Kaggle Challenge: ISOT Fake News Dataset

Sian Chee Tong

We will be using the ISOT Fake News Dataset [Ahmed et al., 2017],[Ahmed et al., 2018] readily available from Kaggle website to perform statistical analysis and predictions. In particular, our dataset analysis is divided into two parts:

- Part 1: Exploratory analysis of the dataset
- Part 2: Fitting a binary logistic regression (LR) model for prediction of real and fake news

The dataset contains real and fake articles (news) collected from different sources throughout 2016 to 2017. There are two .csv files, named as "Fake.csv" and "True.csv". "Fake news" were flagged and collected from a fact-checking website called Politifact and Wikipedia. "Truthful news" were obtained by scraping from Reuters news website.

Commands for analyzing the data using Spark and compressed results of analysis are exported as .csv files to our local machine with WINSCP for plotting purposes. Here, we will be using Python for plotting which includes usage of libraries: "numpy", "pandas", "matplotlib" and "seaborn".

Exploratory Analysis Summary

Upon careful parsing of the .csv files, we have confirmed that there are 23481 records in "Fake.csv" and 21417 records in "True.csv". There are no missing entires in both dataset provided. Both dataset contain "title", "text", "subject" and "date" columns which represents news headline, content, subject and reported date respectively.

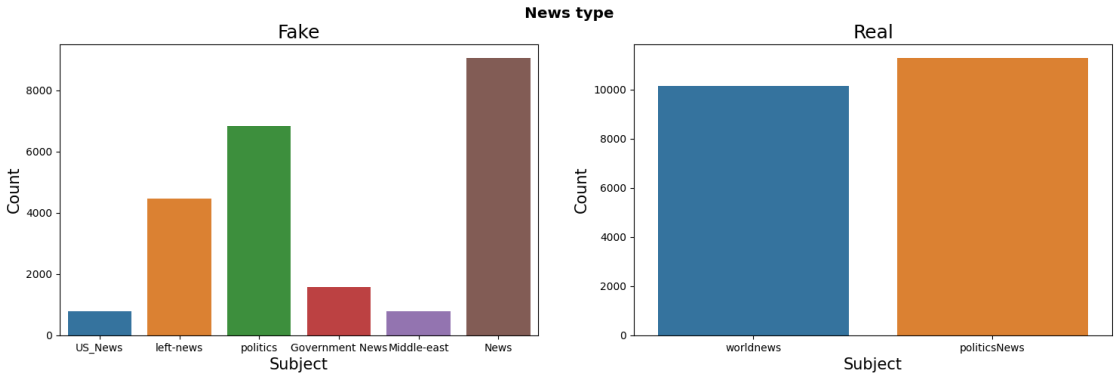

Figure 1: Distribution of news subject in the fake and real news dataset.

From Figure (1), we can see that "Fake" dataset categorized news into multiple different subjects, "US-News", "left-news", "politics", "Government News", "Middle-east" and "News" while "Real" dataset has only "worldnews" and "politicsNews". Apart from "News" subject, "politics" and "left-news" constituted the highest number of fake contents circulated in the internet. Real politics news are recorded more than world news. There is some ambiguity with regards to the column namings especially on how "News" subject is different from the remaining columns in the "Fake" dataset. Nevertheless, we can summarize news topics into two main categories: "Politics" and "General News".

