Data Mining Assignment 2

Daniele Di Grandi
7035616

Jens Hartkamp
6021271

Alice Hartog
7035608

Classification for the Detection of Opinion Spam

1 Introduction

Online reviews have increased in number in the last years, thanks to the large distribution of devices connected to internet, at least in the most developed countries in the world. This lead to the possibility for everyone to spread their thoughts on experiences had in product purchases, restaurants and hotels through reviews. Many websites have been developed only for reviews in any type of environment (e.g. TripAdvisor for restaurants and hotels or Trustpilot mainly for websites selling any kind of product) and it has now become automatic for everyone to look at reviews before choosing a restaurant or an hotel, buying a certain product or starting an experience.

In any case the pattern is the same: if the majority of reviews are negative, thus if the average evaluation score is around or below a certain value, the consumer tends to not select the considered item. As an example, usually the evaluation is based on a scale of 1-5 stars. In this case an average evaluation of 2.5 by common sense is already considered to be low. In some cases even higher scores are considered to be low even though reviewers have rated the item or service sufficient or better. Furthermore, it is a common experience that even though an item or experience has plenty of positive reviews, only a few negatives are enough to let people change their mind.

This phenomenon is the reason why it is quite popular to write several *deceptive reviews* [1], i.e. fake reviews, to discredit competitors. Some of these types of negative reviews are easy to be detected as spam, but most of them are very well studied in order to seem truthful [2]. These well studied deceptive reviews are an even bigger problem for businesses because they are harder to identify as such.

Ott et al. [3] studied this topic in their paper focusing on detecting deceptive and truthful negative reviews on hotels, in order to find a way to understand whether a review is real or not. According to this paper, the best human judge is accurate only 65% of the time, reason why an automated text classifier was needed.

Problem motivation. This paper focuses on negative reviews, more in particular *deceptive* and *truthful* ones, and uses text mining to classify them through four different classifiers: Multinomial naive Bayes, Regularized Logistic Regression, Classification Trees and Random Forests.

Multinomial naive Bayes is a learning algorithm frequently used in text classification, based on Bayes' theorem, where features are mutually independent (i.e. if one features occurs, this does not affect the probability that a second feature occurs) [4]. It uses word counts (frequency) to predict the class of a review. To do so every unclassified review is represented as a list (also called bag) of words. For every word of the unclassified review the probability is multiplied for both the deceptive and truthful class. This probability is calculated dividing the amount of occurrences by the total amount of words in the training data for both the truthful and deceptive reviews, respectively. Then, Laplace smoothing [5] is performed to avoid zero probabilities estimates. The new review will then be predicted to belong to the class with the highest score.

Regularized Logistic Regression is defined as a model measuring the relationship between a dependent and one or more independent variables. Dependent variables are the predicted ones, while independent variables are used to predict [6]. The fact that it is regularized is done to avoid overfitting [7].

Classification Trees are search trees which split data based on certain attributes [8]. In our case every unigram (word) or bigram (sequence of two words) are the attributes on which the classification tree can split. This means that when classifying a new document one will continue left or right in the tree depending on the frequency of a word in the review which has to be classified. Once the tree is fully traversed through it will predict whether this review is deceptive or truthful based on the class of the leaf node in which the unclassified review ended its traversal.

Random Forests are essentially an ensemble of classification trees with the same distribution but each one is constructed on random attributes and by sampling with replacement, so that they all differ from each other [9]. Then, the predicted class is the class that the majority of trees has singularly predicted.

The code we developed that supports this analysis can be found at reference [10].

2 Literature review

Text mining is a technique which has been used in many applications. For example, to measure brand position in the hotel industry [11], to do a sentiment analysis on tweets regarding the Syrian refugee problem [12], or to determine whether a text is deceptive or truthful [13]. For this paper the latter is of most interest.

Distinguishing deceptive from truthful texts has proven to be relatively difficult for humans [3, 14]. Moreover, determining whether reviews are deceptive would be incredibly labour intensive due to the number of reviews being published on the internet. Using computer power to detect deceptive reviews instead would be ideal as this is a lot more efficient and possibly more accurate when compared to using human resources.

In their paper, Ott et al. [15] showed that human judgement is significantly inferior to judgement using machine learning with a 16.6% difference. This is also the point where some problems arise. To properly train a model, labelled datasets are often required. These datasets are hard to come by in this context as it is difficult to distinguish deceptive from non-deceptive reviews for humans. Using artificial deceptive reviews, which are retrieved from crowd sourcing initiatives such as Amazon Mechanical Turk [16], to create such a dataset could also lead to issues as the people writing these deceptive reviews often lack domain knowledge, psychological state of mind or experience to write convincing fake reviews [17, 18] . Vidanagama et al. [17] even conclude machine learning should focus on solving this problem while using no or very limited labelled data and find other means to solve the problem of deceptive reviews.

However, most current state of the art methods still heavily rely on labelled training data and even though research on unlabelled data is being performed with reasonable results [19] this paper will still focus on training classifiers using labelled data. To get some insight on what to expect regarding the results, a few papers with similar research were examined. In a performance analysis of supervised techniques regarding review spam detection Badresiya et al. [20] found decision trees performed the worst with 51% accuracy. Next came naive Bayes and K-Nearest Neighbors (KNN) with 66% and 68% accuracy, respectively. Lastly, support vector machines (SVM) and Logistic Regression performed best with 82% and 83% accuracy, respectively. From these results we see quite a big difference in performance of these methods when used on the same training data.

In a paper discussing different methods for detecting deceptive from non-deceptive online reviews by Banerjee et al. [21], a similar result is shown where naive Bayes was performing the worst followed by SVM and Logistic Regression. Although the differences in accuracy in this paper were a lot smaller, this could be an indication of what to expect in the result section. Interestingly Banerjee et al. also used a Random Forest as opposed to a Decision Tree. This Random Forest performed nearly as good as the other classifiers which could indicate Decision Trees in the form of a Random Forest could still be a useful classifier whereas a Single Tree would perform far worse. Other papers regarding similar topics and using (a selection of) these classifiers often show results which are in line with the results found by Badresiya et al., where Logistic Regression and SVM often perform (slightly) better and naive Bayes and Random Forests perform slightly worse than the other classifiers [3, 19, 22, 23, 24].

3 Data description

The dataset we used has been collected by Ott et al. for their analysis in papers [25][3] and, in particular, in this paper we used data relative to [3], containing a dataset for negative reviews. These reviews are divided into two categories. The first one are *deceptive* reviews, which have been gathered from Mechanical Turk [16]: 400 reviews, 20 for each of the 20 most popular hotels in Chicago, written specially for this analysis by several workers in a maximum time of 30 minutes, where they had to imagine they were working as marketing operators in an hotel and thus write realistic negative reviews. The average length of these reviews was 178 words. The second category are *truthful* reviews, 400 real negative reviews found on the internet taken from six most popular review websites (Expedia, Hotels.com, Orbitz, Priceline, TripAdvisor and Yelp [3]). However there is a problem. Since these reviews come directly from the internet, we can not be 100% sure that those are actual truthful reviews, but, as we previously said, obtaining such a dataset of truthful reviews is a very difficult operation. Hence, we must assume these 400 reviews to be actual truthful. The length of these reviews was higher than the deceptive ones, thus they had been sampled using a log-normal distribution in order to fit them with the deceptives.

Since we have 400 truthful and 400 deceptive reviews, predicting the majority class would yield to an accuracy of exactly 50%. This accuracy can be considered as a baseline for a comparison of the models we will train.

All reviews have been split into separate text files and aggregated into 5 folds of 160 reviews each. Folds 1-4 of this dataset have been used as training set, while fold 5 has been used for the testing part. Thus 640 reviews have been used for training the model and 160 for testing it. In our model we decided to use the class '0' to identify deceptive reviews and the class '1' for truthful ones. Also, we will use Python as a programming language for our analysis, in order to exploit all the powerful libraries and the out-of-the-box functions that Python has for Machine Learning.

4 Preparatory steps

4.1 Pre-processing

The pre-processing phase is necessary for the model to have an easier and structured text that can understand and therefore process. The library used for this phase is called `nltk` (Natural Language Toolkit [26]), developed in Python on purpose for processing English written texts. As a first thing, the text has been tokenized, which means split into a list of words, punctuation or numbers considering as separator the space between them. Then every letter in every word has been set to lower case to make sure the model did not consider one word as two separate words only because of an upper case letter; punctuation and numbers were removed right after this passage, because they were not useful for the purpose. Lastly, all the English stop-words have been recognized using `nltk` and deleted from the token list. Some examples of these removed stopwords are: *I, me, my, who, this, is, am, been, having, must, should, same, all, any, when, before, because.* Using the method `PorterStemmer`, from `nltk` *stemming* has been done, which is the process of reducing words to their stem (or root). This is important for merging words with the same meaning but, for example, with different verb tenses or plural nouns, as our models would otherwise recognize them as two different words. Some examples of stemmed words are: 'decid', 'experi' or 'locat', the stem versions of, for example, 'decided', 'experienced' or 'located'.

