University of Washington, Seattle
CSE 415 Assignment 7 Part B
Autumn 2017

Report on Fake News Detection Exeperiment

February 19, 2018

1 Introduction

Fake news is basicly described as the news that does not actually tell the truth. In the advent of Internet era, the number of news has been exploding. People would try to write anything to attract more readers and hits. However, they can only forge some fake news when they do not have anything to write. And one of the most obvious things is that they would use some fancy headlines to get more readers since people can easily get caught with a shining headline.
Based on the classification given be *Claire Wardle* in her article *Fake news. Its complicated* published in *FirstDraftNews.com*, there are seven categories of fake news: False Connection, False Context, Manipulated content, Satire or Parody, Misleading Content, Imposter Content, Fabricated content. Some of them such as False Connection and Misleading Content are pretty popular nowadays, while a news with 100% false content is still rare.
Therefore, a fake news detection is expected to detect those seven kinds of news automatically.

2 Formulation

From my perspective, a typical formulation of a fake news detection problem should provide data, in the form of text, on the headline and text body. The additional information such as the source of the news would definitely help to judge the correctedness. So both training and testing data should provide as many articles as possible, which should consist of both true and false samples.

3 Techniques Used

Nowadays, the most common classification method is either the tree model or the neural network model. In this experiment, I decided to use XGBoost (Gradient Boosting) and Random Forest to classify the data. I chose not to use a neural network model because a RNN (Reconcurrent Neural Network) or CNN (Convolutional Neural Network) is a little bit difficult to achieve.
XGBoost (Gradient Boosting) is a machine learning algorithm that products a prediction model that combines weak prediciton models, which are typically decision trees. The idea of gradient boosting was first reported by *Leo Breiman*. The algorithm of XGBoost, similar to regular boositng, is to fit a residual function after the previous result. The usage of gradient descent is primarily on finding the factor for the current function.
Random Forest is also a tree-based machine learning algorithm. It was originally developed by *Tin Kam Ho*. Random Forest usually applies only about one third of the data to fit the model and then combine the models together for an ensemble model.
Since XGBoost is an iterative algorithm, it takes lots more time than Random Forest depending on the number of iterations set. Since both models fit multiple models then create an ensemble final model, the overfitting issue is not a big concern here. Both models are tree-based, so the structures for them are similar.

4 Training and Testing Data Used

For this fake news detection experiment, I chose the data from a competition called *Fake News Challenge*, which started on December 1st, 2016 and ended on June 15th, 2017. The data provided consists of training and testing data, which was derived from the Emergent Dataset created by *Craig Silverman.* Each sample data consists of the headline and the body text. The label is named stance, which consists of Agrees (The body text agrees with the headline), Disagrees (The body text disagrees with the headline), Discusses (The body text discuss the same topic as the headline, but does not take a position) and Unrelated (The body text discusses a different topic than the headline). The definition of each category shows that the label of this dataset covers the seven categories of fake news really well. Hence, I believe that this is a great dataset to practice the fake news detection algorithm.
According to the statistics given by the competition, there are 49972 samples in the training set with 73.1% of them is labelled as Unrelated, 17.8% are Discusses, 7.4% are Agrees and 1.7% are Disagrees. The test data contains 25,413 samples with 72.2% are Unrelated, 17.6% are Discusses, 7.5% are Agrees and 2.7% are Disagrees. The average number of characters of the article body in the training set is 2208, while the average number of characters of the article body in the test set is 2076.

5 Experiments

Since the test data seems to have a dimension problem when transform to a matrix used in XGBoost, I splitted the training data into training and validation set for this problem. The training set consists of 60% of the samples (29,983), while the validation set has the remaining (19,989). I also set the number of iterations as 1000 for the XGBoost to generate a more accurate model. I ran both algorithms on the data once, since, by theory, the resulting models should be similar no matter how many times I ran. The training time has a large discrepancy between these two models. Here is the setting for the training of XGBoost:

```
'max\_depth': 6, 'colsample\_bytree': 0.6, 'subsample': 1.0,
'eta': 0.1, 'silent': 1, 'objective': 'multi:softmax',
'eval\_metric':'mlogloss', 'num\_class': 4.
```

The number of iterations is set to 1000. And the parameter I set for Random Forest is

```
n\_estimators = 25.
```

The training of XGBoost algorithm took 8 minutes and 24 seconds, while that of Random Forest algorithm only took 23 seconds.

