# Classification of Tweets using Supervised and Semisupervised Learning

Achin Jain, Kuk Jang

## I. Introduction

The goal of this project is to classify the given tweets into 2 categories, namely "happy" and "sad." Two data sets are provided, one of which is labeled and the other unlabeled. Each set consisting of 4500 observations includes the top 10000 words observed in the collection, raw tweets, word counts for each tweet, downsized images linked to the URL in the tweets, and some CNN features and color mappings related to images.

It is also known that the labeled data set was hand-labled. Superimposition of tweets, corresponding images and the labels indicates the data is extremely noisy. One illustrative example is shown in Fig. 1. In this report, we investigate different feature extraction techniques and classification algorithms to predict whether a given tweet should be labeled as "happy" or "sad." Note that the problem is different from classifying whether the tweet-er was actually "happy" or "sad."

The report is organized as follows. In Sec. II we explain standard approaches for feature extraction and feature manipulation for text classification. This includes dimensionality reduction by removing missing features, Principal Component Analysis (PCA) or stemming, choosing topwords on the basis of information gain, and addition of bigrams for making features more informative. Sec. III describes different supervised and unsupervised classification algorithms based on Logistic Regression (LR), Support Vector Machines (SVMs), Naive Bayes (NB), Mixture Models (MM) and Ensemble Methods and the results obtained from them. We summarize the results and conclude in Sec. IV.

## II. Feature Extraction

In this report, we will only discuss using word counts as the training features. We have tried using image features as well, but the performance was not comparable to using bag-of-words alone.

We use the following notation throughout the paper. The bag-of-words features are represented by $n \times p$ matrices $\mathsf{X}_l$ and $\mathsf{X}_u$, where $n$ is 4500 and $p$ 10000 in our case. In this notation, the word count for the word $w_t$ in the observation $i$ is given by $\mathsf{X}(i,t)$. The subscripts $l$ and $u$ indicate labeled and unlabeled sets, respectively. The labels for these sets are given by $n \times 1$ column vectors $\mathsf{Y}_l$ and $\mathsf{Y}_u$ with entries 1 and 0 for "happy" and "sad" tweets, respectively.

Feature reduction is very important for some algorithms (unlike Naive Bayes). In the following sections, we present different feature reduction or feature manipulation steps we tried before training our classifiers.

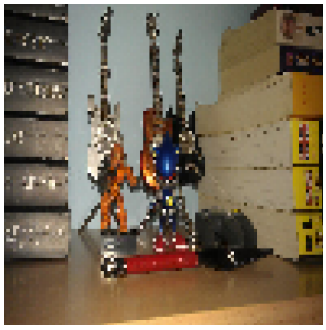

Fig. 1: One of the many examples of mislabeling. The raw image with a "sad" label is shown above. The corresponding tweet reads: "*hell yeah #sonic #halo #n64 #naruto #guitars / injrcmhb*".

### A. Missing Features

In both $\mathsf{X}_l$ and $\mathsf{X}_u$, we observe columns with all zero's. This means that none of the corresponding top words appear in any tweet in the respective sets. It is reasonable to say that the features which do not appear even once are useless for training. More specifically, 4155 columns in $\mathsf{X}_l$ have all zero entries. We considered removing these missing features from $\mathsf{X}_l$ while doing supervised learning. Similarly, 4106 columns in $\mathsf{X}_u$ also have all zero entries, of which 2353 columns are zero in both labeled and unlabeled sets. As and when indicated in Sec. III, we will also remove these 2353 top words while doing semi-supervised learning.

### B. Mutual Information

A systematic way of measuring importance features is using mutual information (MI) [5], [8]. We calculate the information $I(w_t)$ measured by a word $w_t$ (in information theoretic sense) as

$$I(w_t) = \sum_{i=1}^{n} \sum_{j \in \{0,1\}} P(\mathsf{X}(i,t) > 0, \mathsf{Y}_i = j) \times \log_2 \left( \frac{P(\mathsf{X}(i,t) > 0, \mathsf{Y}_i = j)}{P(\mathsf{X}(i,t) > 0) P(\mathsf{Y}_i = j)} \right). \quad (1)$$

Since we also require labels to calculate the metric in (1), this procedure applies only to the labeled data set. We can sort $I$ and choose the features corresponding to the $k$ largest values; the value of $k$ is chosen by cross validation. We sorted the information gain for all the features in a decreasing order. The cumulative sum over these sorted indices is shown in Fig. 2. As expected, the accumulated gain ceases to increase over the last 4155 features because they are simply absent in the training set.

