# Twitter Sentiment Classification

**[Memory Error]** Concetto Emanuele Bugliarello, Manik Garg and Zander Harteveld
*Pattern Classification and Machine Learning (CS-433), Second Project, Fall Semester 2016, EPFL, Switzerland*
{emanuele.bugliarello, garg.manik, zander.harteveld}@epfl.ch

***Abstract*—Social media platforms such as Twitter are a rich source of text examples expressing positive and negative sentiment. In this paper, we investigate the classification accuracy achieved by different learning algorithms in determining sentiments associated to tweets. Through data pre-processing and hyperparameter tuning, we report our computationally light TF-IDF Logistic Regression model, scoring 87.00 % accuracy in the EPFL ML Text classification challenge on Kaggle. .**

## I. Introduction

Sentiment analysis is an interesting problem aiming to give a machine the ability to understand the emotions and opinions expressed by humans. This is an extremely challenging task due to the complexity of human language, which makes use of rhetorical devices such as sarcasm or irony.

Twitter is a popular "micro-blogging" social networking website for conveying opinions and thoughts, and thus a successful sentiment classification model based on Twitter data could provide interesting trends regarding prominent topics in the news or popular culture. For example, one could gauge the popular opinion of a politician by calculating the sentiment of all tweets containing the politician's name. Sentiment analysis in Twitter is a significantly different paradigm than past attempts at sentiment analysis through machine learning since its users are only allowed to post short status updates of less than 140 characters ("tweets"). Moreover, as in most online social networks, users create their own words and spelling shortcuts, that, in addition to misspellings, slang, and abbreviations, make this task even more challenging.

The aim of this project is to build an accurate sentiment analyzer for tweets. That is, given a user-generated status update, our classification model determines whether the given tweet reflects a positive or negative opinion on the users behalf.

In this report, we will discuss about the methods for building such sentiment analyzer including data pre-processing, creation of word vector representations and hyperparameter optimization for different classifiers.

## II. Data

The dataset we use to train and test our classifiers is the one provided for the EPFL Machine Learning text classification challenge on Kaggle[1]. The dataset consists of two small sets of training tweets for each of the two classes[1] (100,000 tweets each), two complete sets of training tweets for each of the two classes (1.25M tweets per class) and a test set (10,000 tweets).

## III. Methodology

In the following sections, we analyze different classifiers: Logistic Regression (LogReg), multinomial Naive Bayes (MNB) and Support Vector Machine (SVM) with the linear kernel. To evaluate the performances of these classifiers, we run experiments on the provided small sets of labeled tweets, splitting them in train (80%) and test (20%) sets. The learning is then performed using a 3-fold cross validation (CV), such that, in each fold, 2 partitions are used for training and 1 partition for validation. Testing is then done on the completely held-out set. Standard deviations (std) of the training set are reported from validation accuracies around 3-fold CV. On the other hand, std on testing set are reported from the accuracy on test set and mean validation accuracy from the 3-fold CV. The only metric used to assess the performances of the classifiers, in accordance with the competition's objective, is their accuracy in predicting correct labels.

### A. Feature generation

To build an accurate text classifier, it is crucial to find a good feature representation of the input text. Out of the numerous text feature engineering methods, we discuss two of them in this following section.

- **GloVe (Stanford University).** The basic idea behind Global Vectors for Word Representation (GloVe) [2] is that ratios of word-word co-occurrence probabilities can encode some form of meaning. Thus, one can measure the relatedness of two words by computing the Euclidean distance (or cosine similarity) between two word vectors.
- **TF-IDF.** Term Frequency - Inverse Document Frequency (TF-IDF) is a common numerical statistic that is intended to reflect the importance of a word to a document in a corpus. It consists of two terms. First, the term frequency (TF) measures how frequent a specific term appears in a document. The second term, inverse document frequency (IDF), is computed as the logarithm of the number of documents in the corpus divided by the number of documents where the specific term appears. It weights down terms occurring very frequently in the corpus and increases the weight of terms that occur rarely; thus giving a measure of how important a term is.

### B. Pre-processing

We now present various pre-processing steps suitable for this text classification task. The dataset provided for the competition has already been pre-processed by (i) replacing user references with a generic token (@XYZ → <user>), (ii) replacing URLs with a generic token (<url>), (iii) lowering the text and (iv) adding whitespace around punctuation.

