Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Santiago Duque - sdp1n17
January 2020

Machine Learning Technologies - COMP3222
Coursework Report

1 Introduction and Data Analysis

1.1 Description of the problem

In social media, ongoing events are used to spread fake images, or images in a fake context. In the context of Twitter, some users would make use of the hashtag related to the event in question to gain more attention to their post, and accompany the hashtag with a piece of multimedia (image or video) unrelated to the event. Journalists and other professionals skimming social media sites for first-hand information on current events will encounter these fake posts, which can hinder their efforts in extracting truthful information. The problem was addressed in the 2015 MediaEval conference (Larson et al. 2015).

1.2 Datasets

The dataset provided is the Image Verification Corpus (Boididou et al. 2014) used by the aforementioned MediaEval conference, and it is divided into two sets - the training set and the test set. Unlike the original MediaEval dataset, however, this one does not have multimedia content, and instead has the image/video hyperlinks. The datasets are structured in tab-separated columns, with the following attributes: tweet ID, content of the tweet, user ID, image ID, username, timestamp and label (real or fake). The training set contains 14484 elements, and the test set 3782. The train-test split therefore amounts to roughly 80-20. The size of the training set is quite large, so good training can be obtained, perhaps at the expense of a substantial algorithm training time.

A problem that the datasets might have is that there are posts in many different languages, including non-Latin alphabets like Arabic or Mandarin, which will no doubt be an obstacle in the algorithm. The fact that more than one language is present makes it much harder to turn words into features to analyse. The data also includes not only non-Latin languages, but also special characters like emojis, and certain non-ASCII characters. This fact will hinder training, as it is much more difficult (in some cases implausible) to obtain features from these characters. The dataset contains multiple columns that will not be strictly useful when predicting whether a tweet is fake or genuine, such as the tweet ID, username of the author, etc. Hence why the algorithm will run only on the content of the tweet and its label. Both the training and test sets contain multiple instances of repeating tweets. These tweets are repeated dozens of times in some cases, which will definitely impact the features extracted from the data. Tweets also contain the hyperlinks to multimedia content, which also might influence the way features are selected.

1.3 Previous work

Several studies have applied user-based features to machine learning methods on social media. The medium that is worked with the most in these studies is Twitter, given the character limit of tweets (280, 140 before November 2017), which facilitates analysis and feature extraction.

An influential bot detection framework—called SentiBot—is described in (Dickerson et al. 2014). A diverse ensemble of user features existing in four categories were used: tweet syntax, tweet semantics, user behaviour, and network-centric properties.

There is some research on applying feature selection to machine learning methods on Twitter. One study (Ji et al. 2016) achieved an F1 score of 96.3% by establishing new features and implementing them through a Random Forest algorithm.

SentiBot, a bot-detection framework, defined in (Dickerson et al. 2014), uses four broad categories (tweet syntax, tweet semantics, user behaviour and network-centric properties) to detect posts made by bots.

Sentiment Analysis is also related to the problem at hand. (Neethu & Rajasree 2013) performed Sentiment Analysis on Twitter using different Machine Learning algorithms (Naive Bayes, SVM, Maximum Entropy Classifier and Ensemble Classifier) and comparing their performance. (Clark et al. 2015) uses natural language processing to detect posts made by bots. Most of these studies are focused on detecting bot spam, and not exactly "fake" posts as defined above, which is the niche that this application will attempt to solve.

2 Algorithm Design

2.1 Development Environment

This project was developed in Python 3.6.5, in the TensorFlow environment, using mainly SKLearn libraries, as well as Pandas and Numpy.

2.2 Pre-processing and feature selection

Pre-processing of data is a very useful step in Machine Learning algorithms, and it is essential to solving problems that are inherent with particular datasets, and also to achieve better results in general (Famili et al. 1997). In this algorithm, we carry out data pre-processing in the training set in different ways:

- Converting to lowercase: Words that have different cases are treated equally.
- Removing special characters: They affect negatively the performance of the algorithm, as they rarely add semantic value to the text.
- Lemmatisation: Reducing words to their core 'lemma' (dictionary root form), as to avoid creating different features for words of very similar semantic value (e.g. "building" and "buildings").
- Turning multiple spaces into a single space: Multiple spaces are almost never semantically relevant.

The Python regular expression library `re` enables us to perform these operations concisely and effectively.