4.2 Bag of words

All words have been divided into *unigrams* and *bigrams.* In general, an n-gram is a sequence of n items, in our case words. Thus an unigram is a list of single words and a bigram a list of two consecutive words [27]. Unigrams and bigrams are taken from each separate review and not from the corpus (a file where all reviews have been put together without distinction), so in this way two different reviews do not risk to become correlated as could happen in the corpus (since reviews are not separated here, the last word of one review could form a bigram with the first word of another review, where the two reviews could even belong to different classes). After selecting all of them, they are put together to fill our empty *bag of words*, a representation of the dataset where only unigrams and bigrams are included after being pre-processed, with no logical structure and only once, with their frequency in the dataset. These unigrams and bigrams constitute our features for the different models. In this paper we will compare the performance of the models using only unigrams and the performance by also adding bigrams as features.

Analyzing the dataset, we discovered that is composed by 4,617 unique words for the unigram case and 38,631 unique 'words' for the bigram case. If we want to use each unique word as a feature, then we will obtain 4,617 features for the unigram case and 43,248 features for the bigram case. Among these features, the frequency of unigrams that are present only once in the corpus is 42.1% and the frequency of unigrams present twice is 14.32%. In other words, 42.1% of the corpus is composed by unique words. For the bigram case, 85.19% of bigrams appear only once and 10.25% twice in the corpus.

Since the total number of features is way too high and the overall feature vector is sparse, a good idea to prevent overfitting is to rationally select these features.

4.3 Feature selection

Feature selection has to be made only for the Multinomial naive Bayes model, because the other models already have a built-in feature selection method: Logistic Regression has λ, a coefficient included in the LASSO penalty, which is an additional term used for punishing large amounts of features that lead to a more complex model and thus risk to lead to overfitting. The LASSO penalty is calculated using the following formula:

$$E(\beta) = -l(\beta) + \lambda \times \Sigma_{j=1}^{m} |\beta_j| \tag{1}$$

Where $-l(\beta)$ is the negative log-likelihood function, j is the number of features and β_j the parameters to estimate.

Classification Trees do feature selection through the cost-complexity pruning coefficient α, included in the cost-complexity equation:

$$C_\alpha = R(T) + \alpha|\tilde{T}| \tag{2}$$

Where $R(T)$ is the resubstitution error and $\alpha|\tilde{T}|$ gives a penalty for the complexity of the tree, that is, how many leaf nodes it is composed of.

Finally, the Random Forest model selects features using the parameter $nfeat$, that denotes the number of features that should be considered for each split.

All these parameters will be tuned through cross-validation.

Thus, for the Multinomial naive Bayes model, that does not have a built-in feature selection method, we used two procedures for manually doing it: deleting *sparse words* and using *mutual information.*

Sparse words. Deleting sparse terms, also called sparse features in our case since each word is a feature, is a valid method to select features in order to avoid overfitting due to a large number of features. The method consists in eliminating features (unigrams and bigrams) that occur in the corpus less than a certain threshold that we called σ, which at this stage is still unknown because it has to be tuned using cross-validation.

In order to have an insight on frequency of the words in the corpus, we performed a frequency analysis and the 20 most frequent unigrams and bigrams found are shown in Table 1.

Unigrams		***Bigrams***	
Word	**Freq.**	**Words**	**Freq.**
room	1730	front desk	217
hotel	1511	room servic	118
stay	922	stay hotel	104
chicago	532	hotel chicago	70
would	505	look like	60
servic	412	chicago hotel	57
one	391	custom servic	56
night	362	call front	55
get	350	hard rock	48
bed	319	got room	47
call	317	would recommend	45
desk	309	even though	40
us	304	recent stay	39
time	302	never stay	39
like	301	anoth room	38
staff	300	go back	38
could	292	book room	37
check	290	hotel room	36
even	271	room clean	34
look	260	hotel stay	33

Table 1: *Most frequent 20 unigrams and bigrams in the corpus.*

We noticed that unigrams in general tend to be way more frequent than bigrams, which makes sense considering that for a specific bigram to be present the two words need to be consequent and this probability is not as high as having only one word. It is legit that unigrams such as 'room', 'hotel' and 'stay' have a very high frequency because usually when writing a review there is the tendency to insert them. A similar motivation is for 'chicago', since all the examined reviews come from hotels in Chicago, thus, citing the city's name is reasonable.

Mutual information. Mutual information measures mutual dependence between two variables. In other words, the uncertainty reduction of one variable obtained observing the other one. Mutual information is related to entropy and its expression is given by the equation [28]:

$$I(X,Y) = \sum_{x}\sum_{y} \frac{n(x,y)}{N} log_2 \frac{N \times n(x,y)}{n(x) \times n(y)} \quad (3)$$

Where X, Y are two random variables, $n(x,y)$ is the number of times that x and y are observed together, $n(x)$ and $n(y)$ are the number of times that x and y have been observed, respectively, and N is the total number of observations. So, in our case, X can stand for potentially any English word and computing $I(X,Y)$ will tell us how much information the presence (or absence) of the word X tells us about the deceptiveness or truthfulness of a review. Hence, the higher is mutual information, the better.

Similar to deleting sparse words, this method consists in eliminating features in the corpus that have mutual information lower than a threshold we called μ, that also has to be tuned using cross-validation. Unigrams and bigrams with the highest mutual information in the corpus are shown in Table 2.

Unigrams		***Bigrams***	
Word	**Mutual inf.**	**Words**	**Mutual inf.**
chicago	0.129	recent stay	0.039
smell	0.061	hotel chicago	0.037
recent	0.049	chicago hotel	0.031
luxuri	0.049	chicago millennium	0.027
locat	0.036	sheraton chicago	0.025
decid	0.035	hotel tower	0.019
elev	0.033	book hotel	0.018
millennium	0.029	millennium knickerbock	0.017
final	0.028	luxuri hotel	0.017
seem	0.26	arriv room	0.017
experi	0.026	fairmont chicago	0.017
great	0.024	conrad chicago	0.017
rude	0.024	knickerbock hotel	0.015

open	0.021	final got	0.015
pricelin	0.021	millennium park	0.014
make	0.02	help us	0.014
relax	0.02	wait hour	0.014
expect	0.02	jame chicago	0.014
cigarett	0.02	stay mani	0.014
review	0.02	four season	0.014

Table 2: *20 unigrams and bigrams with the highest mutual information value in the corpus.*

These values will be used in the cross-validation procedure to define the range of possible values for testing μ. Similar to the difference in frequency for unigrams and bigrams analysed earlier, the same happens for mutual information: unigrams have higher values compared to bigrams.

5 Quality metrics

The objective was to assess whether a negative review is truthful or deceptive. To do this, we used some performance indicators that will be defined in this section.

A *confusion matrix* is a matrix where columns represent the model predictions and the rows represent the actual classes that should have been predicted. Thus, the main diagonal tells us the number of correct predictions within each class. The general structure we adopted for this paper is showed in Table 3.

		Predicted class	
		Deceptive	*Truthful*
Actual class	*Deceptive*	TN (True negative)	FP (False positive)
	Truthful	FN (False negative)	TP (True positive)

Table 3: *General confusion matrix representation.*

Moreover, in order to establish how good is the model in the assessment, four performance indicators have been calculated based on the confusion matrix:

1. Accuracy: is calculated as

$$\frac{TN + TP}{TN + FN + FP + TP} \tag{4}$$

So is defined as the number of correctly classified cases over the total. In other words, it represents the fraction of correct predictions.

2. Precision: is calculated as

$$\frac{TP}{FP + TP} \tag{5}$$

So, among all cases classified as positive, how many of them were actually positive.

3. Recall: is calculated as

$$\frac{TP}{FN + TP} \tag{6}$$

So, among all the actual positive cases, how many have been classified as positive.

4. f1-score: is calculated as

$$2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{7}$$

Which simplified is:

$$\frac{TP}{TP + 0.5 \times (FN + FP)} \tag{8}$$

And measures an harmonic mean of precision and recall.

6 Experiments

6.1 Setup

In this section we will discuss the setup that we used for our experiments. The objective is clear: we want to train 4 classifiers with the same training data, use those data to also perform the tuning of some of the parameters that each classifier requires and use a separate test data to evaluate the models predictions. For this purpose, we mainly used the out-of-the-box classifier models contained in the `scikit-learn` library for Python: `LogisticRegression`, `DecisionTreeClassifier`, `MultinomialNB` and `RandomForestClassifier`. For the reproducibility of the results, we always have passed a random seed of 0 to each of these functions.

Furthermore, we assumed some of their parameters to be fixed, hence, they don't require the tuning process.

For the Regularized Logistic Regression we set the penalty to be the LASSO penalty `l1` and we used the `saga` solver.

For the single Decision Tree we decided to use `gini` as impurity measure and to always perform the best split, rather than a random one.

For the Multinomial naive Bayes we haven't set any of its parameters, but we will perform a cross-validation tuning procedure to determine the value of the parameters we called σ and μ.

Finally, for the Random Forest we decided to use the `gini` criterion and to use bootstrap samples when building trees.

All the parameters that we don't have mentioned in this section, and that we will not tune in the next section, have been left to their default value.

