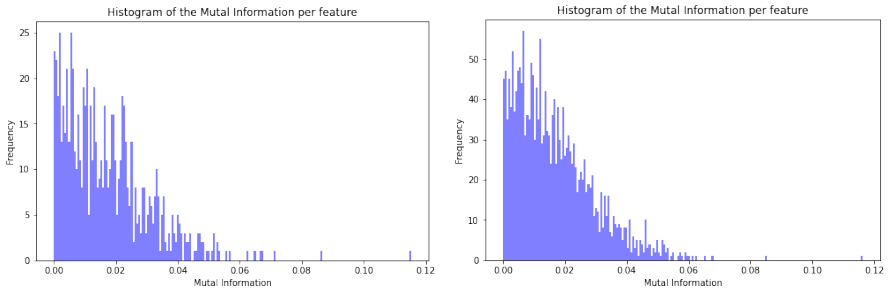
- (1) Function words, that are syntactically relevant and hence tend to be frequent (using the stopwords library from the 'nltk' package).
- (2) Other very frequent words that occur in more than 2/3 of all documents of the training set ($n = 426$).
- (3) Artifacts (e.g. spelling errors), infrequent and unique words, which do not occur in more than 1% of all documents of the training's set ($n = 6$).

Altogether, this feature pruning eliminates 4972 and 45537 features for unigram and bigram (initially 6338 and 47607 features), leaving a subset of 1368 and 2070 features, respectively.

Mutual information criterion. In addition to the term-frequency selection, the features are selected accordingly to their degree of dependence on the class label, as quantified by the Mutual Information Criterion (MI). The MI of two random variables is a measure of the mutual dependence between the two variables. Features were excluded if the expected amount of information in the feature about the class label is zero, meaning that the feature had no relationship to the class [5]. This leaves a final set of respectively 753 and 1060 features

[5] the larger the value, the greater the relationship between the two variables and the variables are independent if the value is 0.

for the uni- and the combined uni- and bigram feature sets, where each feature represents a unique (lemmatized) word from the corpus.



(a) (b)

Figure 1: Histogram of values of Mutual Information between features of our dataset and class labels ((a) for unigrams, (b) for uni- and bigrams)

TF-IDF representation. The information in the documents were chosen to be represented as 'Bag of Word' (BoW) embedding using a tf-idf score instead of raw counts. The words were vectorized with respect to the degree of the presence of a word, measured by the tf-idf score (a real number between 0 and 1), and the absence (denoted as '0'), while the order of the words (or the syntactical aspects of the sentence) were discarded. Since most words do not occur in a a single review, the document-term vector is therefore a sparse vector. The tf-idf score is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The value increases proportionally to the number of times a word appears in the document and is off set by the number of documents in the corpus that contain the word (idf); the formula for the inverse document-frequency is

$$idf(w) = \log \frac{N}{df(w) + 1}$$

where N = total number of documents and $df(w)$ = number of documents containing the word w. Notice that rare words have a high idf values.
This idf normalization improves over the simple BoW representation, since it normalizes the word frequencies in the corpus, controlling for the (in-)frequency of a word. Similar to the Bow representation, the TF-IDF representation is preferred over a Bernoulli representation using a one-hot encoded vector that simply quantifies the absence or presence of a word, but not the (normalized) degree of the presence. Thus a Bernouli representation would result in a loss of information, which is particular problematic for longer documents.

3 Models

3.1 Classifiers.

In continuation of Ott's work [6], the analysis of the data with linear classifiers (naive Bayes and Support Vector Machines with linear kernel), the predictive performance might be improved by training more (flexible) classifier. In line with this endeavor, four different machine learning models were chosen:

- (1) Multinomial Naive Bayes (generative linear classifier)
- (2) Logistic regression with Lasso penalty (discriminative linear classifier)
- (3) Classification trees with Gini index(non-linear classifier)
- (4) Random forests with Gini index (ensemble of non-linear classifiers)

Naive Bayes. The parameter settings for the Naive Bayes are defined by the α value for the Laplace smoothing to handle the problem of zero probability of certain features given a class. Moreover, an additional layer of feature selection is applied for the Naive Bayes model, where all features in the data set are sorted according to their mutual information value and different sets ($n = 100$) containing a random selection of the top features. Both the $\boldsymbol{\alpha}$ value and **the number of included** features according to their mutual information value are optimized with the hyperparameter tuning (see table 1 for the parameter settings, interval and optimized values).