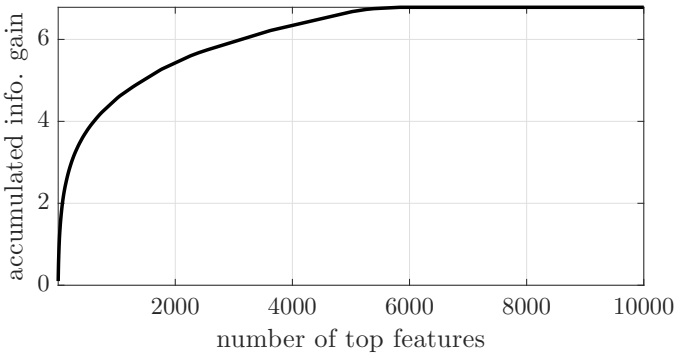


Fig. 2: Accumulated information gain calculated after sorting $I$ for all the features. Last 4155 features are absent in the training set, so they don't provide more information.

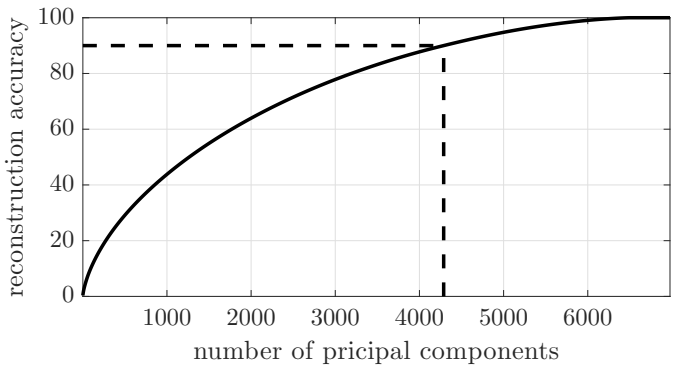


Fig. 3: Reconstruction error when PCA is applied to $[\mathsf{X}_l; \mathsf{X}_u]$ after removing 2353 missing features.

### C. Principle Component Analysis

Another way of reducing the dimension of features is PCA where we project the data onto a lower dimensional space while preserving the data structure as much as possible. Since our feature matrices $\mathsf{X}_l$ and $\mathsf{X}_u$ are sparse and large in dimension, we do not center the data as the resulting matrices would not be sparse. We select the principal loadings such that we achieve at least 90% reconstruction accuracy. For example, applying PCA to $\mathsf{X}_l$ and $\mathsf{X}_u$ together with 90% reconstruction accuracy requires 3482 principal components. This is shown in Fig. 3.

### D. Bigrams as Features

One of the limitations of using only unigrams is that the context of words can be lost. An example of this could be the topword 'nice.' In the labeled data set the probability of observing 'nice' in the "happy" tweets is three times the probability of observing 'nice' in the "sad" tweets. However, its occurrrence with another topword 'not' preceding it (i.e. 'not nice') would result in the opposite sentiment. A popular and simple approach to overcome this is to utilize *n-gram* language modeling features. Compared to unigrams alone, using bi-grams and skip-grams in conjunction with unigrams improves classification accuracy [1]. Therefore, we decided to test this approach by generating bi-gram features from the raw tweets.

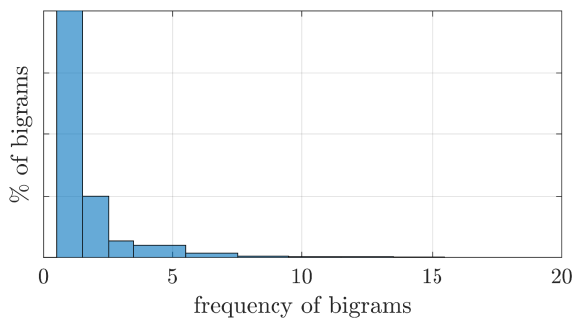


Fig. 4: Of all bigrams, 85.23% of those occur only once. We consider only the ones that occur more than once.

To generate the bigram features, we used an implementation of `ngrams` from [10], which creates an n-gram model from a document of text. We first generated a bi-gram model for each tweet and accumulated the generated features into a bi-gram frequency table. From this table, only bi-grams which occurred more than once were kept. This was to reduce the number for superfluous bigrams unique to only one tweet. This reduced the number of bi-grams features generated by 91.97% . A plot of the number of bi-gram features versus their occurrence is shown in Fig. 4.