6.2 Cross-validation

In this section we are going to be discussing the procedure that we implemented in order to achieve the best possible values for the parameters that have to be tuned within each model. For that, we used a k-fold cross-validation, using the whole training set as input. This procedure contains an outer loop of k sets and an inner loop of l sets. The total dataset in input is split into k sets. One by one, a set is selected as the (outer) test set and the $k-1$ other sets are combined into the corresponding outer training set. This is repeated for each of the k sets. Each outer training set is further sub-divided into l sets. One by one, a set is selected as inner test (validation) set and the $l-1$ other sets are combined into the corresponding inner training set. This is repeated for each of the l sets. The inner training sets are used to fit model parameters, while the outer test set is used as a validation set to provide an unbiased evaluation of the model fit. This is repeated for many different hyperparameters and the validation set is used to determine the best hyperparameter set for this inner training set. After this, a new model is fit on the entire outer training set, using the best set of hyperparameters from the inner cross-validation. The performance of this model is then evaluated using the outer test set. As a side bonus, cross-validation used for tuning hyperparameters could also prevent overfitting: if the tuned hyperparameter is a feature-selection one, by correctly tuning it, we should obtain less features, hence, a more generalizable solution.

To implement the whole procedure, we used the `cross_score_val` function from the `scikit-learn` library [29] using a 10 fold cross validation because this is an accepted optimal number of folds according to the literature [30].

6.2.1 Multinomial naive Bayes tuning

As previously said, this model is the only one that does not have a built-in method to make feature selection, but we showed that is possible to manually implement it by removing sparse words and terms with low mutual information. However, what does it mean for a word to be 'sparse' or to have 'low' mutual information? This is exactly the thresholds that we want to tune in this section. Hence, we used cross-validation to understand the best values for the threshold σ that specifies how frequent has to be a word in order to be kept *and* the threshold μ that specifies what should be its mutual information in order to be kept. Thus, a word is kept as a feature only if it is present more than σ times in the processed corpus and has a mutual information greater than μ. Obviously, these two parameters could be different for the unigram and bigram cases, hence, they need to be tuned in a separate manner.

Unigrams. For the unigram case, we decided to test the following values:

σ**:** $[1, 2, 6, 10, 15, 20, 30, 50, 60, 80, 100, 200, 300]$

μ**:** $[0, 0.0001, 0.0004, 0.001, 0.0015, 0.002, 0.004, 0.01, 0.02, 0.04]$

From our experiments, leaving σ equal to 1 or μ equal to 0 means to not eliminate any feature, while setting σ to 300 or μ to 0.04 means to eliminate almost all the features. The resulting plot - in three dimensions, since we are changing two parameters at the time - is shown in Figure 1.

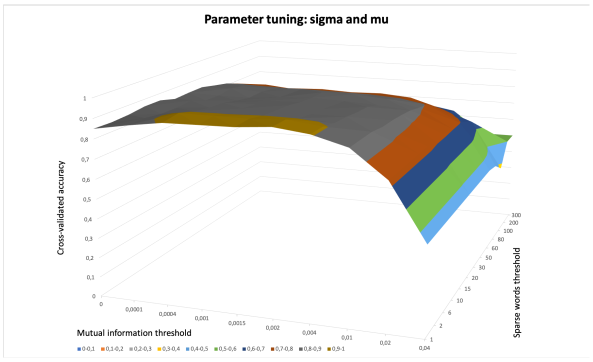

Figure 1: *Cross-validated accuracy for unigrams - Multinomial naive Bayes as a function of σ and μ.*

From the depicted graph, and after having evaluated the number of remained features, we decided to choose $\sigma = 10$ and $\mu = 0.0015$ as optimal values, with a cross-validated accuracy of 0.89. With these values, from the original unigrams dictionary length of 4,617, we eliminate 3,696 words because of σ and 1,432 words because of μ. Some of the eliminated words were overlapping among the two thresholds, but some words could have been left if the condition of the thresholds were not an 'and'. Overall, the total dictionary length for the unigram case of the Multinomial naive Bayes model has become 424 after this parameter tuning operation. Hence, we will use 424 features for the Multinomial naive Bayes unigram case.

Bigrams. For the bigram case, we decided to test the following values:

σ**:** $[1, 2, 3, 4, 5, 6, 8, 10, 15, 20, 30, 50, 70, 100]$

μ**:** $[0, 0.0003, 0.001, 0.0014, 0.0016, 0.005, 0.01, 0.013, 0.015, 0.02]$

Please, note that the tested values are slightly different compared to the unigram case: the reason is that unigrams and bigrams dictionaries are very different in terms of frequencies of words from each other, and eliminate too many words would result in the risk of eliminate almost all the bigram words, leaving only the unigram ones. The resulting plot is shown in Figure 2.

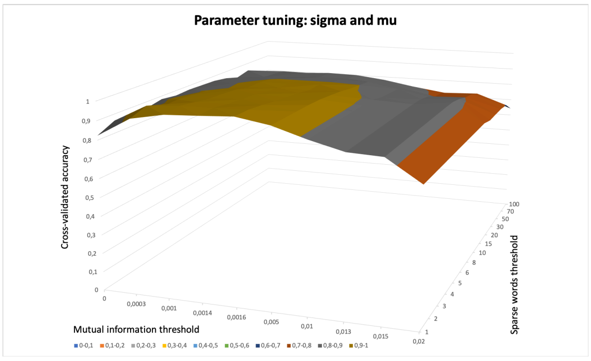

Figure 2: *Cross-validated accuracy for bigrams - Multinomial naive Bayes as a function of* σ *and* μ.

The optimal combination of parameter tuning for the bigrams case resulted to be $\sigma = 1$ and $\mu = 0.005$. With these settings, from the original length of the combined unigrams and bigrams dictionaries of 4,617 (only unigrams) + 38,631 (only bigrams) = 43,248 features, we don't eliminate any of the sparse words, but we eliminate 38,369 features due to the mutual information threshold. Overall, we will use 519 features for the bigram case, among which 262 are bigrams and 257 are unigrams.

6.2.2 Regularized Logistic Regression tuning

The parameter that has to be tuned in the Logistic Regression model is λ, that is, the penalty for the complexity of the model. Using the `scikit-learn` library, we have the possibility to tune the C parameter, which is defined as $C = 1/\lambda$ and, as explained, we will perform a cross-validation procedure.

Unigrams. Regarding the unigram case, for C we decided to test the following values:

C**:** $[0.0001, 0.001, 0.0088, 0.0259, 0.0759, 0.1, 0.2, 0.2222, 0.3, 0.4, 0.5, 0.6, 0.6510, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.6, 1.8, 1.9065, 2, 3, 4, 5, 5.5836, 6, 7, 8, 9, 10, 16.35, 25, 47.89, 50, 100, 120, 140.25, 160, 200, 220, 260, 300, 320, 360, 400, 410.76, 420, 460, 500, 520, 560, 600, 1000, 1202.97, 1500, 2000, 3000, 3523.07, 4000, 10317.82, 30217.24, 88495.57]$

The results in cross-validated accuracy are depicted in Figure 3, where we decided to plot λ on a logarithmic scale, in order to see the little fluctuation in accuracy that would have disappear if a normal scale was used instead. In this figure we also plot the number of remaining features as λ is changing: the more λ is decreasing, the more features we have in the model, because the penalty given for the complexity is decreasing with λ.

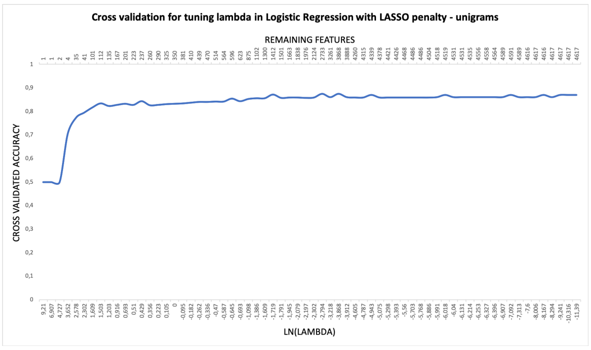

Figure 3: *Cross-validated accuracy for unigrams - Logistic Regression as a function of* $\log(\lambda)$.

By looking at this graph, we decided to set $C = 1.3$, which corresponds to $\lambda = 0.769$ and $\log(\lambda) = -0.262$, obtaining a reduction from 4617 features to only 439 features, but keeping a cross-validated accuracy of 0.84.

Bigrams. For the bigram case, for C we decided to test the following values:

C**:** $[0.0001, 0.001, 0.008, 0.025, 0.075, 0.1, 0.2, 0.222, 0.3, 0.4, 0.5, 0.6, 0.651, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.6, 1.8, 1.906, 2, 3, 4, 5, 5.583, 6, 7, 8, 9, 10, 16.352, 25, 47.891, 50, 100, 120, 140.256, 160, 200, 220, 260, 300, 320, 360, 400, 410.760, 420, 460, 500, 520, 560, 600, 1000, 1202.970, 1500, 2000, 3000, 3523.072, 4000, 10317.826, 30217.246, 88495.575]$

The results are visible in Figure 4.

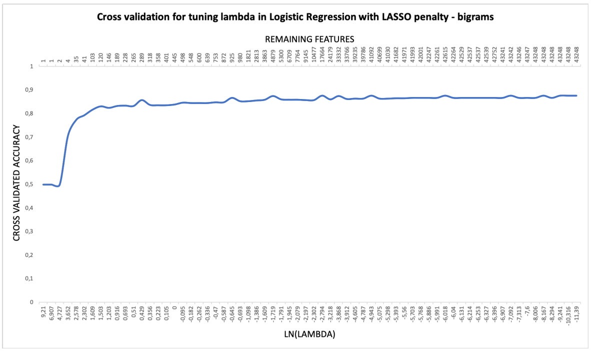

Figure 4: *Cross-validated accuracy for bigrams - Logistic Regression as a function of* $\log(\lambda)$.

