Fake News Detection Using Vectorization in Python

Marko Javorac
Software Engineering
Lakehead University
Thunder Bay, ON, CAN
mjavorac@lakeheadu.ca

Steven Mckenna
Software Engineering
Lakehead University
Thunder Bay, ON, CAN
spmckenn@laekehadu.ca

Peter Sertic
Software Engineering
Lakehead University
Thunder Bay, ON, CAN
pisertic@lakeheadu.ca

Abstract - This paper covers the process of exploring and implementing a fake news detection algorithm through vectorization in Python

I. Introduction

We begin our project by reviewing contemporary literature surrounding the topic of fake news detection and its surrounding issues with proposed solutions. We expand our domain knowledge by understanding the different methods and techniques currently used to mitigate and respond to fake news.

II. Literature review

Rapid evolution of telecommunications technology and internet connectivity provides massive access to knowledge and news which would have been previously impossible. This allows newly posted articles to be consumed by thousands almost instantly, and massive amounts of new information to be added to the internet every minute. Where previously the distribution of news was limited by large operating and distribution costs, the internet article has allowed small news organizations to reach wider audiences previously unattainable. With so much news, it can be difficult to access the credibility of all the information being dispersed to viewers[1]. As a result, many articles that are published online are erroneous, often intentionally, as a tool to spread misinformation to the masses. This often is used maliciously to shift public opinion on important areas of life, such as politics, education, business, and as such can result in real life consequences[1]. Giving the potential harm, it is vital that a system is created to reliably classify news articles as fake or not. Before starting work on a solution to this problem, it is important to analyse previous attempts at creating fake news classifiers. Prior to any models or solutions being proposed, it is important to generate and collect the data needed to develop accurate models. A variety of research covers the datasets themselves. Some have been assembled in a painstaking manner for the sole reason of fake news detection[6]. Research has also explored other data surrounding the news itself. Techniques can include the authors, articles hosted location, and reviews[7]. Certain techniques look at network metadata[3] for their analysis along with the linguistic content. A key area of discussion includes the ability to distinguish between user-generated content vs traditional media content. The proposed techniques for deception detection can range in application areas from the end-user to the content generators themselves[4]. Before analysing the data and building a model, pre-processing must be applied to ensure the model can be properly built.

Due to the binary nature of the classification, either fake or real, binary logistic regression seems to be a promising technique in the classification of fake news. NLP techniques must be utilized in order to create discriminants for the classifier model. Existing third party API's such as Alexa Page Rank API may be utilized for data pre-processing in order to determine page rank, while others may be used for sentiment analysis, and content structure error[1]. Other techniques involve *Context Free Grammar Trees*[5] which analyzes

deeper linguistic data such as grammatical structure and prepositions. Ogdol *et al.*[1] propose three main discriminants for the model: Content Structure Error; Sentiment Analysis; Page Rank. The content structure error discriminant shows to be the most promising in successfully classifying the news articles(0.9971 odds ratio), followed by page rank(0.7199 odds ratio)[1]. TF-IDF has been shown to be a promising feature selection technique for classifying fake news. It can classify the importance of a word in a giving document[8]. It is a better solution than TF as it reduces the importance of common words such as "the", "and", as it references the occurrence of that term among every single text document used, not just the document it belongs to[9]. This also allows less common words to be weighted more heavily, and results in a maximum accuracy of 89% when using a logistic regression classifier. The authors of this paper[9] also provide algorithms for web link parsing, analyzing article content/title/publication date which will be helpful for our solution.

Ahmed, H *et al.* propose sentiment analysis to be done through N-Gram analysis[2]. In this study, the authors examine using N-Gram analysis as a way to identify fake news articles. N-gram analysis in this case refers to sets of words with size N. The study examines 1-word, 2-word, 3-word, and 4-word groupings to see the commonality between real and fake articles. These are the frequencies of each of the most common groups of words for real and fake articles. The results are then used as inputs for the logistic regression classifier to build a model. As stated by the authors, a 89% accuracy rate was achieved when compared to methods using the same datasets. All of these papers implement some of these techniques for the classifier model and we can improve upon their solutions by implementing the pang rank[1], content structure error[1], sentiment analysis[2][8] discriminants into a single model.