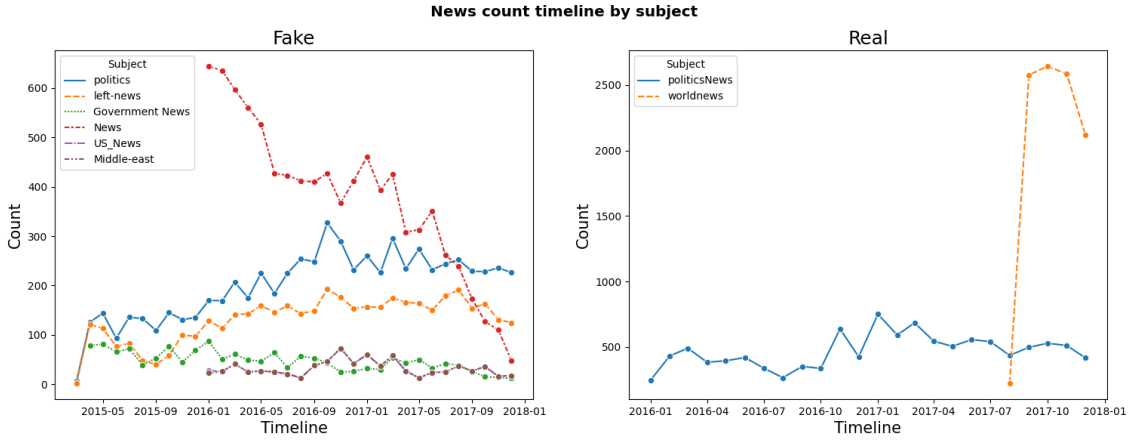

Figure 2: News count timeline by news subject in both fake and real datasets.

From the left plot of Figure (2), we see that fake left-news and politics tend to emerge unanimously throughout the the timeline of the fake dataset. We remark that that the 2016 US Presidential election occurs during October-November period. We witness the increase of left-wing and politic fake news which reaches to peak occurrences during that period. The rise in misinformation may have caused manipulation of people's political stances and hence the election outcomes.

Meanwhile, "News" and "Middle-east" categories only enters the dataset beginning of 2016. Interestingly, we see dramatic decrease in fake "News" category being flagged and collected. On the right plot, politics news were consistently collected throughout 2016 to 2017 but world news did not enter the real dataset until August 2017. This cast some doubt on the data retrieval method as the collection of news can be highly biased towards collecting political topics only.

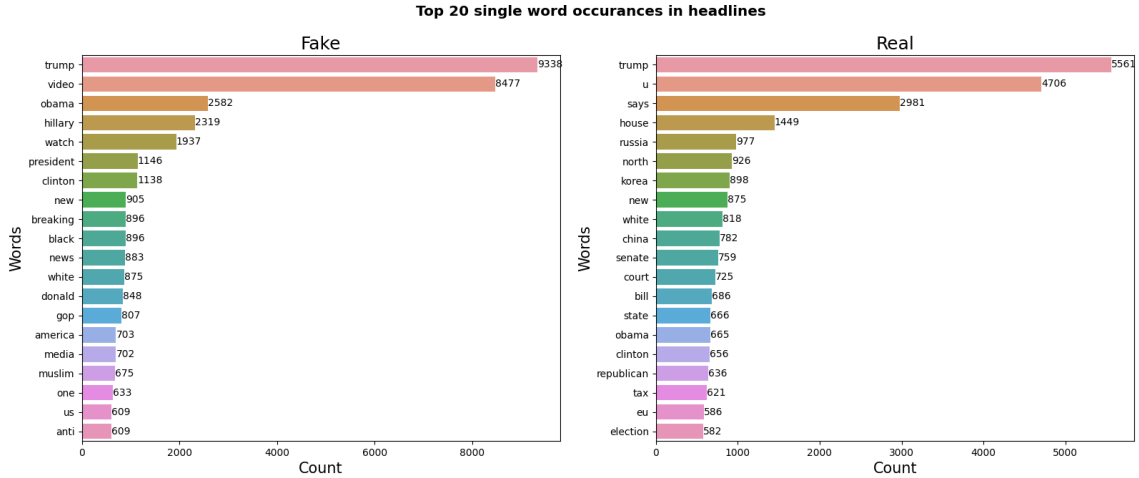

Figure 3: Top 20 single word occurances in the news headlines upon removal of insignificant symbols, punctuation corrections and stop words such as English pronouns and prepositions.

Figure (3) shows the word "Trump" emerges most of the time in both datasets, with 9336 mentions from fake dataset while 5561 mentions from real dataset. Also, we observe that real news tend to report a variety of topics revolving Russia, North Korea, China and US. Meanwhile, fake news widely reports local US news (most likely election related) with "Obama", "Hillary", "President" and other words topping the charts.