From Figure 4 we decided to set the optimal C parameter for the bigram case to $C = 0.7$, which corresponds to $\lambda = 1.428$ and $\log(\lambda) = 0.356$. In this case, from a total of 43,248 features we have only 318 features left, obtaining a cross-validated accuracy of 0.837.

6.2.3 Classification Trees tuning

The parameter that has to be tuned in the Classification Tree classifier is α, that is, the complexity parameter used for Minimal Cost-Complexity Pruning. Basically, Minimal Cost-Complexity

Pruning recursively finds the node with the ‘weakest link’. The weakest link is characterized by an effective α, where the nodes with the smallest effective α are pruned first. To get an idea of what values of the effective α could be appropriate, `scikit-learn` provides a function called `cost_complexity_pruning_path` that returns the effective α and the corresponding total leaf impurities at each step of the pruning process. As α increases, a bigger part of the tree is pruned, which increases the total impurity of its leaves, but also is a form of overfitting prevention.

Unigrams. The `cost_complexity_pruning_path` function was used to obtain the following values of α to be tested:

α**:** $[0, 0.00154, 0.00155, 0.0020, 0.0023, 0.0025, 0.0027, 0.00281, 0.00286, 0.00291, 0.00292, 0.00293, 0.00294, 0.00296, 0.0030, 0.0034, 0.0038, 0.0039, 0.0044, 0.0046, 0.00470, 0.00477, 0.00480, 0.00486, 0.0049, 0.0050, 0.0051, 0.0052, 0.00540, 0.00541, 0.00554, 0.00559, 0.0057, 0.0059, 0.0063, 0.0065, 0.0066, 0.0068, 0.0070, 0.0078, 0.0098, 0.0099, 0.01, 0.0108, 0.0125, 0.0137, 0.0148, 0.086]$

Performing cross validation using these values, we have obtained the following graph:

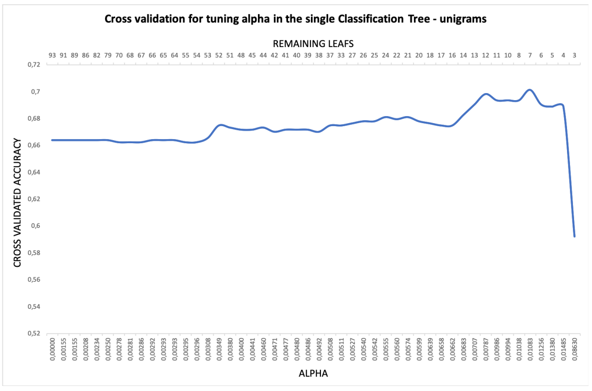

Figure 5: *Cross-validated accuracy for unigrams - single Classification Tree as a function of α.*

Where in the upper x-axis it is visible how many leaves the tree is composed by, as α is changing, hence, as how ‘big’ chunks of the tree are pruned.

Another interesting analysis would be to understand the trend of the total impurity of the leaves as a function of α, whose results are depicted in Figure 6.

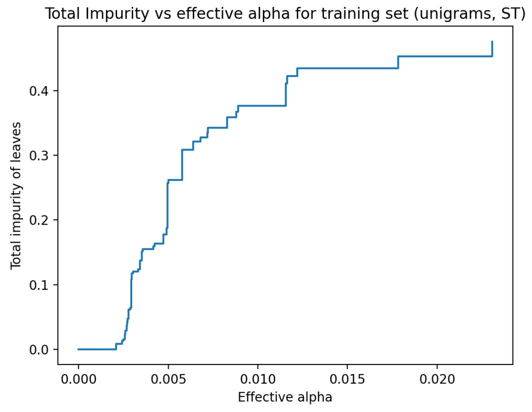

Figure 6: *Total impurity of the leaves of the Classification Tree - unigrams as a function of α.*

As predictable, the more α is increasing, the more the total impurity of the leaves increases.

By combining the results of both these analyses, we decided to set $\alpha = 0.01$ for the unigram case, which yields to a tree with only 8 leaf nodes, a depth of 6 and a cross-validated accuracy of 0.70.

Bigrams. For the bigram case we tested the following values for α:

α**:** $[0, 0.0020, 0.0023, 0.0024, 0.0026, 0.00281, 0.00286, 0.00291, 0.0029296875, 0.00296, 0.00304, 0.00305, 0.0036, 0.00375, 0.00377, 0.0039, 0.0043, 0.0046, 0.00480, 0.004861, 0.0049, 0.00504, 0.00507, 0.00540, 0.00541, 0.0055, 0.0056, 0.0057, 0.0058, 0.0059, 0.0063, 0.0066, 0.0067, 0.0071, 0.0078, 0.0079, 0.0090, 0.0094, 0.0095, 0.0098, 0.0099, 0.0103, 0.0108, 0.0125, 0.0137, 0.0148, 0.0862]$

The result of the cross-validation procedure is depicted in Figure 7.

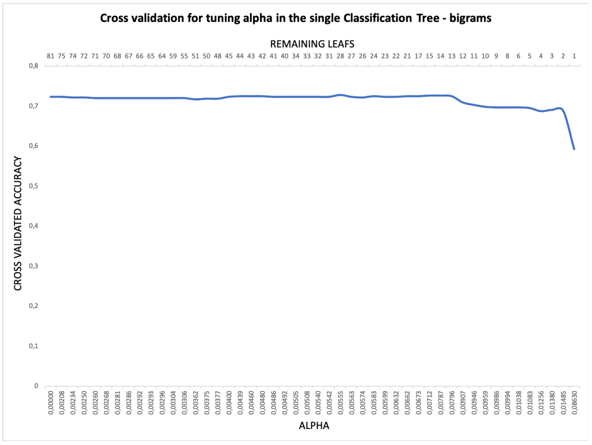

Figure 7: *Cross-validated accuracy for bigrams - single Classification Tree as a function of α.*

Even in this case, it is possible to perform the analysis of the total impurity of the leaves as a function of α, depicted in Figure 8.

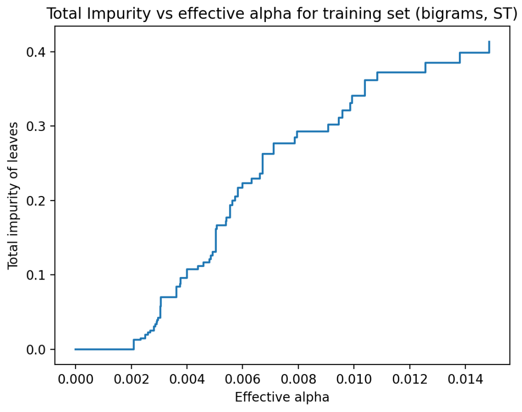

Figure 8: *Total impurity of the leaves of the Classification Tree - bigrams as a function of α.*

By combining the results of both the analyses, we obtained a value of $\alpha = 0.008$ which yields to a tree with 13 leaves, a depth of 7 and a cross-validated accuracy of 0.725.

6.2.4 Random Forest tuning

The Random Forest classifier has 2 main parameters that have to be tuned, namely, the number m of trees that the forest must be composed by (which in `scikit-learn` is called `n_estimators`), and the number of features $nfeat$ that has to be used when determine the best split on a node (which in `scikit-learn` is called `max_features`). Since we already performed an extensive analysis for the single Classification Tree model, we think that it is not necessary to perform the same detailed analysis even for the Random Forest classifier - which is an ensemble of single trees - hence, for both the unigram and bigram cases, rather than using the `cross_val_function` we performed the automatic cross validation procedure using the most common `GridSearchCV` function.

Unigrams. For the unigram case, we tested the following parameters:

m: $[30, 50, 70, 90, 100, 110, 120, 130]$

$nfeat$: $[\text{'}sqrt', \text{'}log2', 0.4, 0.8]$

Please, note that the $nfeat$ parameter can assume different type of values. When the value is 'sqrt' or 'log2' it means that each time a split has to be computed, $nfeat = \sqrt{total\_features}$ or $nfeat = \log_2(total\_features)$, respectively. When the value is a `float` is to be considered as a percentage of the total features that have to be sampled and used. Finally, when the value is `int`, it means that the value is the exact number of features that have to be used. Since in this last case the search space can be too large, we decided to omit the search for an integer number of features.

As an output from the `GridSearchCV` function, we obtain the best parameters for the unigram Random Forest classifier, that are $m = 130$ and $nfeat = \text{'}sqrt'$.

Bigrams. For the bigram case of the Random Forest classifier we decided to test the same m and $nfeat$ values as in the unigram case.

Even in this case, as an output from the `GridSearchCV` function, we obtain for the bigram case the same best parameters of the unigram Random Forest classifier, that are $m = 130$ and $nfeat = \text{'}sqrt'$.

6.3 Model results

6.3.1 Multinomial naive Bayes

Unigrams. First we trained a Multinomial naive Bayes model which only included unigrams, obtaining the confusion matrix shown below in Table 4.