Logistic Regression. For Logistic regression, the parameter settings consist of **Lasso (L1)** Regularization, the regularization value **C** (the inverse of $\boldsymbol{\lambda}$), a **liblinear** optimizer and the **maximal number of optimization** steps. Regularization of the loss function was used in order to avoid overfitting the logistic regression to idiosyncrasies of the training data. In contrast Ridge (L2) Regularization, which adds a penalty term which is equal to the square of the magnitude of coefficients, Lasso (L1) Regularization adds a penalty that is equal to the absolute value of the magnitude of coefficient, or simply restricting the size of coefficients. L1 regularization was preferred over L2, because it provides a more sparse solution to the feature space (selecting relevant features) as the BoW representation for text analysis result in large number of features. In addition, L1 provides a computational advantage because features with zero coefficients can be ignored. The parameter value C is optimized with hyperparameter tuning. With respect to the cost function used to minimize the prediction error of the Logistic regression, 'liblinear' was prefered over the Stochastic Average Gradient Accelerated Method (SAGA), both support the lasso (L1) regularization. The solver uses a Coordinate Descent (CD) algorithm that solves the optimization problem by successively performing approximate minimization along coordinate directions or coordinate hyperplanes ([3]). However, a major drawback of this optimization approach is that it might terminate in at a non-stationary point

(i.e. non-optimal) if the level curves of a function are not smooth. The SAGA optimizer supports the non-smooth penalty from L1 Regularization. Although SAGA is a preferred choice for sparse logistic regression, as it has a good theoretical convergence compared to other solvers [2], the logistic regression with SAGA was unable to converge in the above describe setting. In addition, a maximum of 300 iteration were chosen to ensure convergence of the model.

Decision Tree. Two variants of a Decision Tree, that differ in their approach to reduce overfitting, were defined. The first Decision Tree uses cost-complexity post-pruning, based on the $\boldsymbol{\alpha}$ value, measuring complexity as the total number of leaf nodes; the larger the $\boldsymbol{\alpha}$ value the more a complex tree structure is penalized. The second Decision Tree uses two stopping rules that apply pre-pruning by limiting the expansion of the tree. The first parameter is **nmin**, which is the minimal number of observations that a node must contain to allow another split (otherwise becoming a leaf node), while the second parameter **minleaf** is the minimal number of observations required for a leaf node; hence a split that creates a node with fewer than minleaf observations is not acceptable. All three parameters are optimized with hyperparameter tuning. In contrast to the ruled-based stopping, performing a greedy selection of the best split in a one-step look head, the cost complexity pruning expands a full tree to be pruned back later. In general, both Decision Trees use the **Gini index** as a measure of node impurity.

Random Forest. Instead of expanding a single tree and prevent overfitting by pre- or post-pruning, Random Forest models avoid overfitting by relying on bootstrap aggregation (similiar to bagging). In essence, the averaged prediction of multiple tree, constructed on multiple bootstrapped samples, inform the prediction of the class label. Since prediction of the trees are correlated, a random set of features is used before each evaluation of the best split in the current node. For the random forest, these two parameters - **the number of trees** and the **number of features** (nfeat) deviate from the default settings and are both optimized with hyperparameter tuning.

3.2 Hyperparameter tuning with crossvalidation.

Crossvalidation. Each model was trained and tested with a crossvalidated hyperparameter tuning procedure. For the crossvalidation procedure, the data set of ($n = 640$) was split into 10 folds, of which the test set contained 64 instances. Crossvalidation allows to improve the hyperparameter tuning, because more training instances can be used to fit the model with the specific hyperparameters settings.

Bayesian optimization. The hyperparameters selection procedure was optimized with Bayesian search. Bayesian optimization, in contrast to random or grid search, keeps track of past evaluation results, which then use to form

a probabilistic ('surrogate') model, mapping hyperparameters to a probability of a score on the objective function. Thus Bayesian methods select the best performing hyperparameters on the surrogate, apply it to the true objective function and update the surrogate model. This procedure is repeated until the max iterations ($n = 30$) are reached. This method is preferred over Grid- or Random-search, since, at a high-level, Bayesian optimization methods are more efficient because they choose the next hyperparameters in an informed manner. Meaning, this method chooses only the relevant search space and discards the ranges that will most likely not deliver the best solution. By evaluating hyperparameters that appear more promising from past results, Bayesian methods can find better model settings than random search in fewer iterations. Thus, instead of randomly choosing the next set of parameters or exhaustively search over the hyperparameter space, the algorithm optimizes the choice, and likely reaches the best parameter set faster than the previous two methods, in particular if the feature space is large and model learning is inherently slow [7].