Model Fitting Summary

In the following, we make use of the Spark "mllib" library to classify real and fake news based on only the headline texts in the combined dataset.
First, we generate a random seeded training and test dataset pair with a train ratio of 0.8, leading to We verified that there is insignificant imbalance of classes in the training set. Upon train-test split, we construct

the pipeline which includes feature transformation and extraction and fitting our Binary Logistic Regression (LR) model (reference: [Nicolas, 2015]). The pipeline stages are as below:

1. Regular expression(Regex) tokenization: Split texts into sequence of individual terms and dropping insignificant symbols and punctuations.
2. Stop words removal: Exclude stop words which do not carry meaning to texts.
3. Word frequency vectors construction: Generate term frequency vectors (features) with "HashingTF" transformer.
4. Scaling of features: Down-weight the overly emphasized features which carries insignificant information and appears frequently in the texts with "IDF".
5. Fitting Logistic Regression (LR) classifier.

By default, the LR model does not include regularization and sets the threshold for classification prediction as 0.5. For model selection, we perform 3-fold cross-validation, using grid-search to find the best hyperparameters from the two types of hyperparameters:

- Type of regularization: Lasso, L_1 or Ridge, L_2.
- Degree of regularization: $\lambda = \{0, 0.01, 0.1, 0.5, 1\}$.

Evaluation of LR model

The top performing model corresponds to L_2 regularization with regularization paramater, $\lambda = 0.1$. The average cross-validation training Area Under Curve (AUC) is 0.9866 which is very close to 1. The following diagram shows the performance using the best LR model based on the receiver operating characteristics (ROC) curve.

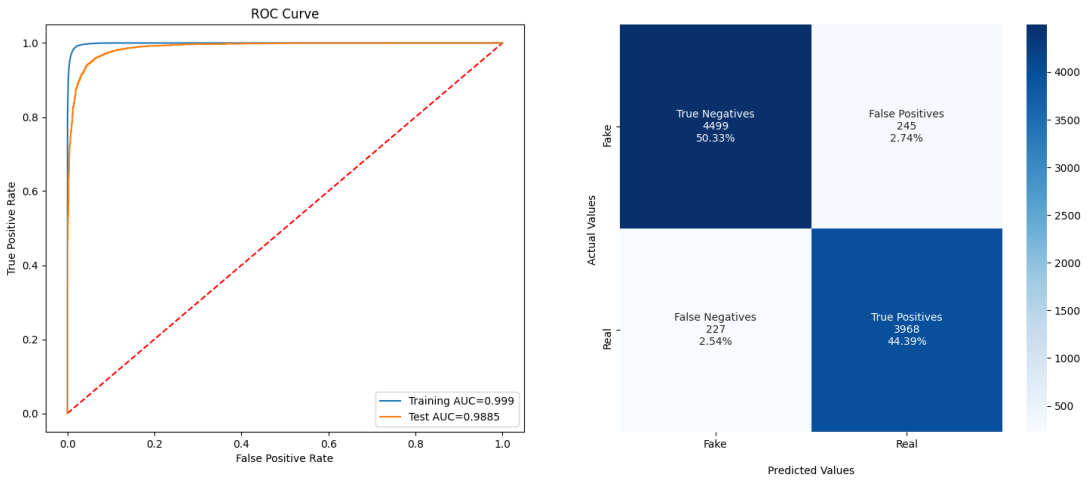

Figure 4: ROC plot for training and test for each threshold LR classification prediction value. The diagnonal indicates a random classifier which has no classifying "power".

From the ROC plot in figure (4), we see that the test ROC performs almost as good as the training ROC. In terms of AUC, the test AUC (0.9885) is only marginally lower than best training AUC (0.990). The curves demonstrates very high true positive rates and low false positive rates at most of the threshold values. The predictive performance is summarized using the confusion matrix in the right panel. The confusion matrix shows that fake news was predicted correctly in 4499 of the cases while real news was 3968 of the cases. 245 were wrongly classified as real news, while 227 were wrongly classified as fake news. Overall, our LR model managed to generate a test accuracy of 0.9472 on classifying fake and real news.