### E. Stemming

Stemming is a common procedure in Natural Language Processing used to remove the suffix from a word. Upon analysis of the topwords we discovered that many words such as 'call,' 'called,' '#call,' 'calling,' etc. are derived from the same stem. Through stemming, we combine the words with the same sentiments and reduce the original 10000 features to 6860 features. We use the Porter Stemming algorithm [9] implemented as a Matlab library available at [4] to extract the stem from each top word. For words with hashtags, we remove the hashtag before stemming. We then derive a transformation matrix $\mathsf{T}_{\text{stem}}$ of dimension $p \times p_{\text{red}}$ where $p_{\text{red}}$ is the dimension of the remaining features after stemming. The transformed feature matrix $\mathsf{X}_{\text{red}}$ is obtained by $\mathsf{X}\mathsf{T}_{\text{stem}}$.

### F. Manual Truncation

Last but the not the least, the set of features that gave us the best test error were based on handpicking features. We removed some (junk) features that we thought should not be used to classify. Specifically, we removed (i) all numbers, (ii) features containing the string '\u', (iii) features with only '..' and '.', (iv) all single letter features, (v) features containing 'http' and (vi) features with punctuations except emoji's. After this procedure, we are left with 9024 features.

## III. Algorithms

In this section, we describe many algorithms we tried over time (not necessarily in the same order) including the four we submitted for evaluation. The submitted models are marked in green. For CV, we always use 10-fold cross validation. The test score means the score obtained after submitting to the leaderboard for evaluation.

### A. Naive Bayes

Naive Bayes is particularly suited for this problem since $n \ll p$. We have tried Naive Bayes with both multinomial distribution and multivariate bernoulli distribution [6]. The main difference between the models lies in the way we define $P(\mathsf{X}_i \mid \mathsf{Y}_i = c; \theta)$. If we denote $\mathsf{X}_i$ as the feature vector of the $i^{th}$ tweet, then in the bernoulli case, we get

$$P(\mathsf{X}_i \mid \mathsf{Y}_i; \theta) = \prod_{t=1}^{p} \mathsf{X}(i,t) P(w_t \mid \mathsf{Y}_i; \theta) + (1 - \mathsf{X}(i,t))(1 - P(w_t \mid \mathsf{Y}_i; \theta)) \tag{2}$$

where elements of $\mathsf{X}$ are all 1 or 0 denoting whether a word is present or absent. So we explicitly consider presence and absence of particular words which is not the case in the multinomial Naive Bayes using words counts as features:

$$P(\mathsf{X}_i \mid \mathsf{Y}_i; \theta) = \frac{(\sum_{t=1}^{p} \mathsf{X}(i,t))!}{\prod_{t=1}^{p} \mathsf{X}(i,t)!} \prod_{t=1}^{p} P(w_t \mid \mathsf{Y}_i; \theta)^{\mathsf{X}(i,t)}. \tag{3}$$

We used the Matlab implementation `fitcnb` for the multinomial case and our own code for Naive Bayes with Bernoulli distribution.

Both these methods use Laplacian smoothing that allows for missing features (see Sec. II-A) to have non-zero probability. With Laplace smoothing, we get prior probabilities of words given the labels as

$$P(w_t \mid \mathsf{Y}_i = c) = \frac{f(w_t, c) + 1}{\sum_{t'=1}^{p} f(w_{t'}, c) + p}, \tag{4}$$

where $f(w_t, c)$ is the frequency of word $w_t$ in category $c$.

Some features that do not appear in the training set have features with common stems which do appear. To account for this, we use Dirichlet smoothing. In [11], it is shown that the Laplace smoothing could add noise and other methods of smoothing may improve classifier performance. Dirichlet smoothing defines the prior probability as:

$$P(w_t \mid \mathsf{Y}_i = c) = \frac{f(w_t, c) + \theta P(w_t|\mathsf{X})}{\sum_{t'=1}^{p} f(w_{t'}, c) + \theta}. \tag{5}$$

We refine this definition as follows:

$$P(w_t \mid \mathsf{Y}_i = c) = \frac{f(w_t, c) + \theta P(w_{\mathrm{stem(t)}}|\mathsf{X})}{\sum_{t'=1}^{p} f(w_{t'}, c) + \theta} \tag{6}$$

where $\mathrm{stem(t)}$ is the set of words that have a common stem as word $w_t$ and $P(w_{\mathrm{stem(t)}}|\mathsf{X})$ is the probability that words of common stem as $w_t$ occurs in the collection. This way we could reflect the distribution of words of common stems, even when the current word is not present as a feature of the training set.