2.3 Feature extraction

Feature extraction is the process whereby we reduce the number of features to be produced from input data, since it can contain redundancies and repetition. The following techniques are used in this algorithm:

- Bag of Words: The Bag of Words (BoW) model represents text as a multiset of its words, turning their frequency of appearance into features. This way, text features are converted into numerical ones. BoW ignores grammar and word order (but clearly not frequency).

- Identifying and removing 'stop-words': Stop words are the most common words in a language, (e.g., in English: "the", "is", "at", etc.), which almost never provide any semantic meaning or value in sentiment analysis (Rajaraman & Ullman 2011).

To this end, we use SKLearn's `CountVectorizer`, which provides parameter options for features and stop words.

2.4 Choice of algorithm

To solve this problem, several Machine Learning algorithms were considered. (Kowsari et al. 2019) collects several of the most popular and effective algorithms in text classification up to the year of its publication. The authors discuss the different algorithms and provide an overview of their advantages, limitations and real-world applications. Some of the most popular algorithms mentioned in the paper are SVM (Support Vector Machine), Decision Trees, Random Forests, CRF (Conditional Random Fields), DNN (Deep Neural Networks) and CNN (Convolutional Neural Networks).

After preliminary research and some baseline testing, the choice of algorithm for this problem was narrowed down to two: Random Forest and Support Vector Machine. The factors that influenced this decision were accuracy[1], runtime, complexity and parameter options. Regarding accuracy, two algorithms perform better than Naïve Bayes, and in the testing carried out in the beginning, not necessarily worse than other, more convoluted methods. Furthermore, they do not involve Deep Learning or complex hidden neural layers, which might have been effective, but would have definitely hindered development time and efficiency.

The choice to make now was between Random Forest and SVM. The theoretical basis was not to be of much help in this decision, since preliminary results were quite similar, and the algorithm now should be tailored to the particular dataset and problem at hand. Thus, further testing was required. As will be expanded on in the following section, different configurations of these two algorithms were tried. Following the results in runtime, but more importantly accuracy, it was determined that Random Forest was the better alternative.

2.5 Algorithm description

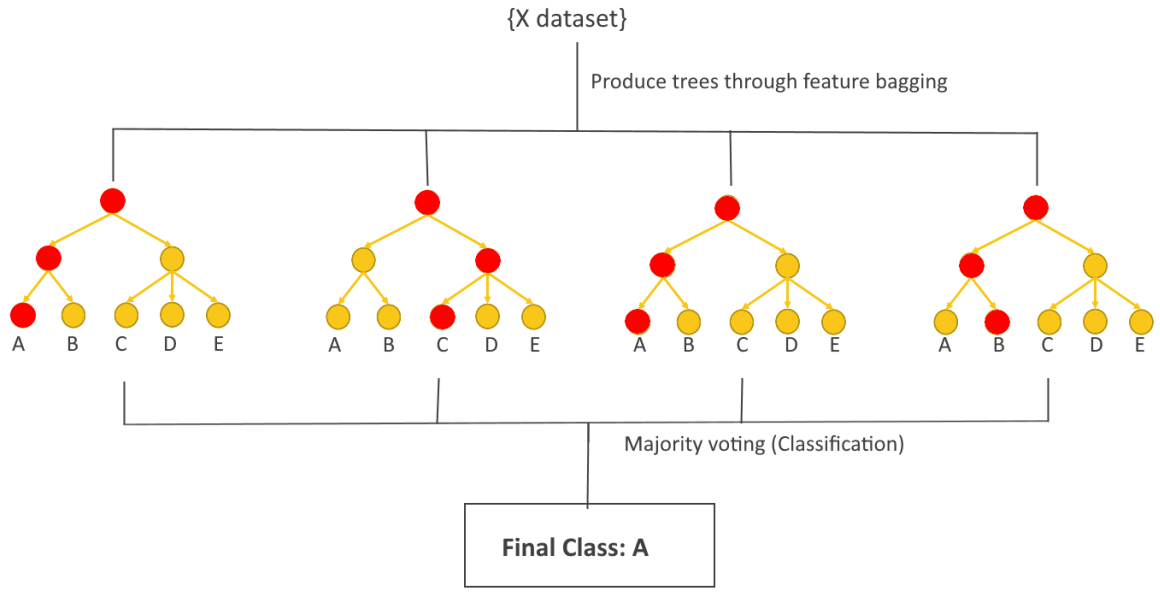
A Random Forest algorithm consists of N decision trees built out of N records from the dataset (Ho 1995). Decision Tree learning is an algorithm where a decision tree is used to represent elements and their characteristics. In this family of problems (classification), leaves represent categories / labels, and branches represent features that are used to classify each element in the corresponding category. A Random Forest is therefore an ensemble method that consists of multiple decision trees.