6 Results

The labels of samples in the dataset provide four categories. So the typical analysis for classification, which include the true positive and false negative, etc, does not suit this dataset really well. Instead, I will only use the f1 score and the confusion matrix to compare the performance.
The confusion matrix from XGBoost model:

Actual \ Predicted	0 (Unrelated)	1 (Discuss)	2 (Agree)	3 (Disagree)	Total
0 (Unrelated)	14,411	73	26	2	14,512
1 (Discuss)	272	3,141	178	36	3,627
2 (Agree)	164	253	996	70	1,483
3 (Disagree)	35	70	100	162	367
Total	14,882	3,537	1,300	270	19,989

The confusion matrix from Random Forest model:

Actual \ Predicted	0 (Unrelated)	1 (Discuss)	2 (Agree)	3 (Disagree)	Total
0 (Unrelated)	14,214	225	68	5	14,512
1 (Discuss)	537	2,785	253	52	3,627
2 (Agree)	264	301	820	98	1,483
3 (Disagree)	66	62	114	125	367
Total	15,081	3,373	1,255	280	19,989

The following table summarizes the F1 score and accuracy score of both methods

Score \ Method	XGBoost	Random Forest
F1 Score	0.936	0.898

7 Discussion

Fake news problem is a challenge in that it requires a very high level of text analysis to differentiate the truth and fakes. From statistics above, it is obvious that the XGBoost model has an apparent advantage in terms of the accuracy. If there are more data or more computation power available, we might be able to use the *Word2Vec* model provided by Google to generate better vector representation for each word in the text. Then it would be much easier to extract the features from these vectors and hence improve the performance of the model with these two algorithms.
The seven categories of fake news provide us with the idea to solve this problem. A few of them are solvable with just the headlines and article bodies. With the development of deep learning, we can use CNN or RNN to extract the features from the text, which can potentially help the machine understand the text and make a judgement. However, categories like Satire and Fabricated Content is really hard for machine to detect, as the articles themselves do not involve any logical error. So it would require extra information, logic and even semantic analysis to solve this problem.

8 Personal Retrospective

This assignment is the first time for me to apply the text analysis. I was able to use XGBoost and Random Forest to the data and generated a satisfying result in terms of the accuracy. Also, the preprocessing of the data is actually not that easy as it DOES require some idea to process it and get it ready for the model training. All in all, this is a great assignment for me to experience the whole process from finding a dataset to drawing the conclusion based on the analysis.

References

[1] Fake News Challenge,
`http://www.fakenewschallenge.org`

[2] Talos Targets Disinformation with Fake News Challenge Victory
`https://blog.talosintelligence.com/2017/06/talos-fake-news-challenge.html`

[3] XGBoost Python API Reference
`http://xgboost.readthedocs.io/en/latest/python/python_api.html`

[4] Samir Bajaj, *The Pope Has a New Baby! Fake News Detection Using Deep Learning*, 2017

[5] Claire Wardle, *Fake news. Its complicated*, February 16, 2017
`https://firstdraftnews.com/fake-news-complicated/`