		Predicted class	
		Deceptive	*Truthful*
Actual class	*Deceptive*	66	14
	Truthful	10	70

Table 4: *Confusion matrix regarding unigrams for the Multinomial naive Bayes model.*

From this matrix we got the results shown in Table 5. The three average parameters have the same value. The accuracy we found here is 85%, which is a pretty big improvement from simply predicting the majority class. This could indicate that Multinomial naive Bayes is a useful model for our problem.

Unigrams	**0**	**1**	**wAvg**
Precision	0.87	0.83	0.85
Recall	0.82	0.88	0.85
f1-score	0.85	0.85	0.85

Table 5: *Precision, recall and f1-score for unigrams in the Multinomial naive Bayes model, divided by class and the weighted average value (wAvg).*

Bigrams. When also including bigrams in our Multinomial naive Bayes model we expected to see an improvement in performance as the model has more data to predict on. Moreover, in the context of reviews, combinations of words could drastically change the meaning of the review (e.g. 'not fun', 'not bad'). Table 6 shows the confusion matrix for bigrams.

		Predicted class	
		Deceptive	*Truthful*
Actual class	*Deceptive*	63	17
	Truthful	7	73

Table 6: *Confusion matrix regarding bigrams for the Multinomial naive Bayes model.*

When analyzing the results as shown in Table 7, however, we notice barely any difference between the unigram and bigram results. Accuracy is again 85% and there is only a slight improvement in precision, which is small enough to be insignificant and this is most likely noise in the data. This could indicate that either the addition of bigrams is unnecessary or not significant specifically for this problem and/or dataset.

Bigrams	**0**	**1**	**wAvg**
Precision	0.9	0.81	0.86
Recall	0.79	0.91	0.85
f1-score	0.84	0.86	0.85

Table 7: *Precision, recall and f1-score for bigrams in the Multinomial naive Bayes model, divided by class and the weighted average value (wAvg).*

6.3.2 Regularized Logistic Regression

Unigrams. Table 8 shows the confusion matrix when using only unigrams.

		Predicted class	
		Deceptive	*Truthful*
Actual class	*Deceptive*	63	17
	Truthful	10	70

Table 8: *Confusion matrix regarding unigrams for the Regularized Logistic Regression model.*

The confusion matrix for unigrams shows that a big portion of the 160 reviews in the test set has been classified in the correct class (63 TN and 70 TP), in fact the accuracy here is 83.12%, quite high as performance indicator but still lower than the Multinomial naive Bayes model. The values for precision, recall and f1-score for each class are shown in Table 9. Average recall is higher than average precision, which makes sense since the FP are almost double the FN. Overall, the performance of this model seemed to highly improve the trivial predicting the majority class model accuracy.

Unigrams	**0**	**1**	**wAvg**
Precision	0.86	0.79	0.82
Recall	0.80	0.88	0.84
f1-score	0.83	0.83	0.83

Table 9: *Precision, recall and f1-score for unigrams in the Regularized Logistic Regression model, divided by class and the weighted average value (wAvg).*

Bigrams. When we also included bigrams, the confusion matrix becomes as shown in Table 10.

		Predicted class	
		Deceptive	*Truthful*
Actual class	*Deceptive*	62	18
	Truthful	11	69

Table 10: *Confusion matrix regarding bigrams for the Regularized Logistic Regression model.*

Compared to unigrams, less deceptive reviews have been classified as such and also less truthful reviews have been identified correctly. The overall accuracy becomes 81.87% which is lower than using only the unigrams. We think that the loss in accuracy is due to the higher λ that we choose for the bigram version, resulting in more features that have been erased by the model. The values for precision, recall and f1-score for each class are shown in Table 11. In this case, the average precision is the same as the average recall.

Bigrams	**0**	**1**	**wAvg**
Precision	0.85	0.78	0.82
Recall	0.77	0.86	0.82
f1-score	0.81	0.83	0.82

Table 11: *Precision, recall and f1-score for bigrams in the Regularized Logistic Regression model, divided by class and the weighted average value (wAvg).*

The comparison of unigrams and bigrams of this model shows that in general when also including bigrams the accuracy has slightly decreased, thus the unigram model seems to be more performing than the bigram, but this can be due to a different hyperparameter we chose for these models.

6.3.3 Classification Trees

Since the depth of the obtained Classification Trees for unigrams and bigrams is not much, we decided to include the trees in the Appendix of this paper, Section 8.

Unigrams. When only using unigrams to train and test our model, we obtained the confusion matrix of Table 12.

		Predicted class	
		Deceptive	*Truthful*
Actual class	*Deceptive*	46	34
	Truthful	21	59

Table 12: *Confusion matrix regarding unigrams for the Classification Tree model.*

Then the results as shown in Table 13 were retrieved. Average precision and recall have the same value, due to the high difference of all values in the confusion matrix that balances between class 0 and 1. Accuracy is 65.62%, already improved when compared to predicting the majority class, which means using this classifier does improve the outcome as compared to random guessing. However, with this accuracy the model performs relatively poorly when compared to its competitors.

Unigrams	**0**	**1**	**wAvg**
Precision	0.69	0.63	0.66
Recall	0.57	0.74	0.66
f1-score	0.63	0.68	0.65

Table 13: *Precision, recall and f1-score for unigrams in the Classification Tree model, divided by class and the weighted average value (wAvg).*

Bigrams. When also including bigrams to our model, the confusion matrix becomes the one shown in Table 14.

		Predicted class	
		Deceptive	*Truthful*
Actual class	*Deceptive*	51	29
	Truthful	22	58

Table 14: *Confusion matrix regarding bigrams for the Classification Tree model.*

And we see the results as shown in Table 15. The averager values in the table all have the same value. Accuracy improved from 65.62% using only unigrams to 68.12% when also including bigrams. This means including bigrams does improve our model even though its only slight improvement.

Bigrams	**0**	**1**	**wAvg**
Precision	0.70	0.67	0.68
Recall	0.64	0.72	0.68
f1-score	0.67	0.69	0.68

Table 15: *Precision, recall and f1-score for bigrams in the Classification Tree model, divided by class and the weighted average value (wAvg).*

6.3.4 Random Forests

Unigrams. When the model only considers unigrams, the resulting confusion matrix is shown in Table 16.

		Predicted class	
		Deceptive	*Truthful*
Actual class	*Deceptive*	67	13
	Truthful	15	65

Table 16: *Confusion matrix regarding unigrams for the Random Forest model.*

An high proportion of reviews has been correctly classified, with an accuracy of 82.5%, a big improvement compared to predicting the majority class. The values for precision, recall and f1-score for each class are shown in Table 17. Precision average value is slightly higher than recall and their singular values for class 0 and 1 are quite close to each other, logical considering that FP and FN are very close values.

Unigrams	**0**	**1**	**wAvg**
Precision	0.82	0.83	0.83
Recall	0.84	0.81	0.82
f1-score	0.83	0.82	0.82

Table 17: *Precision, recall and f1-score for unigrams in the Random Forest model, divided by class and the weighted average value (wAvg).*

Bigrams. When also using bigrams the model confusion matrix is shown in Table 18.

		Predicted class	
		Deceptive	*Truthful*
Actual class	*Deceptive*	56	24
	Truthful	6	74

Table 18: *Confusion matrix regarding bigrams for the Random Forest model.*

Deceptive reviews have now been correctly classified less than when using only unigrams, while for truthful reviews the opposite happened since more were classified as such. This translates in an accuracy of 81.25%, which is slightly lower than before. The values for precision, recall and f1-score for each class are shown in Table 19. Here precision is a bit higher than recall then it was in all the other models (2% more).

Bigrams	**0**	**1**	**wAvg**
Precision	0.9	0.76	0.83
Recall	0.7	0.93	0.81
f1-score	0.79	0.83	0.81

Table 19: *Precision, recall and f1-score for bigrams in the Random Forest model, divided by class and the weighted average value (wAvg).*

It can be seen that for the Random Forest model using only unigrams lead to a higher accuracy and a more performing result, which can also be seen from the weighted average values in Tables 17 and 19, where both recall and f1-score are higher in the unigram model.

The lower accuracy of bigrams compared to unigrams in the Random Forest model is due to its underlying construction: the model always chooses the best split on $\sqrt{nfeat}$, where for unigrams is $nfeat = 4{,}617$ and for bigrams $nfeat = 38{,}631$. In the bigram model, though, both unigrams and bigrams are present, thus when calculating $\sqrt{nfeat} = \sqrt{43,248}$ the selected features are 208 of which most of them are likely to be bigrams (because of the difference in number between unigrams and bigrams). As a result of which, the model is using almost only the bigram features to calculate the best split, as it is confirmed by the bigram tree in Appendix, Section 8. This could lead to a lower information usage compared to having both unigrams ans bigrams.

6.4 Statistical significance

The analyses made in the previous Section, 6.3, showed the models are barely different from each other, apart from the single Classification Tree, which seems to perform poorly. To better compare the model performances, we decided to plot the Receiver Operating Characteristic (ROC) curve and calculate the corresponding Area Under the Curve (AUC).