To form the Forest, *bagging* (**b**ootstrap **agg**regat**ing**) is used to produce multiple trees and vary the training data. Bagging selects a random sample from the training set, with replacement, N times, and produces trees to fit these samples. This technique is used to increase performance by making the average of the trees less susceptible to noise. After applying this method, a Forest is formed. One extra step is used by Random Forests, however: in the learning process of individual trees, at each split in a node, a random subset of features is selected, as to avoid correlation between trees for features that are very strong predictors (*feature bagging*). The reason why it is very desirable for Trees in the Forest to be as uncorrelated as possible, is that the average of the trees will outweigh the possible mistakes of individual trees.

[1]Henceforth referring to the "accuracy" score provided by `sklearn.metrics.accuracy_score`, which uses the Jaccard Score

Once the trees are created, they then vote on the category label that the particular element belongs to, and reach a consensus through majority.

Some advantages of using Random Forest is that it is not biased and uses majority voting, it is very stable, and works well if the data has missing or inaccessible values, such as the one in this particular dataset. It has disadvantages as well, mainly its very large complexity, which results in longer training time and computational usage than other algorithms.



A simple representation of a Random Forest algorithm.

The Random Forest algorithm is implemented through SKLearn's `RandomForestClassifier`, which includes options that allow us to perform hyperparameter tuning, like the number of trees generated or the number of features to consider when looking at the candidate split.

3 Evaluation

3.1 Stages of development

Before doing parameter tuning / grid-search for particular algorithms, a few standard ones were used to establish a baseline for tuning the parameters of the pre-processing and feature extraction. These algorithms are Random Forest Classifier, SVM Classifier and Gaussian Naive Bayes, with the default parameters provided by the SKLearn library. The following scores are the average of the three algorithms on each parameter setting.

In the first iteration of development, no pre-processing or feature selection or extraction of any sort was carried out. That is, algorithms were executed directly on the training and test sets without any alteration (other than numerical data categorization to allow the algorithm to run). The average score for this setting was 0.37. This is a very poor performance (worse than random, as there are only two categories), as expected, since the datasets need to be processed at least through some basic operations.

In the second iteration, pre-processing was performed as described in 2.1. This improved the average score to 0.46, which is still poor and worse than random, but a significant upgrade with regards to the previous iteration.

In the third iteration, feature extraction was applied as described in 2.2. The Bag of Words model was used, with a TF-IDF (Term Frequency-Inverse Document Frequency) converter. This step brought the average score up to 0.61. This was a significant improvement from the previous step, and the first iteration where the score was better than random, but it still was a rather mediocre performance, and it did not yet meet the expectations of the author.

These iterations were concerned with data pre-processing and feature handling. The following iterations will regard the choice of ML algorithm, and parameter tuning. In the first few of these iterations, parameter tuning was not performed exhaustively. That is, it was a case of manually changing parameters in the algorithm until a satisfying score was produced. Once the choice of algorithm was settled, parameter tuning was performed more comprehensively using grid search with cross-validation (provided by the homonymous SKLearn library `GridSearchCV`).

3.2 Different configurations

As mentioned in 2.3, the two main algorithms that were tested are Random Forest and SVM. The SKLearn libraries provide many different parameters for SVM, with many different options and therefore combinations of parameters. To find the optimal combination of parameters, involving C (regularisation parameter), type of kernel (linear, RBF or polynomial) and the gamma for the last two (kernel coefficient). Using a slightly modified version of the Grid Search with Cross Validation provided in this SKLearn tutorial, the following output was yielded:

```
Best parameters set found on development set:

{'C': 1, 'kernel': 'poly'}

Grid scores on development set:

0.328 (+/-0.000) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.328 (+/-0.000) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.681 (+/-0.124) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.328 (+/-0.000) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.741 (+/-0.087) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.681 (+/-0.124) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.730 (+/-0.086) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.730 (+/-0.090) for {'C': 1, 'kernel': 'linear'}
0.713 (+/-0.085) for {'C': 10, 'kernel': 'linear'}
0.701 (+/-0.088) for {'C': 100, 'kernel': 'linear'}
0.701 (+/-0.082) for {'C': 1000, 'kernel': 'linear'}
0.758 (+/-0.049) for {'C': 1, 'kernel': 'poly'}
0.722 (+/-0.063) for {'C': 10, 'kernel': 'poly'}
0.724 (+/-0.074) for {'C': 100, 'kernel': 'poly'}
```