Through our error analysis and intuition, we notice several tweet characteristics that could potentially be useful as pre-processing for a TF-IDF feature representation. The following shows a list of pre-processing steps we have extensively been investigating:

- **Remove numbers.** Usually, numbers do not convey any emotions and could thus be stripped away. This is not always true: dates such as 09/11/2001 can clearly identify the polarity of a tweet.

[1]Positive/negative sentiment associated to the tweet.

- **Stemming**. Stemming is reducing words back to their root word. This might be useful as those words usually have a very similar meaning and can be grouped together.
- **Lemmatization.** Lemmatization is determining the lemma of a word based on its intended meaning with the use of a vocabulary and morphological analysis of words. This results in removing inflectional endings only and to return the base or dictionary form of a word.
- **N-grams.** N-grams can help detecting the correct meaning of a sentence by including combination of multiple words as tokens. N specifies the number of words in each combination.
- **Remove stop words.** Usually, stop words are extremely common words which would appear to be of little value in helping select documents matching a user need (sentiment classification in our case).
- **Remove `<user>` and `<url>`.** Any group of words can actually be chosen as the stop words for a given purpose. Thus, due to their high frequency, `<user>` and `<url>` can be considered as stopwords and so could be removed from the text.
- **Remove the pound sign.** A hashtag may just consist of a single word and thus removing the pound sign (#) at the beginning of it would improve the TF-IDF weights.
- **Group emoticons.** Preserving emoticons and mapping different representations of the same emoticon into one (e.g., { :-), : ), ( - :, (: } → { :) }) could let the classifier associate it to a specific polarity.
- **Negate verbs.** Inspired by Pang and Lee's sentiment analysis research[3], stripping out the word "not" from tweets and appending the characters NOT_ before the following token in a tweet could identify negative feelings associated to otherwise positive words. For instance, "She does not like me" would become "She does NOT_like me" and so "NOT_like" would be a new (negative) token.
- **Fix common typos.** Typos are very common in tweets and thus fixing the most common ones (e.g., people forget the apostrophe when negating an auxiliary) might correctly identify several instances.
- **Replace repeating letters.** By looking at the tweets, it is possible to see that sometimes people repeat letters to stress the emotion (e.g., "hunggrryyy", "huuuuuuungry" for "hungry"). Thus, another pre-processing step is to look for two or more repetitive letters in a word and replace them by just two of them.

## IV. Results

In this section we specify the results obtained for each of the methodologies discussed, and the analysis performed in order to construct our final sentiment analysis model.

### A. Feature generation

To select the best model for generating numerical word representations, we compare the GloVe word embedding, with both 20 and 100 features per word, with the default instance of TF-IDF from *scikit-learn*[4]. To obtain a first insight into the performances of the models, we perform the training of the obtained word-to-vector matrix and word-frequency-based feature matrix using the LogReg and SVM classifiers from *scikit-learn* without any hyperparameter optimization. As it is shown in Table I, the best classification results of the raw data are obtained using TF-IDF word vectors. We therefore proceed using TF-IDF to optimize data pre-processing and hyperparameter optimization steps for LogReg and SVM classifiers in order to improve their accuracy. Moreover, the accuracy of a classifier increases with the dataset size, as shown in Fig. 1.

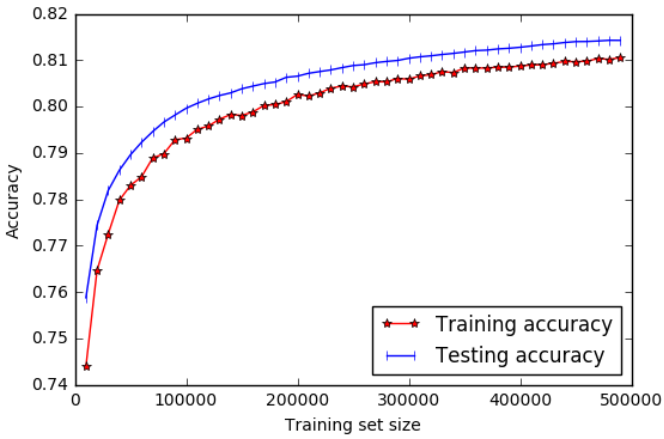


Fig. 1: Effect of dataset size on accuracy as observed on test set with default TF-IDF LogReg model.

