Analysis of F-16 Accidents by NLP

Gursev Pirge and Alp Pirge

This article involves the use of Natural Language Processing (NLP), with the target of analyzing the causes of the F-16 fighter aircraft accidents and incidents between 1979 and 2019 in the US Air Force (USAF). We used the data set provided by F-16.net, which gave us the dates, type and the accident report. In the previous paper and in the very first paper, the analysis focused on both civilian and military aircraft. In this study, we decided to focus on the F-16 Fighting Falcon, which is operated by 26 air forces around the world since 1978, over 4600 have been built and still widely used globally. It was the primary fighter aircraft for the USAF for a long time, experienced combat globally, a single-engine and single-pilot airplane, has the tendency to be unstable etc. All these factors contributed to the 512 accidents/incidents (of which we used 500) for over 42 years.

Wikipedia's definition is: "Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. The result is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them. The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves."

This paper comprises two parts. In the first part, we handled data cleaning and prepared various Word Clouds (by using the accident reports) in order to analyze the F-16 accidents. Wikipedia definition for a word cloud is as follows: "A tag cloud (word cloud or wordle or weighted list in visual design) is a novelty visual representation of text data, typically used to depict keyword metadata (tags) on websites, or to visualize free form text. Tags are usually single words, and the importance of each tag is shown with font size or color. This format is useful for quickly perceiving the most prominent terms to determine its relative prominence. Bigger term means greater weight."

In the second part, the focus is on using NLP techniques with the aim of finding details in the accident reports in order to predict whether the pilot was killed during the accident. The text files were converted into numeric feature vectors to run machine learning algorithms. Here, we tried to compare the performances of for CountVectorizer and TfidfVectorizer. CountVectorizer converts a collection of text documents to a matrix of token counts, whereas TfIdfVectorizer transforms text to feature vectors that can be used as input to estimator. The second section may be considered as an application of sentiment analysis. Sentiment analysis (or opinion mining) is a NLP technique used to determine whether data is positive, negative or neutral. We used different machine learning algorithms to get more accurate predictions. The following classification algorithms have been used: Logistic Regression, Naive Bayes, Support Vector Machine (SVM), Random Forest and Ada Boosting.

We tried to concentrate on the following issues:

- What are the major causes of the military fighter aircraft accidents?
- Considering that F-16 is still in service after 42 years, is there a trend in the number of accidents by the year, month, combat action?
- Comparison of the performance of classification algorithms.

Data Cleaning

The first step is to import and clean the data (if needed) using pandas before starting the cluster analysis.

```
df.sample(8)
```

	Date	Type	Details	Fatality
202	Oct-09	F-16D Block 25E	Made an emergency landing short of the runway ...	0
235	Nov-14	F-16C Block 30B	Aircraft had not completed modifications to QF...	1
76	Jan-84	F-16A Block 15A	\t\nWas lost in a training accident in mounta...	0
441	Feb-93	F-16D Block 40H	\t\n\t\nDestroyed near Eielson AFB, Alaska. B...	1
360	Sep-94	F-16C Block 40C	\t\n\t\nCrashed at about 11:00 hours in a cor...	0
176	Sep-88	F-16C Block 25D	\t\nHit the ground near Sumter, South Carolin...	1
80	Jul-96	F-16A Block 15B ADF	\t\n\t\nThe aircraft crashed on the runway. I...	0
216	Jan-91	F-16C Block 25F	Crashed 75 miles north of Las Vegas, in the Ne...	0

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Date      500 non-null    object
 1   Type      500 non-null    object
 2   Details   492 non-null    object
 3   Fatality  500 non-null    int64 
dtypes: int64(1), object(3)
memory usage: 15.8+ KB
```

According to the dataset, there were only 8 missing reports, so we dropped those columns.

After that, we added separate “Year” and “Month” columns from the “Date” column. There were 16 unknowns, so we got rid of them.

```
df['Year'] = df['Date'].astype(str).str[4:6]
```

```
df.sample(3)
```

	Date	Type	Details	Fatality	Year	Month
454	Nov-05	F-16C Block 50B	aircraft made emergency landing ninoy aquino m...	0	05	Nov
485	Jan-99	F-16c Block 50P	went mountainous forest near city kamaishshi n...	0	99	Jan
288	Jan-91	F-16C Block 30F	shot desert storm combat loss number desert st...	0	91	Jan

```
df['Month'] = df['Date'].astype(str).str[0:3]
```

```
df.sample(3)
```

	Date	Type	Details	Fatality	Year	Month
419	Feb-94	F-16C Block 40G	heavily damaged porotoz slovenia operation den...	0	94	Feb
278	Feb-91	F-16C Block 30E	crashed diyarbakir turkey engine exploded ref...	0	91	Feb
443	Feb-00	F-16D Block 40J	crashed eight kilometre north donaldsonville g...	0	00	Feb

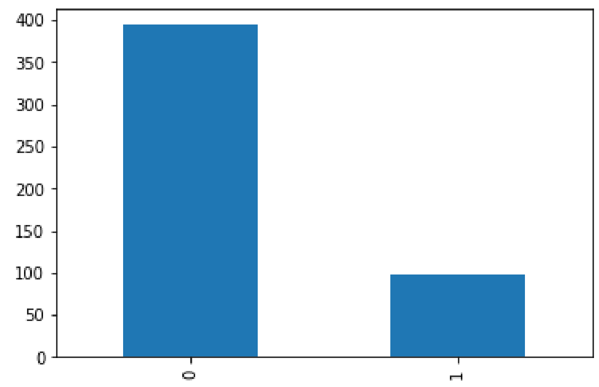
In the next step, namely text mining, the steps were: tokenizing, removal of punctuations and stopwords, lemmatizing and then joining as a string.

When the “Fatality” column was checked, it was observed that (killed/total accident) ratio was 98/492 = 0.1992 – almost one-fifth of the pilots were killed during an accident.