Commands for statistical analyses with Spark are as below:

```
import org.apache.spark.sql.functions._
import org.apache.spark.sql.Row
import org.apache.spark.ml.feature._
import org.apache.spark.ml.classification.{LogisticRegression, LogisticRegressionModel,
BinaryLogisticRegressionSummary}
import org.apache.spark.ml.tuning.{ParamGridBuilder, CrossValidator}
import org.apache.spark.ml.{Pipeline, PipelineModel}
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.ml.linalg.{Vector => MLVector}


////////////// Load data ////////////
val path_fake = "file:///home/user452/bd-sp-2017/data/ISOT_dataset/Fake.csv"
val path_true = "file:///home/user452/bd-sp-2017/data/ISOT_dataset/True.csv"

val df_fake = (spark.read.option("header", "true").option("multiline", "true")
.option("quote", "\"").option("escape", "\"").csv(path_fake)
.toDF("title", "text", "subject", "date"))
val df_true = (spark.read.option("header", "true").option("multiline", "true")
.option("quote", "\"").option("escape", "\"").csv(path_true)
.toDF("title", "text", "subject", "date"))

////////////// Exploratory analysis //////////////////////

////////////////////////// Check records and columns ////////////////////////
df_fake.count
df_fake.columns
//Fake: 23481 records and 4 columns (title, text, subject, date)
df_true.count
df_true.columns
//True: 21417 records and 4 columns (title, text, subject, date)

////////////////////////// Check if there are missing data ////////////////////////
df_fake.select(df_fake.columns.map(c => sum(col(c).isNull.cast("int")).alias(c)): _*).show
df_true.select(df_true.columns.map(c => sum(col(c).isNull.cast("int")).alias(c)): _*).show
// No missing entries

////////////////////////// Summary news subjects ////////////////////////
val fake_news_subject = df_fake.groupBy("subject").count()
val real_news_subject = df_true.groupBy("subject").count()

fake_news_subject.show
real_news_subject.show
// Report out for plotting
(real_news_subject.repartition(1).write.format("csv").option("header", "true")
.save("file:///home/user452/bd-sp-2017/data/ISOT_dataset/real_news_subject"))
(fake_news_subject.repartition(1).write.format("csv").option("header", "true")
.save("file:///home/user452/bd-sp-2017/data/ISOT_dataset/fake_news_subject"))


/////////////// Plot for monthly trends for real and fake news ////////////////////////
val df_fake_tmp = df_fake.withColumn("date",
```

```
date_format(to_timestamp($"date", "MMMM d, y"), "yyyy-MM"))
val df_true_tmp = df_true.withColumn("date",
date_format(to_timestamp($"date", "MMMM d, y"), "yyyy-MM"))

val fake_trend = df_fake_tmp.orderBy("date").groupBy("subject", "date").count()
val real_trend = df_true_tmp.orderBy("date").groupBy("subject", "date").count()

fake_trend.show
real_trend.show
// Report out for plotting
fake_trend.repartition(1).write.format("csv").option("header","true")
.save("file:///home/user452/bd-sp-2017/data/ISOT_dataset/fake_trend")
real_trend.repartition(1).write.format("csv").option("header","true")
.save("file:///home/user452/bd-sp-2017/data/ISOT_dataset/real_trend")


Calculate the word count for news headlines and get the top 20 words with counts
cleaning up punctuations and stop words

// Preprocess the data
val regexTokenizer = new RegexTokenizer().setInputCol("title").setOutputCol("words").
setPattern("\\W+")
val remover = new StopWordsRemover().setInputCol("words").setOutputCol("filtered_words")

val fake1 = regexTokenizer.transform(df_fake)
val fake2 = remover.transform(fake1)
val fake3 = fake2.select($"filtered_words", explode($"filtered_words"))
val headlines_fake = fake3.select("col").groupBy("col").count().sort($"count".desc).limit(20)
headlines_fake.show

val real1 = regexTokenizer.transform(df_true)
val real2 = remover.transform(real1)
val real3 = real2.select($"filtered_words", explode($"filtered_words"))
val headlines_true = real3.select("col").groupBy("col").count().sort($"count".desc).limit(20)
headlines_true.show

// Report out for plotting
(headlines_fake.repartition(1).write.format("csv").option("header", "true")
.save("file:///home/user452/bd-sp-2017/data/ISOT_dataset/headlines_fake"))
(headlines_true.repartition(1).write.format("csv").option("header", "true")
.save("file:///home/user452/bd-sp-2017/data/ISOT_dataset/headlines_true"))

// There are many stop words which we have to clean prior to fitting the model


///////// Prediction with machine learning model using mllib libraries ////////////////

// In order to perform logistic regression, we add label columns to both fake and true news
val df_fake2 = df_fake.withColumn("label", lit(0))
val df_true2 = df_true.withColumn("label", lit(1))


// Combine the data
val combinedData = df_fake2.union(df_true2)
```

```
// Using 80-20 train test split
val Array(train, test) = combinedData.randomSplit(Array(0.8, 0.2), 123)

// Check for class imbalance
train.groupBy("label").count().show()
test.groupBy("label").count().show()


// Preprocess the words we use the title only for predictions since the size is smaller