The scoring strategy to select the performance of the cross-validated model is the 'AUC' rather than 'accuracy' and 'F1' score. Accuracy is problematic when the number of instances per class are not uniformly distributed. In contrast to both the 'AUC' (Area Under the Curve) metric and the 'F1' score, accuracy does neither take recall nor precision, which is the performance with respect to false negative (how many true positives were missed) and false positive samples (many true positives were detected), respectively. Because the class distribution for the present data set is homogeneous, the AUC is unbiased towards the negative samples and was therefore selected.[6]

Parameter Settings. For each of the five models (two variants of the the Decision Tree) a parameter distribution for the set of relevant optimizable hyperparameter was specified. Table 1 summarizes the parameter settings chosen, the parameter interval and the result of the Bayesian optimization. Note, for the Naive Bayes classifier a minimum of the top 150 features (indicated by the mutual information criterion) were included to ensure that a minimum of the best features were included; for the Random Forest classifier a minimum of the natural logarithm of total number of features was the lower bound, while half of the total features where the upper bound of the search space, as too little features have a smaller chance to contain the best features and too much result in more correlation of the predicion.

[6]The AUC of the ROC describes all different levels of F-score values for different threshold, the AUC averages the F1 score over all thresholds. Therefore, the AUC metric is more heavily influenced by an unbalanced class distributions (a side-effect of too many negative examples), such that an increasing AUC does not per se reflect a better classifier [4].

Classifier	Parameter	Distribution	Unigram optimal	Bigram optimal
Multinomial Naive Bayes	Laplace smoothing ($\boldsymbol{\alpha}$)	0.01 - 1 (i = 0.001)	0.747	0.663
	Number of Features	150 - N (i = 1)	756	1134
Logistic Regression	$\mathbf{1/\lambda}$ ($\boldsymbol{C}$)	1 - 10000 (i = 1)	8854	8765
Cost-complexity Decision Tree	$\boldsymbol{\alpha}$ (***cpp_alpha***)	0 - 0.5 (i = 0.001)	0.016	0.044
Rule-based Decision Tree	nmin (***min_samples_split***)	2 - 30 (i = 1)	14	14
	minleaf (***min_samples_leaf***)	1 - 30 (i = 1)	25	27
Random Forest	Features per split (***max_features***)	log(N) - N/2 (i = 1)	240	348
	Number of Trees (***n_estimators***)	5000 - 10000 (i = 100)	8300	8700

Table 1: Summary of hyper-parameters search per model. See the section classifiers for an explanation of the parameters; N = maximum number of features; i = step size.

4 Analysis

After the Baysian optimization converged on the most optimal hyperparamter, each model with its accompanying paramter settings was fit on the full trainings set and tested agaisnt the hold-out test set. The performances of the five models, as quantified by the accuracy, precision, recall and F1 score, are summarized in Table 2 and Table 3, respectively for unigrams features and combined uni- and bigram features.
From Table 2 it's clear that the top performer of our models is Multinomial Naive Bayes which scores the highest values in all the performance measures.

Classifier	Accuracy	Precision	Recall	F1 score
Classification Tree (post-complexity)	0.625	0.61628	0.6625	0.63855
Classification Tree (rules)	0.63125	0.63636	0.6125	0.62420
Random Forest	0.725	0.71429	0.75	0.73170
Logistic Regression	0.7875	0.78049	0.8	0.79012
Multinomial Naive Bayes	0.85625	0.84337	0.875	0.85890

Table 2: Performance measures unigram features

Comparing model performance. In line with the literature, the Decision Tree with post-complexity pruning performed better than rule-based pre-pruning, for both the uni and the combined uni- and bigram data set. But the increase in performance is substantially larger for the uni-bigrams. This is sensible, as post-pruning usually results in a better tree than pre-pruning because pre-pruning is greedy and may ignore splits that have subsequently important splits. In addition, the Random Forest is able to outperform both single Decision Tree models by almost 10% in the 'F1' score ($\boldsymbol{\Delta}$(dt, rf) = 0.1). Although, Logistic Regression appears to be better than Random Forest, it is

outperformed by Multinomial Naive Bayes From this first analysis of the performance measures Multinomial Naive Bayes appears to be the best model for our classification problem. Therefore, the performance of the generative linear model (multinomial naive Bayes) is substantially higher than the the discriminative linear model (regularized logistic regression) for both uni and uni-and bigrams ($\boldsymbol{\Delta}$(nb, lr) = 0.05).