### B. Pre-processing

We now show the additional pre-processing steps, in an aggregating manner, used to obtain the best classification accuracy for each of the aforementioned classifiers. The baseline pre-processor is referred to the received dataset (having applied the pre-processing steps described in section III-B).

*1) Dummy:* The baseline classifier is a random classifier that is evaluated on the received dataset without any pre-processing. As expected, the accuracy of such classifier is around $50\%$. Specifically, $0.5018 \pm 0.0019$ for a 3-fold CV.

*2) LogReg:* For each additional pre-processing step, a LogReg classifier (with default regularization parameter $C$=1) is trained. The pre-processing steps that yielded improvements on test accuracy are listed in Table II. Other pre-processing steps yielded lower test set accuracies.

| | **train** | **test** | **change** |
|---|---|---|---|
| base | 0.8048±0.0009 | 0.8097±0.0024 | 0.00% |
| +repeating letters | 0.8227±0.0015 | 0.8271±0.0015 | 1.74% |
| +stem | 0.8254±0.0020 | 0.8303±0.0025 | 0.30% |
| +remove # | 0.8258±0.0022 | 0.8306±0.0024 | 0.03% |
| +2-grams | 0.8372±0.0013 | 0.8457±0.0046 | 1.51% |

TABLE II: Best pre-processing steps for default LogReg.

The improvement observed by replacing repeating letters is reflective of the way people express their emotions in a textual form.

*3) SVM:* The preprocessing steps that yielded improvements on test accuracy for SVM are listed in Table III.

| | **train** | **test** | **change** |
|---|---|---|---|
| base | 0.7908±0.0016 | 0.7918±0.0005 | 0.00% |
| +repeating letters | 0.8142±0.0011 | 0.8163±0.0010 | 2.45% |
| +stem | 0.8142±0.0020 | 0.8171±0.0014 | 0.08% |
| +2-grams | 0.8178±0.0018 | 0.8198±0.0010 | 0.27% |

TABLE III: Best pre-processing steps for default SVM.

| Embeddings | Classifier | Accuracy |
|---|---|---|
| GloVe with $\eta$ = 0.001, $\alpha$ = 0.75, nmax = 100, epochs = 10 | | |
| • 20 features | LogReg | 0.5965 $\pm$ 0.00197 |
| | SVM | 0.6083 $\pm$ 0.00140 |
| • 100 features | LogReg | 0.6159 $\pm$ 0.00069 |
| | SVM | 0.6041 $\pm$ 0.00165 |
| TF-IDF | | |
| • default | LogReg | 0.8097 $\pm$ 0.00243 |
| | SVM | 0.7914 $\pm$ 0.00039 |

TABLE I: Numerical word representation models on small data set. Classifiers are from scikit-learn, with default settings.

*4) MNB:* The preprocessing steps that yielded improvements on test accuracy are listed in Table IV.

| | train | test | change |
|---|---|---|---|
| base | 0.7642±0.0011 | 0.7650±0.0004 | 0.00% |
| +replacing not | 0.7664±0.0015 | 0.7671±0.0003 | 0.21% |
| +english stop words | 0.7668±0.0009 | 0.7675±0.0003 | 0.04% |
| +2-grams | 0.7846±0.0013 | 0.7903±0.0028 | 2.35% |

TABLE IV: Best pre-processing steps for default MNB.

A pictorial representation comparing the best pre-processing steps for each classifier is given in Fig. 2.

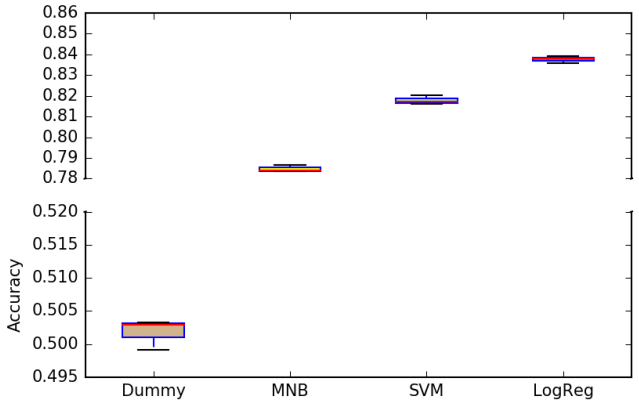


Fig. 2: Accuracies for each investigated classifiers (default hyperparameters from *scikit-learn* modules) including its optimal pre-processing.

