COMP3222 Machine Learning Technologies Coursework

Metodi Istatkov, mi2u17@soton.ac.uk

Student ID: 29338743

Table of Contents

Introduction

The problem being addressed by the MediaEval 2015 “verifying multimedia use” dataset is the spread of fake news in social media. Nowadays social media platforms like Twitter and Facebook have become major sources of news. Millions of people use them on a daily basis and this is how they keep themselves informed of what is happening around the world [1]. All of this means that news are spread rapidly online. However, not all information posted or shared in social media is credible. There is considerable amount of fake news propagating online and they can be shared very quickly leading to a lot of people being misinformed. Fake news could be spread easily in cases of natural disasters, terrorist attacks and in general when there are incidents connected to public safety. Very often this could lead to the misinformation of not only social media users but authorities as well [1]. Thus comes the need for creating a tool or a system that is capable of handling large amounts of information and verify how credible it is in real time. This way fake news could be disregarded and hopefully the overall quality of news posted on social media would increase.
The majority of the tweets are connected to Hurricane Sandy. There are also other events covered, like the Nepal earthquake from 2015 and a solar eclipse from the 20th March 2015.

Data Analysis

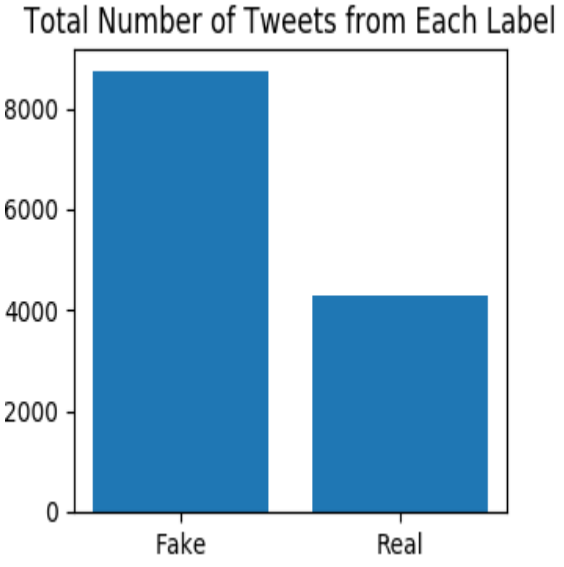
The dataset that has been provided is split in two parts - training data and testing data. The training data should be used to train the model while the testing one should be used to evaluate how well the model predicts. The training set consists of about **14 000** entries and the testing one has almost **4000**. Since the datasets are relatively small there is no need for large computational power. The computational speed of a regular computer suffices. The way that data is formatted in each of the files is the same. It is presented in a table format and is split in the following columns:

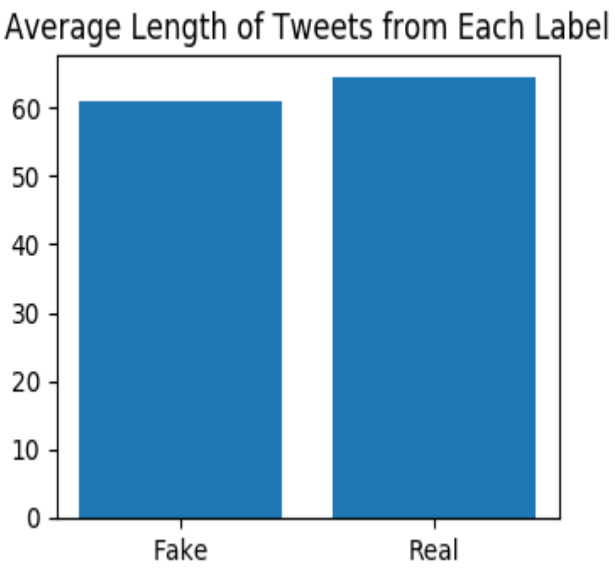
tweetId	The identifier of the post in Twitter’s database.
tweetText	The tweet body. This is what the machine learning model would use to evaluate whether the tweet is fake or real.
userId	The id of the user’s account in Twitter’s database
imageId(s)	A lot of the tweets have an image associated with them. This column displays the id of

	the image.
username	The username of the Twitter user who has tweeted or retweeted the post
timestamp	The timestamp indicates when the tweet has been published. It contains date and time and is in the following format: Day of the week / Month / Day of the month / Time / Year
label	This label could be either *fake* or *real*, signifying whether the tweet is credible or its content is imaginary.