Comparing uni/bigram performance. In addition, when comparing the model performance fitted on the unigram feature set against the combined feature set, the inclusion of the bigram features significantly improves the performances for all the five models, allowing to increase the number of relevant features that characterize the two different classes, capture unique semantic concepts represented by an ensemble of words and replace those individual words that are highly correlated with each other.

Classifiers	Accuracy	Precision	Recall	F1 score
Classification Tree (post-complexity)	0.63125	0.62353	0.6625	0.69565
Classification Tree (rules)	0.69375	0.69136	0.7	0.65823
Random Forest	0.7875	0.78049	0.8	0.79012
Logistic Regression	0.8125	0.8125	0.8125	0.8125
Multinomial Naive Bayes	0.8625	0.84524	0.875	0.86585

Table 3: Performance measures uni- and bigram features

McNemar test: Testing statistical significance. In order to have a more accurate and objective comparison between model performance metrics, the McNemar's test is applied to the model predictions. The null hypothesis of the McNemar's test states that the proportion of error in the test set is the same in the two classifications, meaning that the two models disagree in the same way. If the null hypothesis cannot be refused, then as a direct consequence the difference in the accuracy is not statistically significant; in fact, having the same proportion of error means having the same proportion of correct prediction (accuracy). Specifically, the test is based on the contingency table, a tabulation displaying the frequency of two binary variables counting the frequencies of the combinations of correct/incorrect predictions in the two models. An exact binomial test is performed, since some values in the contingency tables are less than 25 and the usual chi-squared approximation would not be accurate. Calling $\boldsymbol{a}$ the counting of the combination "correct predictions for the first model and incorrect predictions for the second model" and $\boldsymbol{b}$ the counting of the combination "correct predictions for the second model and incorrect predictions for the first model", the exact binomial test compares $\boldsymbol{a}$ with a binomial distribution with size parameter n = $\boldsymbol{a} + \boldsymbol{b}$ and probability of success of 0.5.
With a significance level of less than **5%** the performance of the Multinomial Naive Bayes appears to be significantly different in accuracy compared to all

the other classifiers, confirming our choice of top performer. The performance of logistic regression also appears to be significantly different remaining three models, based on accuracy. Regarding the Random Forest classifier, the significant test confirms the expectations that the performance is statistically different from the two Classification Trees, since the Random Forest averaging prediction of several trees. However, the two Decision Tree Classifiers clearly are not statistically different in accuracy, since they are the same type of classifier, built with different pruning rules.
Adding bigrams does not change the situation regarding the statistical differences of the models, so Table 4 refers to both cases.

Multinomial Naive Bayes	Logistic Regression	*
	Random Forest	* **
	Decision Tree 1	* **
	Decision Tree 2	* **
Logistic Regression	Random Forest	*
	Decision Tree 1	* **
	Decision Tree 2	*
Random Forest	Decision Tree 1	* **
	Decision Tree 2	* **
Decision Tree 1	Decision Tree 2	-

Table 4: Statistical significant differences in accuracy between classifiers. One asterisk mean the McNemar's test result in the refuse of null hypothesis with a significance level of **5%**, two asterisks mean significance level of **1%**, no asterisks mean the null hypothesis can not be refused with either of the two.

Qualitative analysis of truthful or deceptive reviews. Interestingly, when extracting from the best performing Naive Bayes model the first five word that support the classification towards deceptive and truthful the most, the truthful negative reviews seem to consist of more concrete concrete words, such as doorman, season or security, that are specific to a hotel environment, while the words from the deceptive review seem to be more general words, such as 'smell' or 'luxury'.