val regexTokenizer = (new RegexTokenizer().setInputCol("title").setOutputCol("words")
.setPattern("\\W+"))
val remover = new StopWordsRemover().setInputCol("words").setOutputCol("filtered_words")
val hashingTF = (new HashingTF().setInputCol("filtered_words")
.setOutputCol("raw_features").setNumFeatures(50000))
val idf = new IDF().setInputCol("raw_features").setOutputCol("features")
val lrWithRegularization = new LogisticRegression().setMaxIter(20)

// val lr = new LogisticRegression().setMaxIter(50)
// val pipeline = new Pipeline().setStages(Array(regexTokenizer, remover, hashingTF, idf, lr))
// val fittedpipeline = pipeline.fit(train)
// val testpredictions = fittedpipeline.transform(test)

////////// Using Cross validation to determine the best model ////////////

// Zero for L2 regularization else it is L1 regularization
val lambdas = Array(1, 0.5, 0.1, 0.01, 0)
val elasticNetParams = Array(0.0, 1.0)


// Compile into parameter grids
val paramGrid = (new ParamGridBuilder()
.addGrid(lrWithRegularization.regParam, lambdas)
.addGrid(lrWithRegularization.elasticNetParam, elasticNetParams)
.build)

// Add binary classification evaluator
val evaluator = new BinaryClassificationEvaluator()

// Set the pipeline
val pipeline = new Pipeline().setStages(Array(regexTokenizer, remover, hashingTF, idf,
lrWithRegularization))

// Crossvalidator to set the estimators, evaluator and estimator parameters and number of folds
val cv = (new CrossValidator()
.setEstimator(pipeline)
.setEvaluator(evaluator)
.setEstimatorParamMaps(paramGrid)
.setNumFolds(3)
.setParallelism(2)) // Evaluate up to 2 parameter settings in parallel

// Fit the model
val cvModel = cv.fit(train)

// // Save our model
// (cvModel.write.overwrite()
```

```
// .save("file:///home/user452/bd-sp-2017/data/ISOT_dataset/Output/model"))

// // Load our model
// PipelineModel.load("file:///home/user452/bd-sp-2017/data/ISOT_dataset/Output/model")

////////////////////////////// Compare the model results ////////////////////////////

// Get the average scores from k=3 folds CV for each models
val params2score = cvModel.getEstimatorParamMaps.zip(cvModel.avgMetrics)

// Get the average scores from k=3 folds CV for each models
params2score.foreach { case (params, score) =>
val lambda = params(lrWithRegularization.regParam)
val elasticNetParam = params(lrWithRegularization.elasticNetParam)
val l2Orl1 = if(elasticNetParam == 0.0) "L2" else "L1"
println(s"$l2Orl1, $lambda => $score")
}

val all_models = params2score.map{ case (params, score) =>
    val lambda = params(lrWithRegularization.regParam)
    val elasticNetParam = params(lrWithRegularization.elasticNetParam)
    val l2Orl1 = if(elasticNetParam == 0.0) "L2" else "L1"
    (l2Orl1, lambda, score)}

val all_models_df = sc.parallelize(all_models).toDF("Regularizer", "Parameter", "Average AUC")

val all_models_df2 = (all_models_df.withColumn("Average AUC", round($"Average AUC",4))
.orderBy(desc("Average AUC")))

// Report out results
(all_models_df2.repartition(1).write.format("csv").option("header", "true")
.save("file:///home/user452/bd-sp-2017/data/ISOT_dataset/CVresult"))

// View the best model with highest average cross-validation metric across fold
params2score.maxBy(_._2)._1
// We have L2 regularization with 0.1 penalty is the best model with 0.9866 AUC score


// Get the best model
val bestmodel = cvModel.bestModel

// Training summary by  getting the logistic regression stages' summary:
val trainingSummary = (cvModel.bestModel.asInstanceOf[PipelineModel].stages.last
.asInstanceOf[LogisticRegressionModel].binarySummary)

// ROC curves
val trainingROC = trainingSummary.roc

// Save our ROC curves result
(trainingROC.repartition(1).write.format("csv").option("header", "true")
.save("file:///home/user452/bd-sp-2017/data/ISOT_dataset/TrainingROC"))


// Best AUC results
trainingSummary.areaUnderROC
```

```
// 0.9990


// Get the predictions of the test dataset from the best model
val predictions = bestmodel.transform(test)

// Process to obtain accuracy of test set
val predictionAndLabels = predictions.select("prediction", "label").rdd.map{
    case Row(prediction:Double, label:Int) => (prediction.toDouble, label.toDouble)}


val metrics = new MulticlassMetrics(predictionAndLabels)


// Confusion matrix of predictions
metrics.confusionMatrix

// Prediction accuracy of the best model
metrics.accuracy


// AUC scores in test set
val AUC = evaluator.evaluate(predictions)
AUC

//Obtain Test ROC scores
val scoresLabels = predictions.select("probability", "label").rdd.map{
case Row(probability:MLVector, label:Int) => (probability(1), label.toDouble)
}


val bm = new BinaryClassificationMetrics(scoresLabels)

val rocArray = bm.roc.collect
// val falsePositives = rocArray.map{_._1}
// val truePositives = rocArray.map{_._2}


val bmout = bm.roc.toDF("FPR", "TPR")

(bmout.repartition(1).write.format("csv").option("header", "true")
.save("file:///home/user452/bd-sp-2017/data/ISOT_dataset/TestROC"))
```