6.4.1 ROC curve and AUC

The ROC curve is a graphical representation of a classification model performance using on x-axis the True Positive Rate:

$$TPR = \frac{TP}{TP + FN} \tag{9}$$

And on y-axis the False Positive Rate:

$$FPR = \frac{FP}{FP + TN} \tag{10}$$

Please note that while TPR is exactly the same as recall, FPR does not correspond to precision. While precision measures the probability of a sample classified as positive to actually be positive, the FPR measures the ratio of FP within the negative samples.

When training a model, a basic assumption is that it will predict in such a way that it minimizes the prediction error. The way models are doing this minimization is to calculate a likelihood that the new case belongs to a certain class. For a binary problem, this translates in having two percentage values that represent the certainty that case belongs to each class. Hence, to minimize the error, the model predicts the class which has this certainty greater than 50%. If we were, for any reason, transforming this certainty value into a threshold that we can tune, we can collect for each new threshold the values of FPR and TPR from the model's confusion matrix and we can plot the ROC curve of that model.

These threshold changes would lead to three different strategies.

A threshold lower than 50% means having a huge amount of values classified as 1, thus a very high value of FP since only a fraction of them would likely actually be a class 1 (TP). Having a high value of FP means that recall is maximised.

On the other hand, a threshold higher than 50% means having a huge amount of values classified as 0, thus a very high amount of FN because of the same reason just explained. Having a high value of FN means to maximize precision.

When the threshold is 50%, accuracy is maximised instead, that is minimize the error.

The more the curve is close to the y-axis, the better the model is. This has a simple explanation: a threshold very close to the y-axis has a FPR close to 0, independently than its value of TPR. A threshold with also high TPR is then the best threshold to use in the model. Thresholds with TPR close to 1 and FPR close to 0 are considered to be the best.

The AUC is then the two-dimensional area under the curve, calculated between each curve and the x-axis until FPR = 1. Generally, a higher AUC corresponds to a better model, since it is a representation of how close the curve is to the y-axis.

Figures 9 and 10 show respectively the ROC and AUC for unigrams and bigrams for each of the four models. ST stand for the Classification Tree model, LR Regularized Logistic Regression, RF Random Forest and MNB Multinomial naive Bayes.

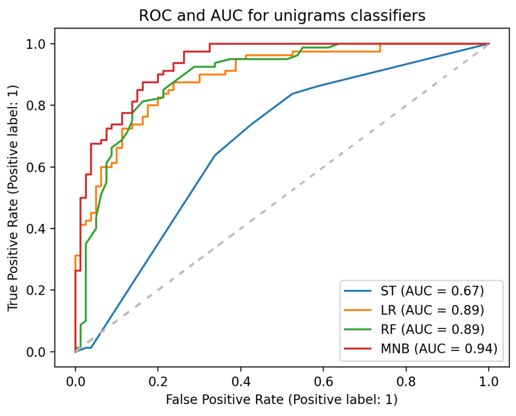

Figure 9: *ROC curve and AUC for unigrams.*

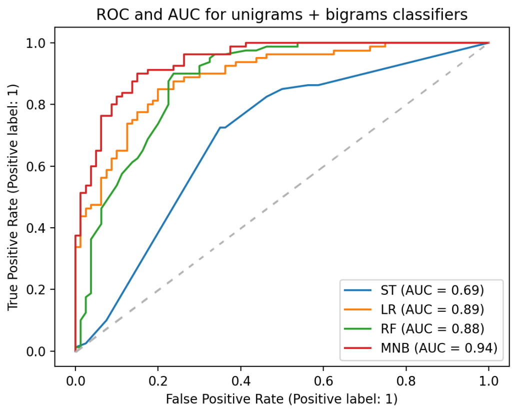

Figure 10: *ROC curve and AUC for bigrams.*

Both sets of curves show that Multinomial naive Bayes is the model with highest AUC, thus the more precise model, meaning it would most likely classify data correctly. It is interesting to notice that Logistic Regression and Random Forest lead to the same area for unigrams and almost the same (0.89 for LR and 0.88 for RF) for bigrams. For the Random Forest model, the difference in AUC in unigrams and bigrams has the same explanation given in Section 6.3.4, which is due to the difference in the number of features of the model with unigrams and the model with both unigrams and bigrams. For sure it can be seen how the Classification Tree model is the worst in classifying data.

To understand if these differences are actually statistically significant, we performed a statistical test.

6.4.2 Statistical test

For establishing whether there is actually difference between the models, we performed a McNemar test, a statistical test that can determine if two related categories have significantly different values.

In this specific test, the H_0 hypothesis states that the two models are similar, hence they do not give statistically significant differences in accuracy, while H_1 states that the accuracies are different. Rejecting H_0 means that the models have statistically significant differences in accuracy, thus H_1 is true. On the other hand, not rejecting H_0 means there is no statistical evidence to state that the models are different, but neither that they are equal.

Before showing the test results a premise has to be done: we have decided to compare only unigrams with unigrams and bigrams with bigrams of different models, without mixing them (i.e. no unigrams vs bigrams of different models) because during the tests we discovered that there is no statistical difference between unigrams and bigrams within the same model, thus, if we had tested also a mix of models, we would have obtained no statistical differences. As an example, if A is unigram of LR, B bigram of LR and C unigram of RF, if testing A and B gives as result that there is no statistical evidence they are different, testing A vs C is the same of testing B vs C, thus avoid one of them for not wasting time.

First we want to understand the difference in performance between the Multinomial naive Bayes and Logistic Regression, thus by setting the probability of type one errors $\alpha = 5\%$, we obtain Table 20 which shows χ^2 and p-value for each model comparison.

	χ^2	**p-value**
MNB vs LR unigrams	0.16	0.689
MNB vs LR bigrams	0.59	0.441

Table 20: *Values for χ^2 and p-value for both unigrams and bigrams obtained in output from the McNemar statistics test comparing Multinomial naive Bayes and Logistic Regression.*

In general, if the p-value of a test is less than α, we can reject H_0 and conclude that there is statistical significance between the two values of accuracy, i.e. the difference between false negatives and false positives is too important and thus not random.

In this first comparison we see that indeed the p-value is way higher than α, both for unigrams and bigrams, thus there is no statistical evidence to say that the models have different accuracy.

Subsequently, we wanted to see if the Random Forest model was able to have a better performance than the linear classifiers, which are Multinomial naive Bayes and Logistic Regression. Setting α=5% again, the results are shown in Table 21.

	χ^2	p-value
RF vs LR unigrams	0	1
RF vs MNB unigrams	0.346	0.556
RF vs LR bigrams	0	1
RF vs MNB bigrams	0.892	0.344

Table 21: *Values for χ^2 and p-value for both unigrams and bigrams obtained in output from the McNemar statistics test comparing Random Forests to both Multinomial naive Bayes and Logistic Regression.*

Also in these tests there is no statistical evidence to state that the Random Forest model overcomes in performance the linear classifiers.

Finally, we wanted to see if the performance of each model improved when using the model with unigrams and bigrams, compared to when only using unigrams. With α=5% again, the comparison has been done in Table 22.

	χ^2	p-value
ST unigrams vs bigrams	1.125	0.288
LR unigrams vs bigrams	0.125	0.723
RF unigrams vs bigrams	0.041	0.838
MNB unigrams vs bigrams	0.1	0.751

Table 22: *Values for χ^2 and p-value obtained in output from the McNemar statistics test comparing each model's unigrams with bigrams.*

The comparison between unigrams and bigrams in each model showed that there exists no statistical evidence stating they are different, which is the reason why in the beginning of this section we have stated that we only compared unigrams with unigrams and bigrams with bigrams.

For the sake of completeness, we decided to report also the test between unigrams of each model with the others, as seen in Table 23 and between bigrams of each model with the others, as in Table 24.

	χ^2	p-value
ST unigrams vs LR unigrams	15.847	0.000069
ST unigrams vs RF unigrams	14.382	0.0001
ST unigrams vs MNB unigrams	15.789	0.00007
LR unigrams vs RF unigrams	0	1
LR unigrams vs MNB unigrams	0.16	0.689
RF unigrams vs MNB unigrams	0.346	0.556

Table 23: *Values for χ^2 and p-value obtained in output from the McNemar statistics test comparing unigrams of every model.*

	χ^2	p-value
ST bigrams vs LR bigrams	10.02	0.00015
ST bigrams vs RF bigrams	9.756	0.001
ST bigrams vs MNB bigrams	12.754	0.0003
LR bigrams vs RF bigrams	0	1
LR bigrams vs MNB bigrams	0.59	0.441
RF bigrams vs MNB bigrams	0.892	0.344

Table 24: *Values for χ^2 and p-value obtained in output from the McNemar statistics test comparing bigrams of every model.*

The only interesting result is contained in Table 23 and 24, where there is statistical evidence only to state that the Classification Tree is worse than the other models. This result was already clear by looking at the AUC: the Classification Tree had an area way lower than the other models.

However, even if from the statistical tests we did not have enough statistical evidence to state that models are different, except for the Classification Tree, the AUC of Multinomial naive Bayes and its accuracy were higher than the other models, thus we can conclude that it is indeed the model to choose for classifying deceptive and truthful hotel reviews.