Appendix A

```
import pandas as pd
import xgboost as xgb
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.cross_validation import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

#Prepare the data
body = pd.read_csv('/Users/leyang24kobe/Desktop/UW/
                    2017 Autumn/CSE 415/HW 7/
                    fnc-1-master/train_bodies.csv')
df = pd.read_csv('/Users/leyang24kobe/Desktop/UW/
                  2017 Autumn/CSE 415/HW 7/fnc-1-master/
                  train_stances.csv')
data = pd.merge(body, df, how='right', on='Body ID')

#Preprocess the data
res_map = {'unrelated': 0,
           'discuss': 1,
           'agree': 2,
           'disagree': 3}
def change(row):
    return res_map[row['Stance']]
data['Stance'] = data.apply(lambda row:change(row), axis = 1)
```

```
def cat_text(x):
    res = '%s %s' % (x['Headline'], x['articleBody'])
    return res
data['all_text'] = list(data.apply(cat_text, axis=1))

data_train, data_val = train_test_split(data,
                                         test_size = 0.4,
                                         random_state = 123)
train_y = data_train['Stance'].values
val_y = data_val['Stance'].values

#Vectorize the data
vectorizer = TfidfVectorizer()
total_train = vectorizer.fit_transform(data['all_text'])
data_train = vectorizer.transform(data_train['all_text'])
data_val = vectorizer.transform(data_val['all_text'])

#Fit XGBoost
dtrain = xgb.DMatrix(data_train, label=train_y)
dval = xgb.DMatrix(data_val, label=val_y)

params_xgb = {'max_depth': 6,
             'colsample_bytree': 0.6,
             'subsample': 1.0,
             'eta': 0.1,
             'silent': 1,
             'objective': 'multi:softmax',
             'eval_metric':'mlogloss',
             'num_class': 4
             }
evallist = [(dval, 'eval'), (dtrain, 'train')]
num_round = 1000

model = xgb.train(params_xgb, dtrain, num_round, evallist)

#Show only the first 20 and last 20 iterations of the training
[0] eval-mlogloss:1.28836    train-mlogloss:1.28521
[1] eval-mlogloss:1.20634    train-mlogloss:1.20083
[2] eval-mlogloss:1.13862    train-mlogloss:1.13052
[3] eval-mlogloss:1.07954    train-mlogloss:1.0695
[4] eval-mlogloss:1.0277     train-mlogloss:1.01573
[5] eval-mlogloss:0.983293   train-mlogloss:0.969288
[6] eval-mlogloss:0.943743   train-mlogloss:0.928074
[7] eval-mlogloss:0.909055   train-mlogloss:0.891555
[8] eval-mlogloss:0.878461   train-mlogloss:0.859332
[9] eval-mlogloss:0.850195   train-mlogloss:0.82946
[10]    eval-mlogloss:0.82587    train-mlogloss:0.803821
[11]    eval-mlogloss:0.803929   train-mlogloss:0.780282
```

```
[12]    eval-mlogloss:0.784203   train-mlogloss:0.758678
[13]    eval-mlogloss:0.766597   train-mlogloss:0.739572
[14]    eval-mlogloss:0.749791   train-mlogloss:0.721208
[15]    eval-mlogloss:0.73463    train-mlogloss:0.704632
[16]    eval-mlogloss:0.721311   train-mlogloss:0.690136
[17]    eval-mlogloss:0.708635   train-mlogloss:0.676286
[18]    eval-mlogloss:0.69718    train-mlogloss:0.663449
[19]    eval-mlogloss:0.686553   train-mlogloss:0.651935
                                 .
                                 .
                                 .
[980]   eval-mlogloss:0.201817   train-mlogloss:0.055087
[981]   eval-mlogloss:0.201721   train-mlogloss:0.054985
[982]   eval-mlogloss:0.201669   train-mlogloss:0.054891
[983]   eval-mlogloss:0.201646   train-mlogloss:0.054824
[984]   eval-mlogloss:0.201605   train-mlogloss:0.054738
[985]   eval-mlogloss:0.201538   train-mlogloss:0.054655
[986]   eval-mlogloss:0.201498   train-mlogloss:0.054583
[987]   eval-mlogloss:0.201431   train-mlogloss:0.054512
[988]   eval-mlogloss:0.201376   train-mlogloss:0.054418
[989]   eval-mlogloss:0.201303   train-mlogloss:0.054331
[990]   eval-mlogloss:0.201243   train-mlogloss:0.054248
[991]   eval-mlogloss:0.201224   train-mlogloss:0.054175
[992]   eval-mlogloss:0.201182   train-mlogloss:0.054074
[993]   eval-mlogloss:0.20108    train-mlogloss:0.053958
[994]   eval-mlogloss:0.201054   train-mlogloss:0.053875
[995]   eval-mlogloss:0.200932   train-mlogloss:0.05377
[996]   eval-mlogloss:0.200872   train-mlogloss:0.053676
[997]   eval-mlogloss:0.20085    train-mlogloss:0.053604
[998]   eval-mlogloss:0.200811   train-mlogloss:0.053513
[999]   eval-mlogloss:0.200732   train-mlogloss:0.053439

#Fit Random Forest
rf = RandomForestClassifier(n_estimators = 25)
rf = rf.fit(data_train, train_y)

#Evaluate XGBoost model
pred_xgb = model.predict(dval)

print("confusion matrix:")
print(metrics.confusion_matrix(val_y,pred_xgb))
print("F score: " + str(metrics.f1_score(val_y, pred_xgb, average='mic

confusion matrix:
[[14411    73    26     2]
 [  272  3141   178    36]
 [  164   253   996    70]
 [   35    70   100   162]]
```

```
F score: 0.936014808144

#Evaluate Random Forest model
pred_rf = rf.predict(data_val)

print("confusion matrix:")
print(metrics.confusion_matrix(val_y,pred_rf))
print("F score: " + str(metrics.f1_score(val_y, pred_rf, average='micro

confusion matrix:
[[14214   225    68     5]
 [  537  2785   253    52]
 [  264   301   820    98]
 [   66    62   114   125]]
F score: 0.897693731552
```