The best parameter combination found was with a polynomial kernel. However, given the dimensionality of this problem, a linear kernel achieved almost the same score (and potentially even better, given the error margin provided by GridSearchCV), and in substantially less time. Thus, if SVM was to be used, a linear kernel was deemed more efficient, taking into account runtime and accuracy.

Random Forest has fewer combinations of parameters. `n_estimators` (number of trees in the forest) is probably the most influential parameter, but it was determined in the preliminary testing phases that any number from 500 to 1000 yielded the best results, without the need of exhaustive grid searching. The other relevant parameter is `max_features` (the number of features to consider when selecting the best split). In here, a simple manual grid search was conducted. With `n_estimators=500` and `random_state=0` (just the random seed for Tree generation), the following values were produced:

```
Accuracy with max_feature = 1 -> 0.7821571238348868
Accuracy with max_feature = 2 -> 0.7864181091877497
Accuracy with max_feature = 4 -> 0.7909454061251664
Accuracy with max_feature = 16 -> 0.8242343541944075
Accuracy with max_feature = 32 -> 0.8372836218375499
Accuracy with max_feature = 100 -> 0.8399467376830893
Accuracy with max_feature = 200 -> 0.8314247669773636
```

As we can see, the accuracy improves slightly the more features we consider. At around 100 features, the accuracy ceases to improve, probably due to overfitting. From 32 to 100, the improvement is minimal (0.00266), and the runtime increases substantially, so the final value selected for `max_features` is 32.

These values signify a hefty upgrade with respect to SVM. Even though SVM with a linear kernel is faster, Random Forest was finally selected due to its considerably better results.

The code that produces these values is included in the Appendix.

One strange thing to note is that, when taking into account *humor* labels as *fake*, accuracy decreased from 0.837 to 0.819. It is not known why this happens, as considering humor labels as fake actually improves performance in most of the other algorithms tested in the preliminary stages. Perhaps overfitting to the training data. The final decision was to do include 'humor' labels as 'fake', since it provides a more generalisable algorithm and a more faithful classification report, and the improved score might be a result of a chance occurrence in this particular algorithm.

3.3 Means of Evaluation

To compute the final scores, tools provided by the SKLearn Metrics library were used, mainly `confusion_matrix`, `classification_report` and `accuracy_score`. The reported final F1 score is the average obtained from the F1 scores of each label respectively.

```
Confusion matrix:
[[2179  367]
 [ 311  898]]

Classification report:
              precision    recall  f1-score   support

        fake       0.88      0.86      0.87      2546
        real       0.71      0.74      0.73      1209

    accuracy                           0.82      3755
   macro avg       0.79      0.80      0.80      3755
weighted avg       0.82      0.82      0.82      3755

Accuracy: 0.8194407456724367
```

Confusion matrix, Classification report and Accuracy score provided by the `sklearn_metrics` tools. The F1 score of *fake* labels is therefore 0.87, and the average total unweighted F1 score amounts to 0.80.

Overall, this is a satisfactory result, given the size of the data, and even though the classification is binary.

4 Conclusion

To summarise the project in a few words, we can state that Machine Learning is a powerful tool which, if used well, can perform tasks that humans could never dream of doing, and can potentially achieve incredible results.

4.1 Lessons learnt

This project has provided some interesting findings and insights. One of the things to be learned from implementing a Machine Learning algorithm is that there a very large number of parameters to be tuned, and therefore a great number of possible combinations. Not only the different algorithms, but perhaps most importantly the parameters within each algorithm and in some of the tools used in the previous steps (like the CountVectorizer). This means that there are a few ways of finding the optimal (or at least a decent enough) combination of algorithm and parameters. First, the simple naïve way: trial and error. This is clearly very time consuming and might lead nowhere if one does not know the underlying principles of the algorithms and their parameters. The other way is through a search heuristic - either an exhaustive grid search (much more time-consuming but will most likely find the optimal combination of parameters for an algorithm) or a randomised search (less time-consuming, but might miss an optimal combination).