Zhang and Cui[10] propose an alternative method described as a deep diffusive network. This technique takes into account the text itself and the accompanying metadata including the authors and the subject of the article. This unique method employs a dataset that includes all this information and is able to extract the *credibility* of the article based upon the *credibility* of each unique data type. As stated by the paper, this multidimensional method results in a 14.5% higher accuracy rate when compared to other state-of-the art classifiers.

The works above discussed different pre-processing techniques for discriminant identification and implementation in order to achieve the highest accuracy for a fake news detection model. Thus, our work will improve upon these models by implementing the most effective techniques into a single model in order to achieve the highest accuracy possible.

III. Proposed Model

Prior to the development of a model, any NLP-based project requires a large amount of data preprocessing.

Preprocessing

Standardization - all the text required for the model need to be lowercase

Lemmatization - This process reduces all text into their root words(geese -> goose) and removes extra letters applied by grammatical situations (cars -> car).

Stopword Removal - A variety of words are deemed low value and have little to no impact on the performance of a model. Examples include "is, the, at, ect…). By removing these words for the dataset, the resources and time required to develop the model is significantly reduced

Punctuation - similar to stopwords, punctuation adds no value to the model and should be removed

Normalization - although our dataset is composed of professionally written articles, a large variety of non-formal words are used. By normalizing the text, the model will be better able to handle (b4 -> before)

Number/Symbol Removal - Remove any numbers from text, as well as any symbols not considered punctuation.

Remove Hyperlinks - Online articles often contain links to other websites or articles within. These should be removed prior to model analysis.

Vectorization

Vectorization is a common technique used in NLP to be able to process the text in a more efficient manner by converting textual information into numerical vectors. Our method explores two variations of vectorization.

Count Vectorization

This method converts a collection of text data into a matrix of tokens and occurrences to extract a feature for prediction. This technique also handles core preprocessing by breaking the text into its individual words prior to its vector representation. Our implementation in *Count_Vectorizer.py* is built on the scikit-learn library for both the tokenization and feature extraction. After initial preprocessing, the *CountVectorizer()* is used to train the model while *MultinomialNM()* is deployed for classification. Preliminary results using this method show strong prediction accuracy of 95.2% (Fig1.1) prior to any validation techniques. Further validation such as K-Fold and cross testing with alternative datasets will be used to make the model more robust.

```
#Prediction based upon countvectorizer technqiue
clf = MultinomialNB()
clf.fit(count_train, y_train)
pred = clf.predict(count_test)
score = accuracy_score(y_test, pred)
print("accuracy:   %0.3f" % score)
```

```
accuracy:   0.952
```

```
#For confusion matrix
cm = confusion_matrix(y_test, pred, labels=[0,1])
print(cm)
plot_confusion_matrix(cm, classes=['FAKE', 'REAL'])
```

```
[[6630  356]
 [ 294 6190]]
Confusion matrix, without normalization
```

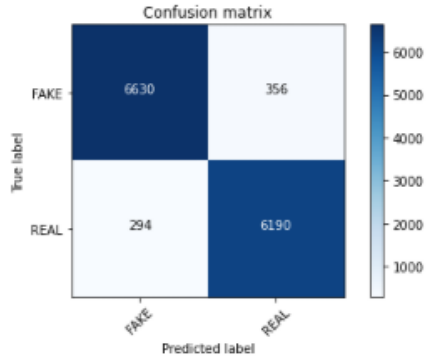


Fig1.1 - *Results of CountVectorization implementation with accompanying confusion matrix*

TF-IDF Vectorization

TF(Term Frequency) counts the frequency of a word as it appears in the current document over the entire amount of words in that document. IDF(Inverse Document Frequency) measures the appearance of a given word in every document over the complete set of all documents.