Next we compare the results from 3 different methods: (i) Multinomial Naive Bayes with Laplace smoothing, (ii) Bernoulli Naive Bayes with Laplace smoothing and (iii) Multinomial Naive Bayes with Dirichlet smoothing with same set of features, i.e. 9024 features as described in Sec. II-F. The quantitative comparison is shown in Tab. I. We observe that Laplace smoothing is better than Dirichlet smoothing in general. Multinomial Naive Bayes does better on the test score and Bernoulli Naive Bayes does better on the CV score. However, the difference between the two is not much.

TABLE I: Comparison of 3 variants of Naive Bayes.

| | CV score | Test score |
|---|---|---|
| Multinomial Naive Bayes - Laplace | 80.04% | 81.33% |
| Bernoulli Naive Bayes - Laplace | 80.39% | 81.13% |
| Multinomial Naive Bayes - Dirichlet | 80.18% | 79.46% |

TABLE II: Comparison of Multinomial Naive Bayes with different feature extraction methods.

| | CV score | Test score |
|---|---|---|
| Missing Features | 80.00% | 79.58% |
| Mutual Information | 80.91% | 79.04% |
| Bigrams | 81.29% | 79.58% |
| Stemming | 81.33% | 79.56% |
| Manual Truncation | 80.04% | 81.33% |

Next we tried all feature extraction techniques described in Sec. II with Multinomial Naive Bayes to evaluate what method works best in this problem. The summary of the results is presented in Tab. II. As we expected, we see that adding bigrams or stemming boosts the performance according to CV score. However, the test score for both of them is worse than using mere manual selection of features. We attribute this mismatch to the noise in the data. Another reason could be the size of the data set; perhaps on a larger set these techniques would generalize as they are known to work quite well in big data sets [1].

### B. Logistic Regression

Logistic regression is the first discriminative method we tried. Since this a supervised learning method, we remove the missing features only from the labeled set and the "junk" features as described in Sec. II-F. We also add a column vector of ones on the feature space to account for the bias term.

At first we tested Logistic Regression with $L_2$ regularization with the features described above. This resulted in CV score of 79.44% and test score of 80.02%. It is intuitive to think that using $L_1$ regularization will zero out the smaller weights and hence use only the more important words as the features. Therefore, it seems more reasonable to use $L_1$ regularization. However, it turns out that both CV score and test score are 78.22% and 78.13%, respectively, which are worse than using $L_2$ regularization. A possible reason is that $L_1$ zeros out weights on many features that could be important for classification. Fig. 5 shows the CV scores for different regularization methods and the various cost parameter settings we obtained while model tuning.

Finally, since the number of observations was much less when compared to the size of the feature space, we decided to use logistic regression after reducing the feature dimension using PCA. We first applied the exact procedure in Fig. 3 where we removed the missing features that are common

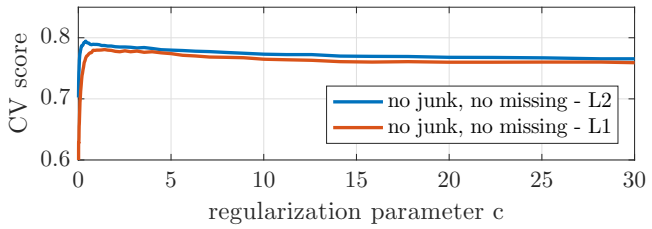


Fig. 5: Logistic Regression CV scores plotted against regularization parameters.

in both labeled and unlabeled sets and chose the number of components that provide 90% reconstruction accuracy. We obtained a CV score of 78.19% with $L_1$ regularized and 79.35% with $L_2$ regularized Logistic Regression. The performance in this case is not much different from using the features without PCA. In fact, we can argue that removing the "junk" features manually in the last part is similar to doing PCA!

We also applied PCA directly on 10000 features (without removing the missing features) and we obtained much better CV scores: 80.24% for $L_1$ and 81.56% for $L_2$. The missing features don't add any information to the data, so there is no reason to believe that PCA without removing missing labels should differ significantly from PCA after removing the missing labels. The reason is strongly attributed to the noise in the data or the mislabeling.