Plotting commands for Python in local machine are as below:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import seaborn as sns
os.chdir(r"C:\Users\Tong\Documents\Academic\Spring term\Big data\bd-sp-2017\coursework")
################################## News subject plot ################################
fake_news_subject = pd.read_csv(r"fake_news_subject\fake_news_subject.csv")
real_news_subject = pd.read_csv(r"real_news_subject\real_news_subject.csv")
fig, ax = plt.subplots(1,2, figsize=(18,5))
```

```
sns.barplot(x='subject', y='count', data=fake_news_subject, saturation=0.7, ax = ax[0])
ax[0].set_xlabel("Subject", fontsize = 15)
ax[0].set_ylabel("Count", fontsize = 15)
ax[0].set_title('Fake', fontsize = 18)
sns.barplot(x='subject', y='count', data=real_news_subject, saturation=0.7, ax = ax[1])
ax[1].set_xlabel("Subject", fontsize = 15)
ax[1].set_ylabel("Count", fontsize = 15)
ax[1].set_title('Real', fontsize = 18)
fig.suptitle("News type" ,
fontsize = 'x-large' ,
fontweight = 'bold' )
fig.savefig('Newstype.png')
plt.close(fig)
################################## News timeline plot ################################
time_plot_true = pd.read_csv(r"real_trend\real_trend.csv")
time_plot_fake = pd.read_csv(r"fake_trend\fake_trend.csv")
# Convert to datetime format
time_plot_fake['date'] = pd.to_datetime(time_plot_fake['date'], format='%Y/%m')
time_plot_true['date'] = pd.to_datetime(time_plot_true['date'], format='%Y/%m')
fig, ax = plt.subplots(1,2, figsize=(18,6))
sns.lineplot(x='date', y='count', hue='subject', style= 'subject', marker="o",
data=time_plot_fake,ax=ax[0])
ax[0].set_xlabel("Timeline", fontsize = 15)
ax[0].set_ylabel("Count", fontsize = 15)
ax[0].set_title('Fake', fontsize = 18)
ax[0].legend(title = "Subject", loc='upper left')
sns.lineplot(x='date', y='count', hue='subject', style= 'subject', marker = "o",
data=time_plot_true, ax=ax[1])
ax[1].set_xlabel("Timeline", fontsize = 15)
ax[1].set_ylabel("Count", fontsize = 15)
ax[1].set_title('Real', fontsize = 18)
ax[1].legend(title = "Subject")
fig.suptitle("News count timeline by subject" ,fontsize = 'x-large', fontweight = 'bold')
fig.savefig('Newstime.png')
plt.close(fig)
################################## Top 20 words in headlines plot ################################
headline_true = pd.read_csv(r"headlines_true\HeadlinesTrue.csv")
headline_fake = pd.read_csv(r"headlines_fake\HeadlinesFake.csv")
fig, ax = plt.subplots(1,2, figsize=(19,7))
sns.barplot(x='count', y='col', data=headline_fake, orient='h', saturation=0.7, ax = ax[0])
ax[0].set_xlabel("Count", fontsize = 15)
ax[0].set_ylabel("Words", fontsize = 15)
ax[0].set_title('Fake', fontsize = 18)
ax[0].bar_label(ax[0].containers[0])
sns.barplot(x='count', y='col', data=headline_true, orient='h', saturation=0.7, ax = ax[1])
ax[1].set_xlabel("Count", fontsize = 15)
ax[1].set_ylabel("Words", fontsize = 15)
ax[1].set_title('Real', fontsize = 18)
ax[1].bar_label(ax[1].containers[0])
fig.suptitle("Top 20 single word occurances in headlines" ,
fontsize = 'x-large' ,
fontweight = 'bold')
fig.savefig('Newsheadlines.png')
plt.close(fig)
################################## ROC plot ################################
```