Another important insight is that there are many nuances which are inherent to each dataset and problem. That is, one might follow a general set of rules for developing an algorithm, but when it comes to obtaining an algorithm with proper performance and accuracy, one must think about the characteristics of the dataset and the problem at hand, so as to be more efficient in processing the data and selecting an algorithm and its corresponding parameters.

Pre-processing is also a very important step, as we have seen during the development of the project. Without pre-processing, or even with insufficient pre-processing, the accuracy of the algorithm decreases dramatically. Hence the importance of performing pre-processing correctly and thoroughly.

4.2 Future improvement

There are certain things that could be done to improve the performance of algorithms for this particular problem. Perhaps lists of sentiment words gathered by third parties could be employed in detecting if a particular Tweet is fake or real.

Another possible improvement is implementing custom features, obtained from observing the datasets that we are working with. This should be done carefully, however, as to not induce overfitting to the current data, and trying to generalise by performing a careful analysis of the datasets.

5 Appendix

Code for the SVM Grid Search with Cross-Validation:

```
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]},
                    {'kernel': ['poly'], 'C': [1, 10, 100, 1000]}]

scores = ['precision', 'recall']

for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print()

    clf = GridSearchCV(
        SVC(), tuned_parameters, scoring='%s_macro' % score
    )
    clf.fit(training, trainingSet_labels)

    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
    print()
    print("Grid scores on development set:")
    print()
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params))
    print()

    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print()
    y_true, y_pred = testSet_labels, clf.predict(test)
    print(classification_report(y_true, y_pred))
    print()
```

Code for the Random Forest hyperparameter tuning:

```
max_features = [1, 2, 4, 16, 32, 100, 200]

for max_feature in max_features:

        classifier = RandomForestClassifier
        (n_estimators=500, random_state=0, max_features=16)

        classifier.fit(training, trainingSet_labels)
        prediction = classifier.predict(test)

        print(confusion_matrix(testSet_labels, prediction))
        print(classification_report(testSet_labels, prediction))
        accuracy = accuracy_score(testSet_labels, prediction)
        print("Accuracy with max_feature =", 16, "->", accuracy)
```

References

Boididou, C., Papadopoulos, S., Kompatsiaris, Y., Schifferes, S. & Newman, N. (2014), Challenges of computational verification in social media, SNOW II: Social News.
URL: *https://github.com/MKLab-ITI/image-verification-corpus*

Clark, E. M., Williams, J. R., Galbraith, R. A., Jones, C. A., Danforth, C. M. & Dodds, P. S. (2015), 'Sifting robotic from organic text: A natural language approach for detecting automation on twitter', *CoRR* **abs/1505.04342**.
URL: *http://arxiv.org/abs/1505.04342*

Dickerson, J. P., Kagan, V. & Subrahmanian, V. S. (2014), Using sentiment to detect bots on twitter: Are humans more opinionated than bots?, *in* '2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)', pp. 620–627.

Famili, A., Shen, W.-M., Weber, R. & Simoudis, E. (1997), 'Data preprocessing and intelligent data analysis', *Intelligent Data Analysis, vol. 1, no. 1, pp. 3-23, 1997* .
URL: *https://content.iospress.com/articles/intelligent-data-analysis/ida1-1-02*

Ho, T. K. (1995), 'Random decision forests'.
URL: *https://web.archive.org/web/20160417030218*

Ji, Y., He, Y., Jiang, X., Cao, J. & Li, Q. (2016), 'Combating the evasion mechanisms of social bots', *Computers and Security* **58**.

Kowsari, Meimandi, J., Heidarysafa, Mendu, Barnes & Brown (2019), 'Text classification algorithms: A survey', *Information* **10**(4), 150.
URL: *http://dx.doi.org/10.3390/info10040150*

Larson, M., Ionescu, B., Sjöberg, M., Anguera, X., Poignant, J., Riegler, M., Eskevich, M., Hauff, C., Sutcliffe, R., Jones, G. J., Yang, Y.-H., Soleymani, M. & Papadopoulos, S. (2015), Proceedings of the mediaeval 2015 multimedia benchmark workshop, Wurzen, Germany.

Neethu, M. S. & Rajasree, R. (2013), Sentiment analysis in twitter using machine learning techniques, *in* '2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)', pp. 1–5.

Rajaraman, A. & Ullman, J. D. (2011), *Data Mining*, Cambridge University Press, p. 1–17.