It is thus clear that replacing multiple occurrences of two words improves the classification accuracy of LogReg and SVM. Another determining hyperparamter is the *n-grams* while feature generation.

### C. Hyperparameter optimization

The *CountVectorizer()* and *TfidfTransfromer()* from *scikit-learn* have multiple hyperparameters that can be tuned. A first hyperparameter is `N-gram`, already described in the Methodology section. Other hyperparameters are `max_df`, a threshold to cut off highly frequent words; and `min_df`, a threshold to clip off very rare words. Moreover, the maximal amount of features, `max_features`, can be set. Another hyperparameter is the inverse L2 regularization strength (`C`) of the LogReg.
In order to search through the complete landscape of each hyperparameter, we first apply a random search using *scikit-learn*'s *RandomSearchCV()*, on a wide range of hyperparameters to get an initial idea of the optimal parameters for the full dataset. The returned values are reported in Table Va and give an accuracy of $0.8686 \pm 0.0023$ on the full dataset.

(a)

| hyperparameter | value |
|---|---|
| `max_features` | *None* |
| `ngram_range` | $(1, 3)$ |
| `max_df` | $\approx 0.9261$ |
| `min_df` | 4 |
| `C` | $\approx 2.1544$ |

(b)

| hyperparameter | value |
|---|---|
| `max_features` | *None* |
| `ngram_range` | $(1, 3)$ |
| `max_df` | $\approx 0.9261$ |
| `min_df` | 4 |
| `C` | 3.41 |

TABLE V: Hyperparameters returned by (a) coarse RandomSearchCV() and (b) fine sampling around the optimal range.

We then perform a fine-grain search on the small dataset to optimize the hyperparameter `C` for Logistic Regression. Fig. 3 illustrates the testing accuracy peak of $0.8534 \pm 0.0039$ at around `C` $= 3.41$ on the small dataset that yield an accuracy of $0.8739 \pm 0.0017$ on the full dataset after 3-fold CV. The final hyperparameters are then the ones specified in Table Vb with `1783165` features.

To analyze the classification performance of our classification system, we plot the *receiver operating characteristic (ROC) curve* 4. This is the true positive rate against the false positive rate for the different possible cut-points. Generally, the closer the curve to the left upper corner of the ROC space, the more accurate the classification; and the closer the curve to the diagonal, the less accurate the classification. Therefore, the *area under the curve (AUC)* is a measure of classification accuracy. One can see that the AUC of the mean ROC over three folds is $0.92 \pm 0.0047$, showing that our classification holds an accuracy of $92\%$.

## V. Discussion

Starting with different types of embeddings, the TF-IDF embedding gives the best baseline results. This is surprising as the popular GloVe algorithm can be regarded as a state-of-the-art. Thus, we speculate that GLoVe would have given comparable accuracies as TF-IDF with proper data pre-processing and/or using computationally expensive Neural Networks.
Despite the fact that the current state-of-the-art[5] makes use of Convolutional Neural Networks, the lack of computational power to train them is not negligible. This is why we chose the TF-IDF: it is ease to be implemented and requires very small computational power. The strength of our final model is its computational simplicity compared to large Neural Networks. Our model can be trained in a moderate time interval (50 minutes on a laptop powered by 2.2-GHz Intel Core i7 processor and 16

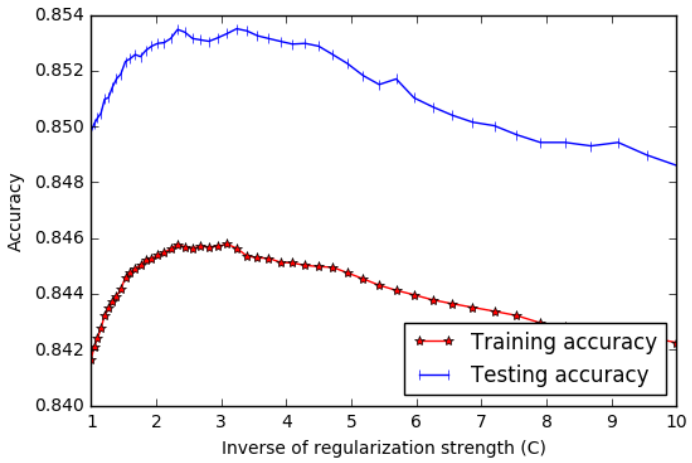


Fig. 3: Landscape of the hyperparameter C of Logistic Regression classifier on features obtained from optimized TF-IDF.