6.5 Terms analysis

At the end of the analysis, the five most important features pointing towards a deceptive or a truthful review have been collected and shown in Tables 25, 26, 27 and 28, from the Multinomial naive Bayes model because it resulted to be the model with the highest AUC value. In order to retrieve this information, we looked at the coefficient attributes of the Multinomial naive Bayes model, which are a re-parameterization of the naive Bayes model as a Linear Classifier. For a binary classification problem, the coefficients are basically the logarithm of the estimated probability of a feature given the positive class. This means that higher coefficient values mean more important features for the positive class. Thus, by sorting in increasing order the coefficient of each feature (word), the top 5 words of this list are the top 5 words pointing towards a deceptive review, while the last 5 (in inverse order) are the top 5 pointing towards a truthful review.

Unigrams	**Deceptive**	**Coefficient**
1	celebr	-8.75
2	cigarett	-8.75
3	musti	-8.75
4	settl	-8.75
5	children	-8.35

Table 25: *Top 5 unigrams pointing towards a deceptive review, with their score.*

Unigrams	**Truthful**	**Coefficient**
1	room	-2.77
2	hotel	-2.9
3	stay	-3.41
4	servic	-4.16
5	call	-4.16

Table 26: *Top 5 unigrams pointing towards a truthful review, with their score.*

Bigrams	**Deceptive**	**Coefficient**
1	across street	-8.83
2	bathroom door	-8.83
3	star hotel	-8.83
4	amount time	-8.43
5	area recommended	-8.43

Table 27: *Top 5 bigrams pointing towards a deceptive review, with their score.*

Bigrams	**Truthful**	**Coefficient**
1	smell cigarett	-2.85
2	talbott hotel	-2.97
3	went room	-3.48
4	servic food	-4.24
5	reserv could	-4.24

Table 28: *Top 5 bigrams pointing towards a truthful review, with their score.*

It appears that the features pointing towards a truthful review contain words that are related to the room, the hotel or the service, denoting that the person who wrote the review actually stayed at the facility and was able to notice particular things that actually experienced.

7 Conclusion

To conclude there are some observations to be discussed. First off, our results show no statistically significant differences when comparing our generative linear model (Multinomial naive Bayes) to the discriminative linear model (Regularized Logistic Regression). When comparing the unigram-trained algorithms and the bigram-trained algorithms, both our significant tests failed to reject the null hypothesis. Secondly, the Random Forest model did not seem to improve the performance of our linear classifiers. When comparing the Random Forest both to the Regularized Logistic Regression and the Multinomial naive Bayes model our tests show no significant differences. Adding bigrams to the training data once again did not change the outcome, meaning Random Forest and linear models perform similarly for all tests we ran. Thirdly, we noticed nothing significant when testing for differences between the unigram-trained and bigram-trained models. As shown in the statistical test Section 6.4.2, only the Classification Tree model performed differently both when trained on unigrams only and when a combination of unigrams and bigrams was used. This is in line with the expectations as a single Classification Tree is a relatively simple model when compared to Multinomial naive Bayes, Regularized Logistic Regression and Random Forests. Moreover, a single Classification Tree is relatively sensitive to small perturbations in the data and, while pruning, the tree can (partially) prevent overfitting but it could also reduce the models' accuracy. Comparing the other three models did not lead to any significant differences. The Multinomial naive Bayes model ended up being the best performing model with a weighted accuracy of 85%, for both unigrams and bigrams, although the statistical tests did not find any statistical evidence to state that it is better. For a more definitive answer more research would have to be performed, preferably with a bigger dataset.

Lastly we can compare our results to those found by Ott et al. [25][3], as they wrote two papers using the same dataset. The most reasonable comparison would be between our results and the results of their paper analyzing negative hotel reviews [3], as it was performed on the exact same task on the same dataset. Instead of using the models explained in this paper however, Ott et al. opted to use a Support Vector Machine (SVM) model. Their results show an accuracy or 86%, slightly higher than our best performing model, Multinomial naive Bayes. This difference in accuracy could be caused either by differences in parameter tuning, or by some differences between the models themselves. Comparing to the other paper by Ott et al. [25], could also be interesting as in this paper a naive Bayes model was used, meaning we can compare the same model directly to itself. However, it is to be noted that this paper distinguished positive truthful and deceptive reviews whereas our paper focused on negative truthful and deceptive reviews. The results obtained by Ott et al. were slightly higher than those found in this paper as they got an accuracy of 88.4% for unigrams and 88.9% for bigrams. This difference could indicate that their parameter tuning was better, however such a small difference could also be explained by the fact they analyzed positive instead of negative reviews for their research, thus the data is different even though it comes from the same dataset.

References

[1] T. Lappas, G. Sabnis, and G. Valkanas, "The impact of fake reviews on online visibility: A vulnerability assessment of the hotel industry," *Information Systems Research*, vol. 27, no. 4, pp. 940–961, 2016.

[2] N. Jindal and B. Liu, "Opinion spam and analysis," in *Proceedings of the 2008 international conference on web search and data mining*, pp. 219–230, 2008.

[3] M. Ott, C. Cardie, and J. T. Hancock, "Negative deceptive opinion spam," in *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: human language technologies*, pp. 497–501, 2013.

[4] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes, "Multinomial naive bayes for text categorization revisited," in *Australasian Joint Conference on Artificial Intelligence*, pp. 488–499, Springer, 2004.

[5] A. Juan and H. Ney, "Reversing and smoothing the multinomial naive bayes text classifier.," in *PRIS*, pp. 200–212, Citeseer, 2002.

[6] S. Swaminathan, "Logistic regression, detailed overview." https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc.

[7] S.-I. Lee, H. Lee, P. Abbeel, and A. Y. Ng, "Efficient l˜ 1 regularized logistic regression," in *Aaai*, vol. 6, pp. 401–408, 2006.

[8] L. Rokach and O. Maimon, "Classification trees," in *Data mining and knowledge discovery handbook*, pp. 149–174, Springer, 2009.

[9] L. Breiman, "Random forests," *UC Berkeley TR567*, 1999.

[10] D. Di Grandi, J. Hartkamp, and A. Hartog, "Python code for the second assignment of the data mining course." `https://github.com/danieledigrandi/DataMining_2021/tree/main/Assignment_2`, 2021.

[11] K.-N. Lau, K.-h. Lee, and Y. Ho, "Kin-nam lau, kam-hon lee and ying ho, "text mining for the hotel industry," cornell hotel and restaurant administration quarterly, (46, 3) 2005, 344-362.," *Cornell Hotel and Restaurant Administration Quarterly - CORNELL HOTEL RESTAUR ADMIN Q*, vol. 46, pp. 344–362, 08 2005.

[12] N. Öztürk and S. Ayvaz, "Sentiment analysis on twitter: A text mining approach to the syrian refugee crisis," *Telematics and Informatics*, vol. 35, no. 1, pp. 136–147, 2018.

[13] R. Mihalcea and C. Strapparava, "The lie detector: Explorations in the automatic recognition of deceptive language," in *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, ACLShort '09, (USA), p. 309–312, Association for Computational Linguistics, 2009.

[14] F. Li, M. Huang, Y. Yang, and X. Zhu, "Learning to identify review spam.," pp. 2488–2493, 01 2011.

[15] M. Ott, C. Cardie, and J. T. Hancock, "Negative deceptive opinion spam," in *NAACL*, 2013.

[16] K. Crowston, "Amazon mechanical turk: A research tool for organizations and information systems scholars," in *Shaping the Future of ICT Research. Methods and Approaches* (A. Bhattacherjee and B. Fitzgerald, eds.), (Berlin, Heidelberg), pp. 210–221, Springer Berlin Heidelberg, 2012.

[17] D. Vidanagama, T. Silva, and A. Karunananda, "Deceptive consumer review detection: a survey," *Artificial Intelligence Review*, vol. 53, pp. 1323–1352, 02 2020.

[18] C. Harris, "Dirty deeds done dirt cheap: A darker side to crowdsourcing," pp. 1314–1317, 10 2011.

[19] Z. Hai, P. Zhao, P. Cheng, P. Yang, X.-L. Li, and G. Li, "Deceptive review spam detection via exploiting task relatedness and unlabeled data," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, (Austin, Texas), pp. 1817–1826, Association for Computational Linguistics, Nov. 2016.

[20] A. Badresiya, “Performance analysis of supervised techniques for review spam detection,” 2014.

[21] S. Banerjee, A. Y. K. Chua, and J.-J. Kim, “Using supervised learning to classify authentic and fake online reviews,” in *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*, IMCOM ’15, (New York, NY, USA), Association for Computing Machinery, 2015.

[22] J. K. Rout, A. Dalmia, K.-K. R. Choo, S. Bakshi, and S. K. Jena, “Revisiting semi-supervised learning for online deceptive review detection,” *IEEE Access*, vol. 5, pp. 1319–1327, 2017.

[23] M. Daiyan, D. S. K. Tiwari, and M. A. Alam, “Mining product reviews for spam detection using supervised technique,” 2014.

[24] T. Pranckevicius and V. Marcinkevičius, “Comparison of naive bayes, random forest, decision tree, support vector machines, and logistic regression classifiers for text reviews classification,” *Baltic Journal of Modern Computing*, vol. 5, 01 2017.

[25] M. Ott, Y. Choi, C. Cardie, and J. T. Hancock, “Finding deceptive opinion spam by any stretch of the imagination,” *arXiv preprint arXiv:1107.4557*, 2011.

[26] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* ” O’Reilly Media, Inc.”, 2009.