$$TFidf(t,d,D) = tf(t,d) \times idf(t,D)$$

$$tf(t,d0 = log(1 + freq(t,d))$$

$$idf(t,D) = log(\frac{N}{count(d \epsilon D : t \epsilon d)}).$$

$Where: t = the\ occurrence\ of\ a\ given\ word$
$D = the\ entire\ set\ of\ documents$
$d = a\ given\ document$

Implementation of this vectorization technique was done using a Binomial Naive Bayes classifier. The Scikit learn api was utilized to implement the TF-IDF vectorizer and the classifier. The dataset was composed of two dataframes of fake and real news. The datasets were appended and formatted to only contain the title, text, and label. The label values 0 and 1 represent fake and real news, respectively. The data was split into two dataset at a 0.33 test-train ratio. The model performed successfully with an accuracy of 93.6% when predicting values for the testset. The results were plotted using a confusion matrix, which can be seen in *Figure1.2*.

```
#implement model (MultinomialNB) (Multinomial Naive Bayes)
model = MultinomialNB()
model.fit(tfIdf_train, y_train)
pred = model.predict(tfIdf_test)
score = accuracy_score(y_test, pred)
print("accuracy:   %0.3f" % score)
cm = confusion_matrix(y_test, pred, labels=[0, 1])
print(cm)
plot_confusion_matrix(cm, classes=['FAKE', 'REAL'])
```

```
accuracy:   0.936
[[6561  425]
 [ 441 6043]]
Confusion matrix, without normalization
```

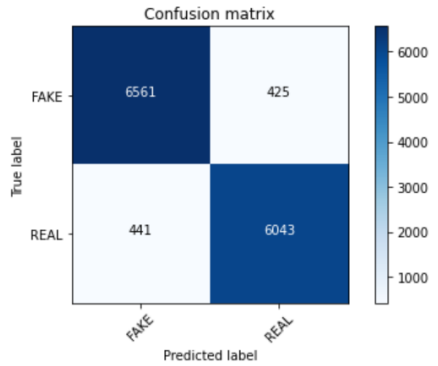


Figure 1.2: *Model implementation in python and confusion matrix comparing predicted labels to true labels.*

Glove

This method uses a fixed size array of numbers to represent a word. The values of the array are determined through a machine learning algorithm that compares how often words occur next to each other throughout the entire corpus. Determining the vectors for each word requires a pass through all text. This type of vectorization can be used alongside LSTM in order to direct the model.

LSTM

Another model that will be tested is the Long Short Term Memory recurring neural network. This model is ideal for natural language processing as it leverages neural network structure while retaining some context from the previous words used. As a string of text is processed, various sets information about the state of the data can be stored. Information such as the gender of the current subject (useful for predicting which pronoun to use) is and example of a state that this model will save. Additionally, this information can be forgotten or overwritten if the state changes (ie. if a new subject gender is specified). This method will be further investigated and results will be compared with those obtained from Count vectorization.

IV. Experimental Analysis

A. Dataset

The dataset used for the countVectorizer and TF-IDF method is the ISOT Fake News Dataset[11]. The dataset consists of 2 sets of data, one being "Real-News", and the other "Fake-News" which contain 21417 and 23481 rows of data respectively. The sets include,

1. **Title** - the titles of the articles.
2. **Text** - the text of the article
3. **Subject** - the subject of the article
4. **Date** - the date at which the article was posted

B. Computing Environment

The analysis is executed in a Google Collab virtual environment. As this is an online service with constantly evolving hardware, it is difficult to state which hardware is used. At the time of writing, the Memory provided is >12.72 GB with Disk Space of <107.77 GB

C. Evaluation Metric

To evaluate the performance of our fake news detection models, we will be employing a confusion matrix with the following dimensions:

True Positive(TP) - model predicted fake news, actually result is fake news
True Negative(TN) - model predicted true news, actual result is real news
False Positive(FP) - model predicted fake news, actual result is real news
True Positive(FN) - model predicted true news, actually result is fake news