The quality of the data is good in general since all entries have both tweet text and a label, which are the two essential fields needed for the machine learning model. There are a few small problems though. One is the fact that not all tweets are in English. A significant part of them are either in Spanish or in some other language. The potential problem that could arise from this fact is a decrease in the classifier's accuracy since most of the tools available have been created for working with text in English. Another issue is that there is a third label (besides fake and real) in the training set data called *humor*. In the FAQ section of the assignment is mentioned that this label should be treated as a fake one, so in the end there are two classification classes (binary classification). Once I converted all tweets with a humor label to a fake one I noticed that the number of fake tweets became twice more than the number of real ones and so the data became skewed.

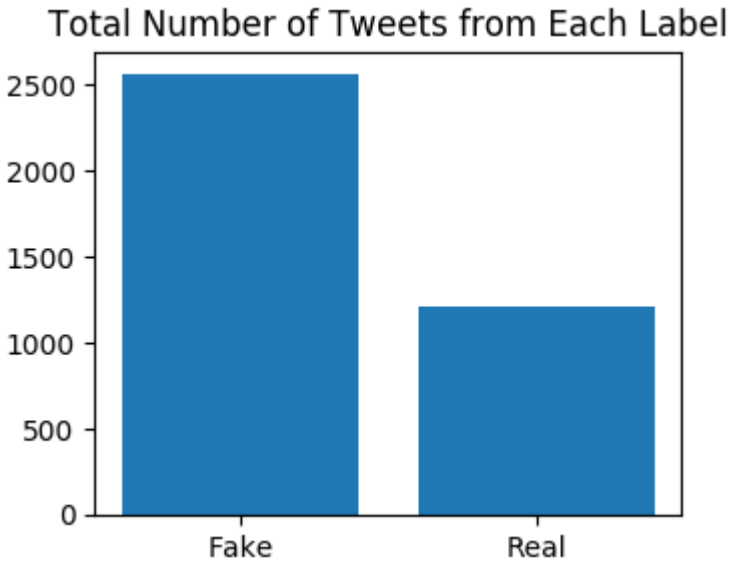
Here are two graphs representing the total number of fake and real tweets and also the average tweet length, in terms of number of characters, for each label from the **training** set:

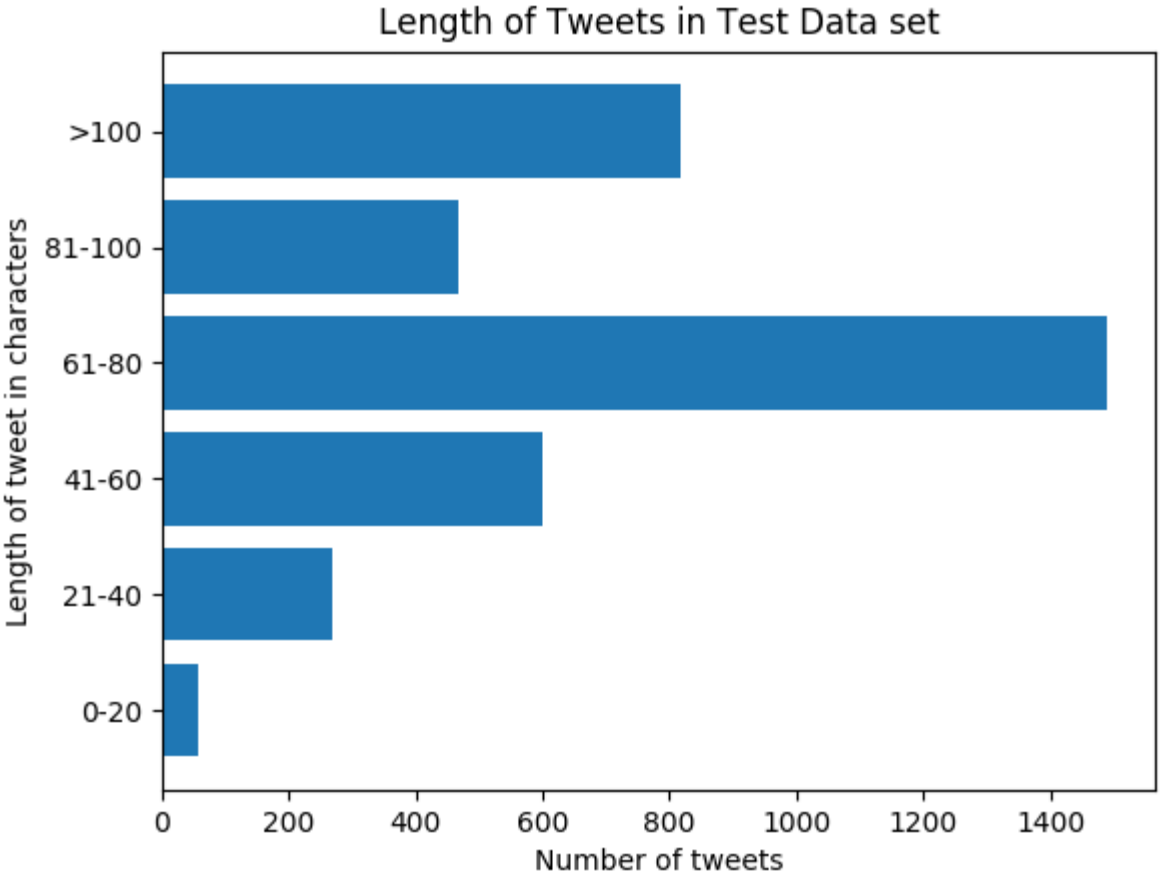




As you can see from the two graphs there are a total of 8741 fake tweets (fake + humor to be precise) and 4282 real tweets. The average length of tweet's text is about the same regardless of the label.

The following graphs relate to the **testing** set:





As in the training dataset, the number of fake tweets is significantly larger than the real one. Most of the tweets range from 61 to 80 characters in length, but there are some that are more than 100.
There is a significant number of retweets in the dataset, which means that there is repetitive information that could be fed to the model. Also the fact that there is a notable difference in the number of fake and real tweets in both training and testing data could lead to data bias.

Algorithm Design

Preprocessing

I tried using various different algorithms (there is information about all options I tried further down in the Evaluation section) but I did the same preprocessing and feature selection for all of them. Preprocessing includes the following steps:

- Data Cleaning for Text

  The first operation that I perform is to convert the *humor* label to *fake* label as mentioned in the FAQ section of the assignment. After that I initially tried removing all non-english tweets and therefore train my models only on English tweets. However, I saw that this led to a lower F1 score so I gave up on this idea. I also tried to translate the foreign language tweets instead of removing them but I could not find a reliable and free python library that could perform accurate enough translations in order to increase the F1 score. The next step was to remove all links from the texts, because they did not contain any valuable information for improving the model. Following that I removed all retweets as well so that I could avoid biasing my model by introducing repetitive information. I had to reset the dataframe indexes after I removed the retweeted entries. Furthermore I performed stemming on all words in the dataset.

- Categorical Features

  I am using scikit learn's TfIdfVectorizer class which performs some preprocessing operations as well. Most of them are categorical features like sentence and word tokenization, and creation of n-grams. So every tweet text was tokenized and I experimented with different sizes of n-grams. In the end I found out that I received the best results when n was equal to 1, which is basically just leaving words unigrams.
  Another property of the TfIdfVectorizer is that it allows you to specify a list of stopwords that will be neglected when tokenizing and creating a dictionary of frequencies. I used a list of common English stopwords.