- **Most supporting truthful:** 'stated', 'season', 'doorman', 'security', 'cool'
- **Most supporting deceptive:** 'egg', 'turned', 'smell', 'millennium', 'luxury'

5 Conclusion

The present work aimed to identify well-suited model architecture to distinguish genuine from fake reviews. In accordance to the hypothesis, the Random Forest classifier outperformed the single trees. In addition, Naive Bayes was demonstrated to be the best performing classifier, achieving similar results as observed Ott and colleagues (2012)[5]. This is not surprising, since a large set of features usually result in overfitting. Naive Bayes is more specific for text mining and can handle smaller feature sets well. For Naive Bayes only a subset of features was selected, possibly contributing to generalization performance demonstrated by the high performance on the holdout test set. Moreover the classifier is fast and has low storage requirements (as compared to random forest). In particular, the Naive Bayes demonstrated a high recall rate, which is in particular important in the goal of identifying and pruning the negative fake reviews from the set of all negative reviews, even if it is at the cost of some genuine ones. Recall is more important than precision in this practical application, as it is desired to have a set of valid and genuine negative opinions. In addition, high-recall models allow to effectively combat negative reviews generated by bots and human alike to limit the potential damage on the reputation of businesses and persons. Nevertheless, the selected models were not able to improve over the classifiers selected by Ott and colleagues (2012)[5].

In addition, including bigrams improved the performance of almost all classifiers, albeit including bigrams did not change the model selection based on accuracy. This might be attributable to the specific preprocessing steps taken, highlighting the importance of including (a smaller subset of) relevant features that are correlated to the class labels.

Possible future improvements could include the deployment of more complex architecture that can capture more complex 'non-linear' patterns in data, and, likewise, including a more complex and sophisticated vector representation of the reviews. A possible suggestion might be to use a vector representation, allowing to include word semantics, using language models, such as BERT or GloVe embeddings, as well as information over the syntactical structure. For this purpose LSTM or transformer models could be utilized; a simple improvement over the Naive Bayes model using a tf-idf vectorization could be a classical fully-connected model.

References

[1] Charles Bond and Bella DePaulo. "Accuracy of Deception Judgments". In: *Personality and social psychology review : an official journal of the Society for Personality and Social Psychology, Inc* 10 (Feb. 2006), pp. 214–34. DOI: 10.1207/s15327957pspr1003_2.

[2] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives". In: *Advances in neural information processing systems* 27 (2014).

[3] Rong-En Fan et al. "LIBLINEAR: A library for large linear classification". In: *the Journal of machine Learning research* 9 (2008), pp. 1871–1874.

[4] László A Jeni, Jeffrey F Cohn, and Fernando De La Torre. "Facing imbalanced data–recommendations for the use of performance metrics". In: *2013 Humaine association conference on affective computing and intelligent interaction*. IEEE. 2013, pp. 245–251.

[5] M. Ott, Claire Cardie, and Jeffrey Hancock. "Negative Deceptive Opinion Spam". In: (Jan. 2013), pp. 497–501.

[6] Myle Ott et al. "Finding Deceptive Opinion Spam by Any Stretch of the Imagination". In: *CoRR* abs/1107.4557 (2011). arXiv: 1107.4557. URL: http://arxiv.org/abs/1107.4557.

[7] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems* 25 (2012).

[8] George K. Zipf. "Human behavior and the principle of least effort." In: *Journal of Clinical Psychology* 6.3 (1950).

Appendix

For all the analysis we used Python and the following functions in the corresponding packages.

	Package	**Function**
Decision Tree	sklearn.tree	DecisionTreeClassifier
Random Forest	sklearn.ensemble	RandomForestClassifier
Logistic Regression	sklearn.linear_model	LogisticRegression
Multinomial Naive Bayes	sklearn.naive_bayes	MultinomialNB
TF-IDF representation	sklearn.feature_extraction.text	TfidfVectorizer
Bayesian search	skopt	BayesSearchCV
Mutual Information	sklearn.feature_selection	mutual_info_classif
Confusion matrix	sklearn.metrics	confusion_matrix
Accuracy	sklearn.metrics	accuracy_score
Recall	sklearn.metrics	recall_score
Precision	sklearn.metrics	precision_score
F1 score	sklearn.metrics	f1_score
McNemar's test	statsmodels.stats.contingency_tables	mcnemar

Table 5: Python function and packages used for our text mining analysis