[27] I. Braga, M. Monard, and E. Matsubara, “Combining unigrams and bigrams in semi-supervised text classification,” in *Proceedings of Progress in Artificial Intelligence, 14th Portuguese Conference on Artificial Intelligence (EPIA 2009), Aveiro*, pp. 489–500, Citeseer, 2009.

[28] D. Jurafsky and J. Martin, “Naibe bayes and sentiment classification,” in *Speech and Language Processing*, 2021.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[30] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media, 2009.

8 Appendix

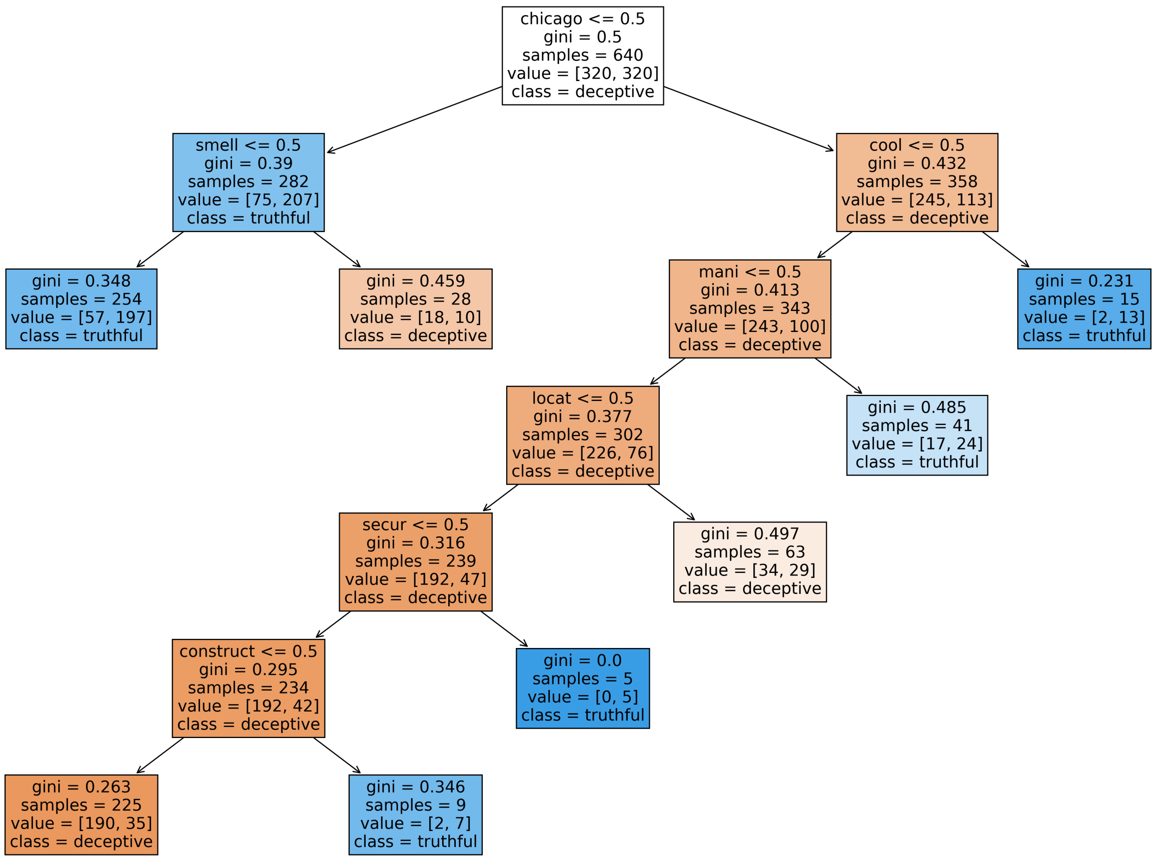

Figure 11: *Decision tree obtained for unigrams.*

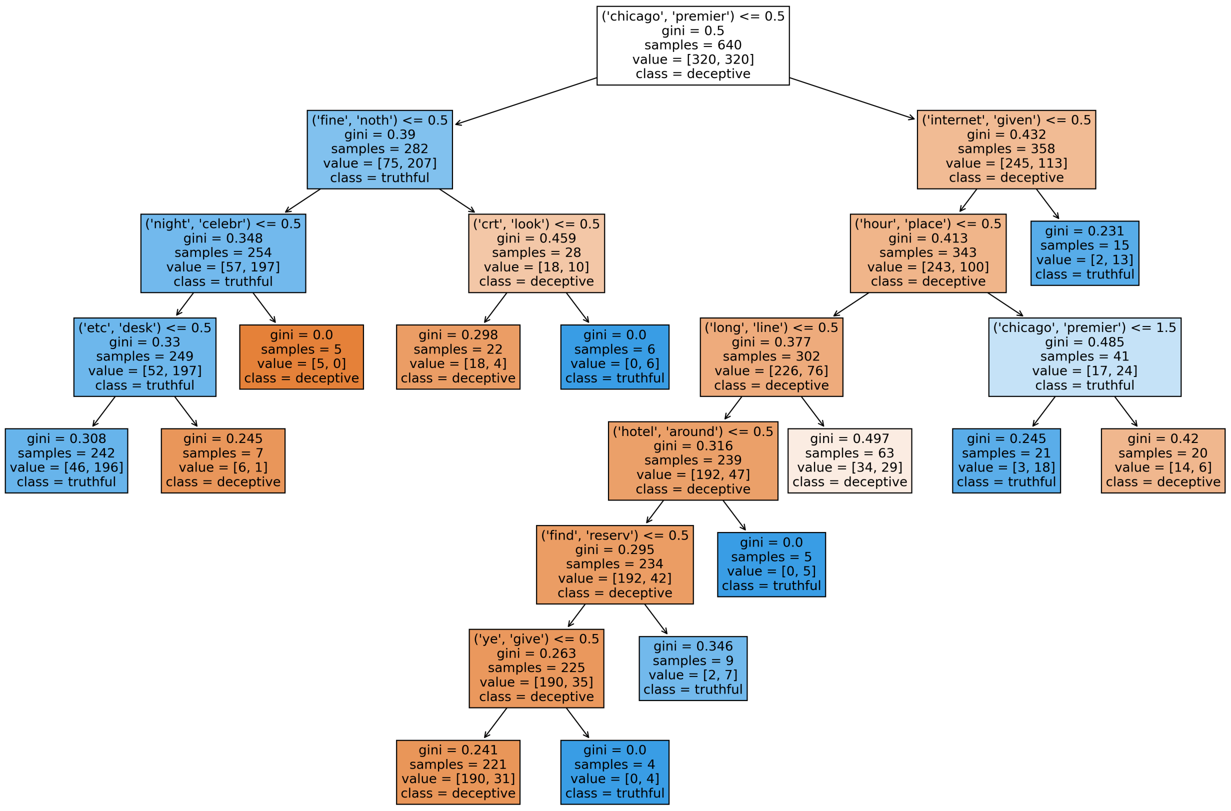
('chicago', 'premier') <= 0.5 gini = 0.5 samples = 640 value = [320, 320] class = deceptive
('fine', 'noth') <= 0.5 gini = 0.39 samples = 282 value = [75, 207] class = truthful
('internet', 'given') <= 0.5 gini = 0.432 samples = 358 value = [245, 113] class = deceptive
('night', 'celebr') <= 0.5 gini = 0.348 samples = 254 value = [57, 197] class = truthful
('crt', 'look') <= 0.5 gini = 0.459 samples = 28 value = [18, 10] class = deceptive
('hour', 'place') <= 0.5 gini = 0.413 samples = 343 value = [243, 100] class = deceptive
gini = 0.231 samples = 15 value = [2, 13] class = truthful
('etc', 'desk') <= 0.5 gini = 0.33 samples = 249 value = [52, 197] class = truthful
gini = 0.0 samples = 5 value = [5, 0] class = deceptive
gini = 0.298 samples = 22 value = [18, 4] class = deceptive
gini = 0.0 samples = 6 value = [0, 6] class = truthful
('long', 'line') <= 0.5 gini = 0.377 samples = 302 value = [226, 76] class = deceptive
('chicago', 'premier') <= 1.5 gini = 0.485 samples = 41 value = [17, 24] class = truthful
gini = 0.308 samples = 242 value = [46, 196] class = truthful
gini = 0.245 samples = 7 value = [6, 1] class = deceptive
('hotel', 'around') <= 0.5 gini = 0.316 samples = 239 value = [192, 47] class = deceptive
gini = 0.497 samples = 63 value = [34, 29] class = deceptive
gini = 0.245 samples = 21 value = [3, 18] class = truthful
gini = 0.42 samples = 20 value = [14, 6] class = deceptive
('find', 'reserv') <= 0.5 gini = 0.295 samples = 234 value = [192, 42] class = deceptive
gini = 0.0 samples = 5 value = [0, 5] class = truthful
('ye', 'give') <= 0.5 gini = 0.263 samples = 225 value = [190, 35] class = deceptive
gini = 0.346 samples = 9 value = [2, 7] class = truthful
gini = 0.241 samples = 221 value = [190, 31] class = deceptive
gini = 0.0 samples = 4 value = [0, 4] class = truthful

Figure 12: *Decision tree obtained for bigrams.*