Feature Selection

When it comes to feature selection TfIdfVectorizer comes in handy again. It combines the functionalities of CountVectorizer and TfIdfTransformer (again from scikit learn). Thus, I am

first creating a sparse matrix with token counts, and then this matrix is transformed to tf-idf one in normalized representation.

Machine Learning Algorithm Used

The machine learning algorithm that worked best for me was MultinomialNB(), which is the implementation of Naive Bayes in scikit learn. It is based on the famous Bayes Theorem from probability. The theorem states that an event is likely to happen, provided that another even has already taken place [3]. Naive Bayes is a supervised learning classifier, which works by predicting (using conditional probability) whether a given word/token is part of a certain class. Then the algorithm checks which class achieves the highest probability and identifies this class as the most likely one to contain a piece of data and therefore predicts a classification. This process could also be referred to with the name *maximum a posteriori (MAP)* rule [3].

Justification for choices

In the preprocessing steps I removed the http links because they did not contain any valuable information about the machine learning module, but I decided to leave the hashtags because they could be used as key words. Then removing retweets was an important step to avoid bias by feeding repetitive data to the model. Also performing stemming on all words allowed for easier identification of similar context.
Then, I decided to use TfIdfVectorizer because it provided some additional preprocessing steps like tokenization and n-gram forming in an optimized way [2]. Moreover, it combines CountVectorizer followed by TfIdfTransformer. The reason why I applied tf idf transformation to the matrix produced by the CountVectorizer is to avoid having a situation where there are a lot of insignificant words with high frequency. This could happen if I just used the token frequencies in raw format. I set the TfIdfVectorizer argument *analyzer* equal to word, because this tells the transformer to make n-grams of words rather than character. Having in mind that the tweets contain sentences and complete text this was the better choice. Also I used Grid Search to determine that the best parameter for *ngram_range* was the default one (1,1). The majority of the tweets in the dataset are in English, thus I used a set of common English stopwords.
Multinomial Naive Bayes is considered as a suitable algorithm in the context of text classification [5]. This comes mainly from the fact that it is performing quite fast(it was the fastest algorithm in terms of computational time from all that I explored), especially when compared to some other algorithms, and it is relatively simple [3]. It is also a good choice when the training data is of small or medium amount [3]. Another characteristic of the algorithm is that it works well with both binary and multiclass classifications [3]. In the case of this coursework the machine learning model needs to make binary classification (decide whether a tweet is fake or real), which fits well with the aforementioned characteristic of Naive Bayes.

Evaluation

Since the goal of this coursework is to create a fake (as defined in the MediaEval data set) news classifier, I am evaluating how well my machine learning algorithm detects fake tweets.