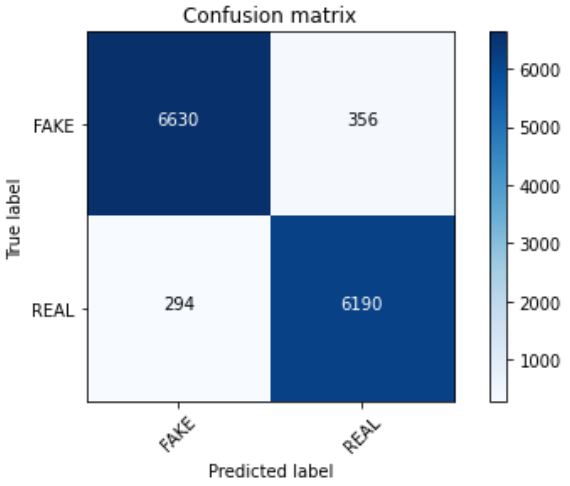
By using these values, we can form other metrics of evaluation including:

Sensitivity = TP / (TP + FN)
Specificity = TN / (FP + TN)
Precision = TP / (TP + FP)
Negative Prediction Value = TN / (TN + FN)
Accuracy = (TP + TN) / All Values
True Positive Rate(TPR) = TP / TP + FN
False Positive Rate(FPR) = FP / FP + TN

We can also use these false and positive rates to construct a ROC Curve. This metric can show us the performance of a classification model compared to other models.

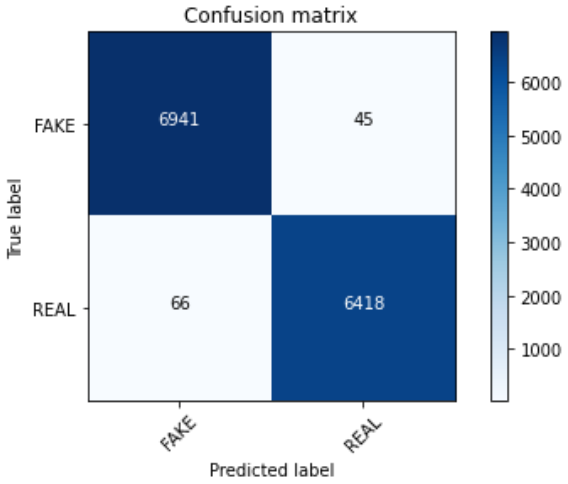
D. Initial Analysis

Our initial analysis combines the two techniques with the two different classifiers Naive Bayes and SVM. The figures below contain these results



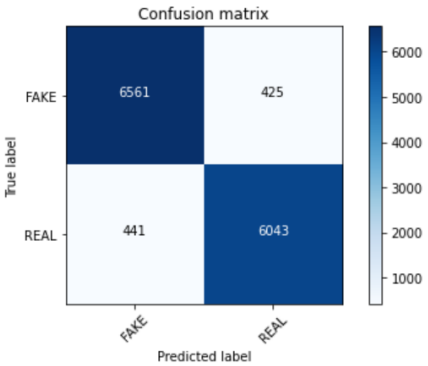
Measure	Value	Derivations
Sensitivity	0.9575	TPR = TP / (TP + FN)
Specificity	0.9456	SPC = TN / (FP + TN)
Precision	0.9490	PPV = TP / (TP + FP)
Negative Predictive Value	0.9547	NPV = TN / (TN + FN)
False Positive Rate	0.0544	FPR = FP / (FP + TN)
False Discovery Rate	0.0510	FDR = FP / (FP + TP)
False Negative Rate	0.0425	FNR = FN / (FN + TP)
Accuracy	0.9517	ACC = (TP + TN) / (P + N)