Given $n \ll p$, our first guess was that Naive Bayes would do better than Logistic Regression which turned out to be true.

### C. Support Vector Machines

SVMs are well known to learn decision boundaries efficiently in a higher dimensional space using the kernel trick. Our intuition was that the tweets were distributed in a highly non-linear feature space that would require a nonlinear transformation to discriminate between positive and negative tweets. To implement our SVM classifier, we used LIBSVM [2]. We used various kernels provided by the library including linear, quadratic, cubic, and radial basis function (RBF). For each kernel we executed an extensive grid search to find the optimal parameter settings. For the features, we tried the full feature space of word counts and a subset of the features which did not include those manually removed as described in Section II-F. In comparison of the CV scores, we found that RBF kernels with $\gamma = 0.02$ and $c = 5$ with manually removed features performed the best with CV score 81.02%. However, the test score for this method dropped down to 79.02%. Compared to the best Naive Bayes, it appears the SVM with same features is slightly overfitting on the training set.

### D. Semisupervised Naive Bayes

Even though Naive Bayes gave our best performance so far, we never made use of the unlabeled features. In this section, we extend the standard Naive Bayes to do semisupervised learning. Now, we use both labeled and unlabeled

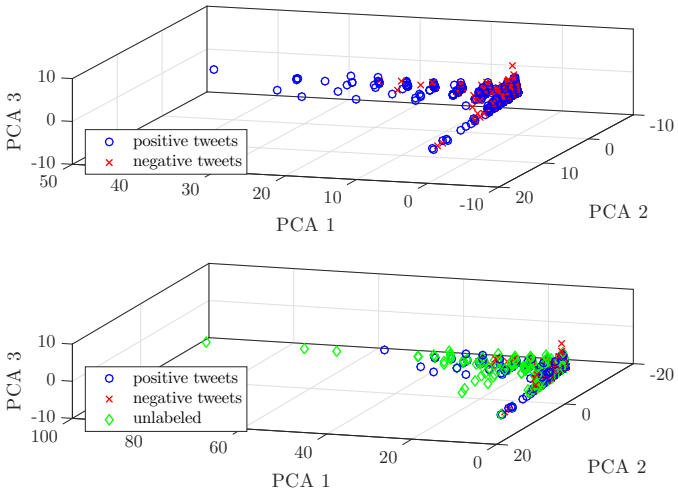


Fig. 6: Top: Clustering of labeled data in top 3 PCA dimensions. Bottom: Transformation of unlabeled data using same principal components superimposed with the labeled data.

data and improve (in terms of log-likelihood) the model iteratively using Expectation Maximization (EM) following the procedure defined in [7].

We had to code this algorithm ourselves since the code is not available. We initialize the algorithm by predicting the class probabilities of the unlabeled data. Then, in each iteration, we train a multinomial Naive Bayes classifier using both labeled data and predicted labels of the unlabeled data. We modify the predictions on the unlabeled set everytime and stop when the parameters converge. The EM algorithm is bound to converge, but as it happened in this case, it converged to a local maxima. It turns out this algorithm is strongly dependent on the initialization. We also tried several random initialization for the class probabilities of the unlabeled data, but the CV score did not really improve over the standard Naive Bayes. When trained on 9024 features after removing the junk as described in Sec. II-F, we get a CV score of 76.84%.

An alternate approach to semisupervised learning is to combine $k$-means clustering with Naive Bayes. We first visualized the data in 3D by calculating the first 3 principal components as shown in Fig. 6. We observe that the labeled data can be easily grouped into clusters parallel to PCA 2 and the same pattern is also visible when we superimpose the unlabeled set onto the labeled one. We performed $k$-means clustering on both the labeled training data and the unlabeled testing data. From the clusters which were formed, we took two approaches:

1) In the first approach, we trained a Naive Bayes classifier using the labeled examples in each of the clusters. In the testing stage, we determined the cluster to which the example was closest and used the Naive Bayes classifier trained for that cluster to determine the label.
2) In the second approach, we trained $\binom{k}{k-1}$ classifiers, each using $k-1$ clusters of the total $k$ clusters. In the

testing stage, we classified by using a maximum-voting strategy for each of the clusters.