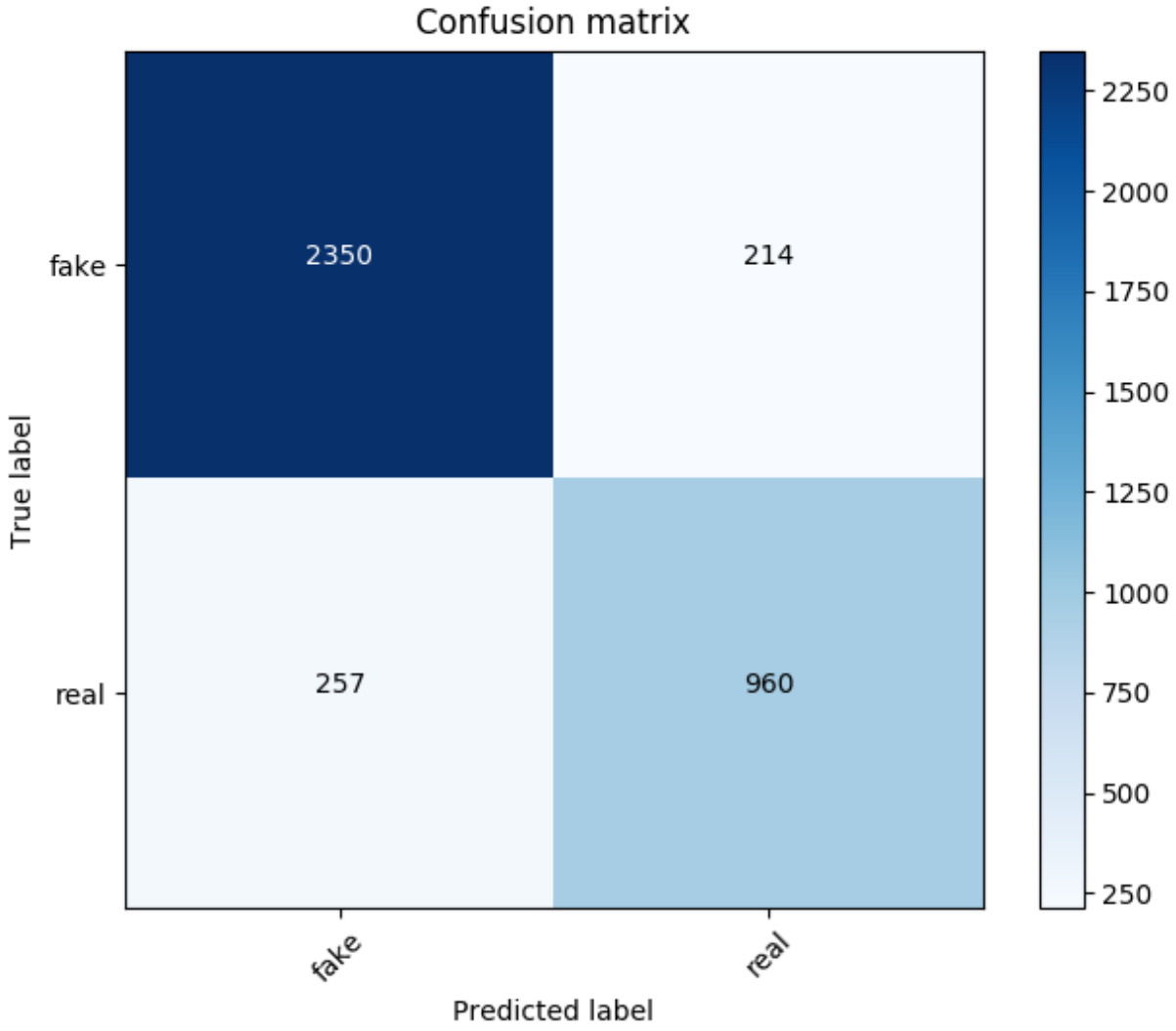
TP = correct fake classification
FP = incorrect fake classification
TN = correct real classification
FN = incorrect real classification

Precision (P) = TP/(TP + FP)
Recall (R) = TP/(TP + FN)
F1 = 2*P*R/(P+R)

The best F1 score that I acquired for correctly classifying fake tweets using Multinomial Naive Bayes algorithm is **0.91.**

	precision	recall	f1-score	support
fake	0.90	0.92	0.91	2564
real	0.82	0.79	0.80	1217
accuracy			0.88	3781
macro avg	0.86	0.85	0.86	3781
weighted avg	0.87	0.88	0.87	3781

And this is the confusion matrix for the same algorithm configuration:



I managed to achieve this score using the following configuration for my algorithm: **MultinomialNB(alpha=1.0, fit_prior=True)**. I found that those are the best parameters after running a Grid Search. I believe that my algorithm had a very good overall performance on the provided dataset. My first idea of using Naive Bayes as a classifier came after I read a paper about detecting fake news in Facebook posts using Naive Bayes. The authors of the paper estimated that their classification accuracy was about 74 % [6]. For comparison, you can see on one of the above pictures (the one with f1 scores) that my accuracy was about 88 %. Most of the papers that I have cited are building fake news classifiers with Naive Bayes and SVM algorithms.