Fig 2.1 CountVectorization with Naives Bayes Classifier



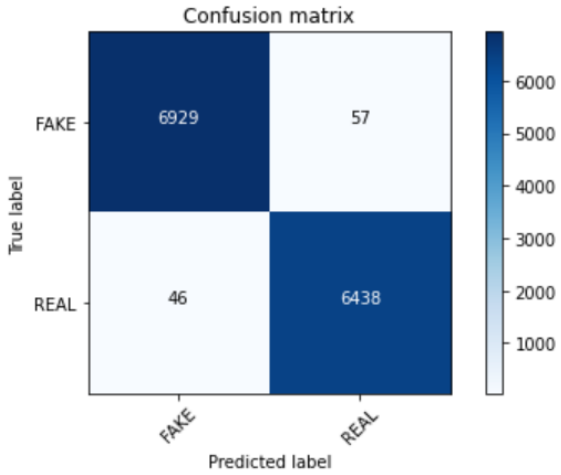
Measure	Value	Derivations
Sensitivity	0.9934	TPR = TP / (TP + FN)
Specificity	0.9912	SPC = TN / (FP + TN)
Precision	0.9918	PPV = TP / (TP + FP)
Negative Predictive Value	0.9929	NPV = TN / (TN + FN)
False Positive Rate	0.0088	FPR = FP / (FP + TN)
False Discovery Rate	0.0082	FDR = FP / (FP + TP)
False Negative Rate	0.0066	FNR = FN / (FN + TP)
Accuracy	0.9924	ACC = (TP + TN) / (P + N)

Fig 2.2 CountVectorization with SVM Classifier



Measure	Value	Derivations
Sensitivity	0.9370	TPR = TP / (TP + FN)
Specificity	0.9343	SPC = TN / (FP + TN)
Precision	0.9392	PPV = TP / (TP + FP)
Negative Predictive Value	0.9320	NPV = TN / (TN + FN)
False Positive Rate	0.0657	FPR = FP / (FP + TN)
False Discovery Rate	0.0608	FDR = FP / (FP + TP)
False Negative Rate	0.0630	FNR = FN / (FN + TP)
Accuracy	0.9357	ACC = (TP + TN) / (P + N)

Fig 2.3 TF-IDF Vectorization with Naives Bayes Classifier



Measure	Value	Derivations
Sensitivity	0.9934	TPR = TP / (TP + FN)
Specificity	0.9912	SPC = TN / (FP + TN)
Precision	0.9918	PPV = TP / (TP + FP)
Negative Predictive Value	0.9929	NPV = TN / (TN + FN)
False Positive Rate	0.0088	FPR = FP / (FP + TN)
False Discovery Rate	0.0082	FDR = FP / (FP + TP)
False Negative Rate	0.0066	FNR = FN / (FN + TP)
Accuracy	0.9924	ACC = (TP + TN) / (P + N)

Fig 2.4 TD-IDF Vectorization with SVM classifier

Results

After getting our results back from the testing, our high performance of >99% immediately raised concerns of overfitting. It is extremely unlikely that our initial run had this great of a performance so we began exploring validation techniques.

K- Fold

K fold validation is a method of detecting overfitting of a model. It involves dividing up the dataset into a number of equal sections called folds. The model is then trained on a subset of these folds and tested on the others. This method was used on the Naive Bayes classifier model that was applied to the count vectorization. Below are the results from the various combinations of folds. The accuracy results are relatively consistent across multiple folds. This method of validation will be applied to the more promising models. Some examples of the cross validation results are shown below.

```
Train on K1-4 Test on K5
accuracy:   0.953
Train on K2-5 Test on K1
accuracy:   0.947
Train on K1,3,4,5 Test on K2
accuracy:   0.952
Train on K1,2,4,5 Test on K3
accuracy:   0.953
Train on K1,2,3,5 Test on K4
accuracy:   0.949
```

Fig 2.5 Result from Naive Bayes Classifier using Count Vectorization

```
Train on K1-4 Test on K5
accuracy:   0.992
Train on K2-5 Test on K1
accuracy:   0.993
Train on K1,3,4,5 Test on K2
accuracy:   0.994
Train on K1,2,4,5 Test on K3
accuracy:   0.992
Train on K1,2,3,5 Test on K4
accuracy:   0.992
```