For both approaches, CV score was performed to determine the feature extraction method to use and the number of cluster for training the NB classifiers. For the first approach, having $k = 2$ and combining features with common stems had the CV score of 80.60% and test score of 79.46%. For the second approach, having $k = 6$ and combining features with common stems had the CV score of 80.82% and test score of 79.51%. We could only try euclidean norm as the distance metric; perhaps a different norm would result in better performance.

### *E. k-Nearest Neighbors*

We used k-NN as an instance based classifier. We used the Matlab function `fitcknn` to train the k-NN classifier. In k-NN, the parameter k, the feature space and the distance metric are the key components in determining the performance of the classifier. For features, we tried both the full feature space as well as the reduced feature space without missing and junk features. For the distance metric, we tried both the euclidean distance as well as the city-block distance metric. Surprisingly, the euclidean distance metric performed better than the city-block distance metric. For features, the reduced feature space performed better. The best CV score we could achieve using k-NN was 66.71% which is not even close to CV score obtained by other methods.

### *F. Ensemble Methods*

We have also tried forming ensembles of a simple Naive Bayes classifier using a bootstrapping procedure [3]. We used random sampling with replacement for the training samples and sampling without replacement for the training features and chose the size of the ensemble, the sample size and the number of features for each classifier by cross validation. It seems that even for an ensemble size of 100, the performance of ensembles is no better than a single Naive Bayes model trained on full data set and all the features.

In addition to the ensemble of Naive Bayes classifiers, we created an ensemble of different classifiers. Specifically we trained a Naive Bayes classifier, a Logistic Regression classifier, and a Support Vector Machine classifier. The label of a test example was determined by taking the max-vote of the three classifiers. The motivation comes from the fact that different classification algorithms with different sets of features and training processes would focus on various aspects of the feature space and in the process we will correct for the ones which might be wrongly classified by a single algorithm. All three classifiers were trained manually removing the features and combining features with common stems. In particular, for Logistic Regression we used $L_2$ regularization and for SVM we used 6 different classifiers of various $\gamma$ and $c$ settings. The best CV score that we achieved is 80.47% with the test error of 79.78%.

## IV. Conclusion

We have tried plethora of feature selection methods with supervised as well as semisupervised learning algorithms. It's intriguing that our best method that also won us the $3^{rd}$ prize is based on a multinomial Naive Bayes algorithm with handpicked features. The feature set was obtained by removing irrelevant words from the top word list like single characters, numbers, unicode characters, punctuation except emoji's and url's.

There are many reasons why we think this method works the best. Firstly, Naive Bayes is very well suited to more features and less observations type of data, or in other words when $n \ll p$. Secondly, the data set is full of noise, i.e. mislabeled data. This is an important reason why ensemble like methods don't seem to outperform a simple model like Naive Bayes. Logistic Regression and SVM do pretty well on CV, but they overfit very easily because of this noise. Naive Bayes on the other hand assigns probabilities as to how each word contributes to "happiness" or "sadness" and is more robust to presence of noise.

We also focused on semisupervised learning with Naive Bayes which looked very promising. We used the EM algorithm to update the model after each iteration, but it turns out that the initialization is quite important, especially when the joint distribution of the data cannot be easily generalized over new observations. In such cases there is danger of converging to a local maxima.

Lastly, we expect that all the methods would show better performance on a bigger data set.

## References


[1] B. M. Badr and S. S. Fatima. Using skipgrams, bigrams, and part of speech features for sentiment classification of twitter messages. *Proceedings of the Twelfth International Conference on Natural Language Processing*, 2015.

[2] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[3] T. G. Dietterich. Ensemble methods in machine learning. In *MULTIPLE CLASSIFIER SYSTEMS, LBCS-1857*, pages 1–15. Springer, 2000.

[4] J. C. Lopez. Matlab impelmentation of porter stemming algorithm. https://tartarus.org/martin/PorterStemmer/, 2006.

[5] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[6] A. McCallum, K. Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[7] K. Nigam, A. K. Mccallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using em, 1999.

[8] J. Novovičová, A. Malík, and P. Pudil. Feature selection using improved mutual information for text classification. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 1010–1017. Springer, 2004.

[9] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[10] L. Shure. Text mining shakespeare with matlab. http://blogs.mathworks.com/loren/2015/09/09/text-mining-shakespeare-with-matlab/, 2015.

[11] Q. Yuan, G. Cong, and N. M. Thalmann. Enhancing naive bayes with various smoothing methods for short text classification. In *Proceedings of the 21st International Conference on World Wide Web*, pages 645–646. ACM, 2012.