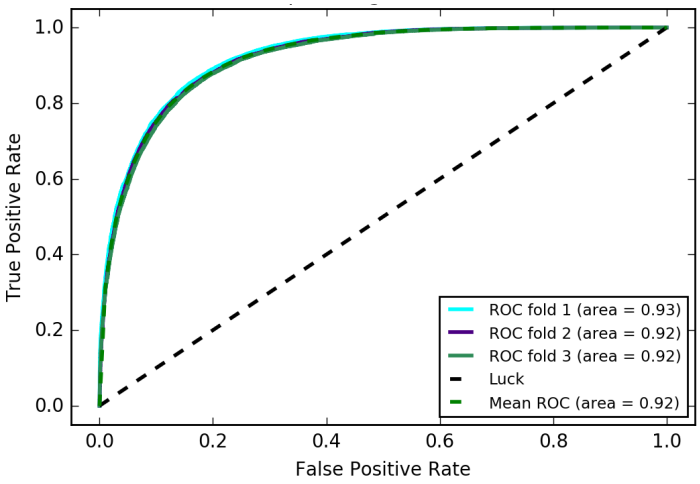


Fig. 4: ROC curve ilustrating the LogReg classifier performances over 3-CV fold.

GB of RAM) while still producing impressive predictions. This is achieved by optimizing the model on different levels from the data pre-processing up to the classifier. We have reported only the pre-processing steps that actually increase the accuracy alone, ensuring a better prediction. The features were engineered using several tricks such as N-grams for better capturing the meaning of a tweet. All hyperparamters have been tweaked extensively using a first round a coarse-grained global "random" search, and in a second round a very fine tuning on a small set of possible parameters around the best result from the first round. Finally, we have *1,783,165* features that lead to a highly optimized Twitter sentiment analyzer with an accuracy of $0.8739 \pm 0.0017$ on 3-fold CV on the full dataset using Logistic Regression. Here each feature represent a token in the vocabulary with one or more words.

Naturally, there exits further possibility to improve our sentiment analyzer. Considering the free available modules from the Natural Language Toolkit (NLTK)[6] for text pre-processing and classification, one can think of introducing extra levels of information such as the Part-Of-Speech tagging or Named-entity recognition. Moreover, it would be interesting to see the pre-processing steps applied to tweets that are then fed into a powerful method, such as CNNs.

## VI. Conclusion

In this report, we described and compared several sentiment classifiers optimized towards predicting the sentiment polarity of tweets. Our approach is based on simple classification scheme in contrast to highly complex Neural Networks and therefore computationally quite cheap. Though it relies on a large amount of training data. After pre-processing, feature engineering and classifier optimization we were able to build a sentiment analyzer trained in a moderate time window while holding a good performance in terms of accuracy.

## References

[1] EPFL. (2016) Epfl ml text classification. [Online]. Available: https://inclass.kaggle.com/c/epfml-text/

[2] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[3] L. L. B. Pang and S. Vaithyanathan, "Thumbs up? sentiment classification using machine learning techniques," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2002, pp. 79–86. [Online]. Available: http://www.cs.cornell.edu/home/llee/papers/sentiment.pdf

[4] 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.

[5] J. Deriu, M. Gonzenbach, F. Uzdilli, A. Lucchi, V. De Luca, and M. Jaggi, "Swisscheese at semeval-2016 task 4: Sentiment classification using an ensemble of convolutional neural networks with distant supervision," *Proceedings of SemEval*, pp. 1124–1128, 2016.

[6] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ser. ETMTNLP '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 63–70. [Online]. Available: http://dx.doi.org/10.3115/1118108.1118117