Fig 2.6 Results from SVM Classifier using Count Vectorization

Larger Datasets

Another dataset was combined with the previous data set to create an even larger dataset to work with. The new dataset adds 20760 new rows of data to the dataset, increasing the size by 46.2%. This was done to decrease the likelihood of overfitting of the dataset. The data was split into test and train sets at a ratio of 0.30. The final results can be seen in the figures below.

```
accuracy:   0.876
[[8958 1171]
 [1264 8305]]
Confusion matrix, without normalization
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

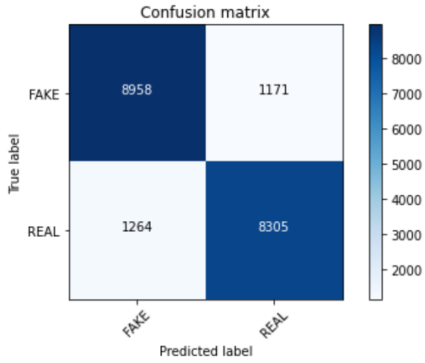


Fig 2.7 - TF-IDF using MNB classifier

```
accuracy:   0.964
[[9814  315]
 [ 401 9168]]
Confusion matrix, without normalization
```

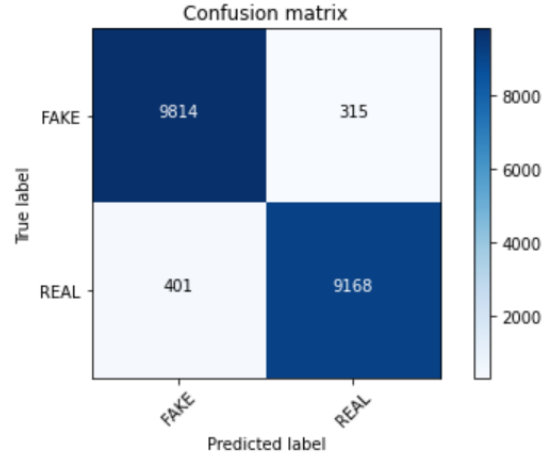


Fig 2.8 - TF-IDF using SVM classifier

```
accuracy:   0.937
[[9557  572]
 [ 678 8891]]
Confusion matrix, without normalization
```

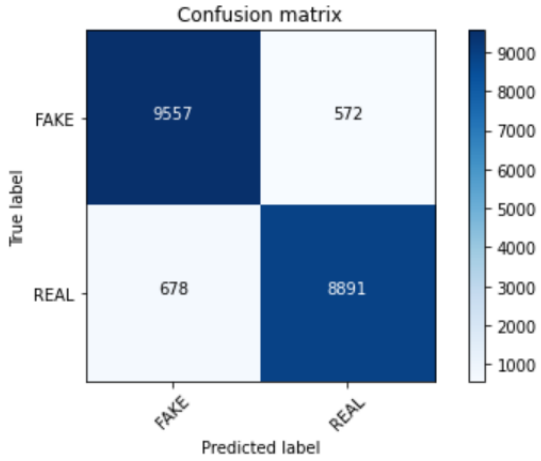


Fig 2.9 - TF-IDF using DT classifier

Decision Tree

Another method that can be used to get more accurate results is to test another classifier. A decision tree classifier is a widely used technique and can be rapidly deployed in our project.

```
#Decision Tree Classifier
```

```
#Train data on decision tree and fit
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(count_train, y_train)
```

```
#Predict
pred_dt = classifier.predict(count_test)
```

This allowed us to quickly run another test and get another high performing result(Fig2.10).