Some other configuration of the same algorithm that I tried were with setting the *alpha* parameter equal to larger floating numbers while keeping fit_prior true. For example when I set it to 5.0 I ended up having 0.81 f1 score for fake classification and 0.08 f1 score for real classification, which was significantly lower result. I also tried keeping alpha equal to 1.0 but changing fit_prior to false. Again, the results were quite low compared to the best score. The f1 score for fake classification was 0.49 and the f1 score for real classification was 0.53.

Besides Naive Bayes I tried training my model with 4 different classifier algorithms. I achieved the following scores with them:

1. Stochastic Gradient Descent CLassifier (SGDClassifier) - 0.90 f1 score for fake classification and 0.76 f1 score for real classification. This score was achieved with the default algorithm parameters.
2. SVM (Support Vector Machine) Classifier (SVC) - 0.89 f1 score for fake classification and 0.74 f1 score for real classification. The score was achieved with the following algorithm parameters: SVC(gamma='scale', C=12, coef0=1, kernel='rbf').
3. Linear SVM classifier (LinearSVC) - 0.86 f1 score for fake classification and 0.73 f1 score for real classification. The score was achieved with the default algorithm parameters.
4. Random Forest Classifier - 0.81 f1 score for fake classification. The f1 score for real classification is not defined because there occurs a division by zero. The parameters used for this algorithm were:
   RandomForestClassifier(max_depth=4, random_state=2)

Conclusion

In the end it turned out that there are 3 machine learning algorithms that perform almost equally well. Those are Multinomial Naive Bayes, SVM, and Stochastic Gradient Descent. All of them are classifiers and achieved identical F1 scores. Out of the three Naive Bayes had the highest F1 score.

My algorithm had the following configuration in the end: MultinomialNB(alpha=1.0, fit_prior=True). It achieved f1 score of 0.91 for fake classification and 0.80 f1 score for real classification. The overall accuracy was 0.88.
Also I discovered that Naive Bayes is faster than the other algorithms and this was beneficial especially when running Grid Search. The SVM classifier, for example, was noticeably slower in both Grid Search and the computation of F1 score.

One preprocessing step that I did not implement is POS tagging. It would be a good idea to try it out as future work.Another thing that would be beneficial is to see how other classifiers perform on this problem. Although I have tried 5 different machine learning algorithms there are many more suggested classifiers. One example might be Passive Aggressive Classifier.

Having the right preprocessing techniques and correct algorithm parameters can significantly increase the algorithm’s performance. One step that I was not performing initially but increased my F1 score notably was word stemming. So it turns out that it is quite an important part of pre processing. Also Grid Search is a very useful technique and saves a lot of time when trying to find optimal algorithm parameters. Before using it I was experimenting randomly with different parameters for the different algorithms, which was not very effective. Another very helpful technique was to create my f1 score calculation in a separate method and then just pas machine learning algorithm as an argument to this method. This way I easily tested achieved f1 scores with different algorithms

References

[1] Boididou, C., Andreadou, K., Papadopoulos, S., Dang-Nguyen, D.-T., Boato, G., Riegler, M. and Kompatsiaris, Y. (2015). Verifying Multimedia Use at MediaEval 2015. [online] Available at: http://ceur-ws.org/Vol-1436/Paper4.pdf [Accessed 7 Jan. 2020].

[2] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

[3] Stahl, K. (2018). Fake news detection in social media. [online] Available at: https://www.csustan.edu/sites/default/files/groups/University%20Honors%20Program/Journals/02_stahl.pdf [Accessed 9 Jan. 2020].

[4] Aurélien Géron (2017). Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems. Sebastopol, Ca: O'reilly Media.

[5] Conroy, N.J., Rubin, V.L. and Chen, Y. (2015). Automatic deception detection: Methods for finding fake news. Proceedings of the Association for Information Science and Technology, 52(1), pp.1–4.

[6] Granik, M. and Mesyura, V. (2017). Fake news detection using naive Bayes classifier. 2017 IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON). [online] Available at: https://ieeexplore.ieee.org/abstract/document/8100379/citations#citations [Accessed 10 Jan. 2020].