```
train_roc = pd.read_csv(r"TrainingROC\TrainingROC.csv")
test_roc = pd.read_csv(r"TestROC\TestROC.csv")
# Confusion matrix results
cf_matrix=np.array(([4499.0,245.0],[227.0,3968.0]))
group_names = ['True Negatives','False Positives','False Negatives','True Positives']
group_counts = ["{0:0.0f}".format(value) for value in
cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
fig, ax = plt.subplots(1,2, figsize=(18,7))
ax[0].plot([0, 1], [0, 1], 'r--')
ax[0].plot(np.array(train_roc['FPR']), np.array(train_roc['TPR']),
label="Training AUC="+str(0.9990))
ax[0].plot(np.array(test_roc['FPR']), np.array(test_roc['TPR']),
label="Test AUC="+str(0.9885))
ax[0].set_xlabel('False Positive Rate')
ax[0].set_ylabel('True Positive Rate')
ax[0].set_title("ROC Curve")
ax[0].legend(loc=4)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues', ax=ax[1])
ax[1].set_xlabel('\nPredicted Values')
ax[1].set_ylabel('Actual Values ');
ax[1].xaxis.set_ticklabels(['Fake','Real'])
ax[1].yaxis.set_ticklabels(['Fake','Real'])
plt.savefig('ROC_CM.png')
plt.close(fig)
```

References

H. Ahmed, I. Traore, and S. Saad. Detection of online fake news using n-gram analysis and machine learning techniques. In *International conference on intelligent, secure, and dependable systems in distributed and cloud environments*, pages 127–138. Springer, 2017.

H. Ahmed, I. Traore, and S. Saad. Detecting opinion spams and fake news using text classification. *Security and Privacy*, 1(1):e9, 2018.

P. R. Nicolas. *Scala for machine learning*. Packt Publishing Ltd, 2015.