```
accuracy:   0.996
[[6952   34]
 [  22 6462]]
Confusion matrix, without normalization
```

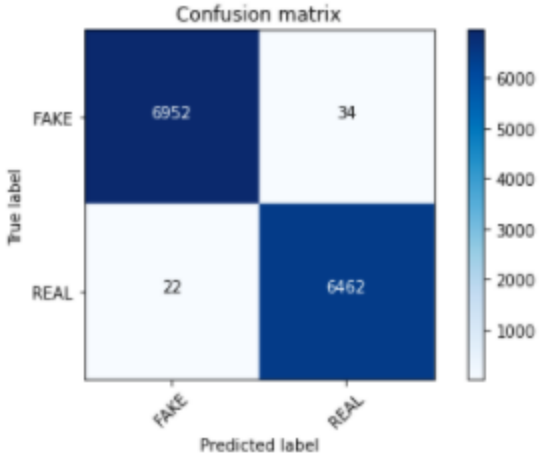


Fig 2.10 - Count Vectorization with Decision Tree classifier.

Combining Validation Methods

When we combine these 3 different methods as tools for validation, we are able to mitigate the issue of overfitting and get more accurate results. Some of our early testing have shown promising results. Below are the results of the k-fold validation using the combined dataset with TF-IID vectorization and Naive Bayes Classification.

```
Train on K1-4 Test on K5
accuracy:   0.874
[[5976  845]
 [ 808 5506]]
Confusion matrix, without normalization
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
Train on K2-5 Test on K1
accuracy:   0.878
[[5937  761]
 [ 847 5585]]
Confusion matrix, without normalization
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
Train on K1,3,4,5 Test on K2
accuracy:   0.882
[[6041  765]
 [ 782 5542]]
Confusion matrix, without normalization
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
Train on K1,2,4,5 Test on K3
accuracy:   0.876
[[5937  826]
 [ 798 5569]]
Confusion matrix, without normalization
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
Train on K1,2,3,5 Test on K4
accuracy:   0.876
[[5978  787]
 [ 844 5521]]
Confusion matrix, without normalization
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

Fig 2.11 - Applying K-Fold validation with larger dataset (TFID vectorization and Naive Bayes Classification)

IV. References

[1]Ogdol, Joseph Meynard & Samar, Bill-Lawrence. **Binary Logistic Regression based Classifier for Fake News.** Journal of Higher Education Research Disciplines, [S.l.], v. 3, n. 1, june 2018. ISSN 2546-0579

[2]Ahmed, H., Traore I., Saad S. (2017) **Detecting opinion spams and fake news using text classification** Security and Privacy Volume 1, Issue 1 https://onlinelibrary.wiley.com/doi/full/10.1002/spy2.9

[3] Conroy, N.K., Rubin, V.L. and Chen, Y. (2015), **Automatic deception detection: Methods for finding fake news**. Proc. Assoc. Info. Sci. Tech., 52: 1-4

[4] Chen, Y., Conroy, N.K. and Rubin, V.L. (2015), **News in an online world: The need for an "automatic crap detector"**. Proc. Assoc. Info. Sci. Tech., 52: 1-4.

[5]Feng, S., Banerjee, R., & Choi, Y. (2012, July). **Syntactic stylometry for deception detection**. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (pp. 171-175).

[6] Wang, W. Y. (2017). " **liar, liar pants on fire": A new benchmark dataset for fake news detection**. *arXiv preprint arXiv:1705.00648*.

[7]Natali Ruchansky, Sungyong Seo, and Yan Liu. 2017. **CSI: A Hybrid Deep Model for Fake News Detection.** In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM '17). Association for Computing Machinery.

[8]Ahmed, Hadeer & Saad, Sherif. (2017). **Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques**. (pp.127-138). 10.1007/978-3-319-69155-8_9.

[9]Al Asaad, Bashar & Erascu, Madalina. (2018). **A Tool for Fake News Detection.** (pp.379-386). 10.1109/SYNASC.2018.00064.

[10]Zhang, J., Cui, L., Fu, Y., & Gouza, F. B. (2018). **Fake news detection with deep diffusive network** model. *arXiv preprint arXiv:1805.08751*.

[11] Ahmed H, Traore I, Saad S. "**Detecting opinion spams and fake news using text classification**", Journal of Security and Privacy, Volume 1, Issue 1, Wiley, January/February 2018.