```
df['Fatality'].value_counts()
```

```
0    394
1     98
Name: Fatality, dtype: int64
```

```
df['Fatality'].value_counts().plot.bar();
```



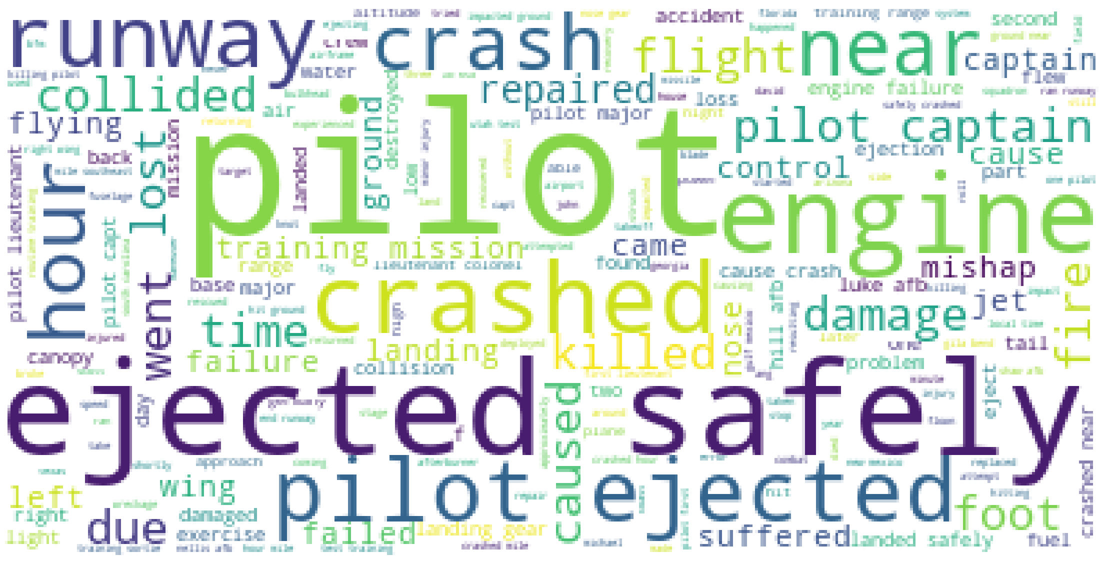
Word Cloud – Accident Reports

Before getting the WordCloud, we wanted to get the most common words to have a feeling about the contents of the accident reports.

```
from collections import Counter
most_common_words = Counter(" ".join(df["Details"]).split()).most_common(20)
most_common_words
```

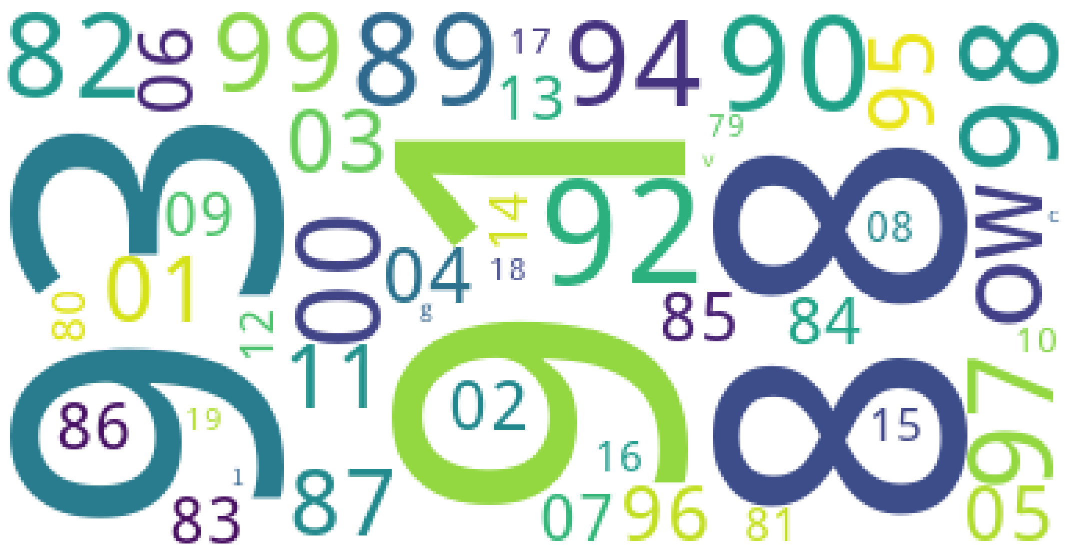
```
[('aircraft', 532),
 ('pilot', 517),
 ('ejected', 220),
 ('safely', 201),
 ('crashed', 177),
 ('engine', 172),
 ('afb', 150),
 ('runway', 120),
 ('hour', 116),
 ('near', 111),
 ('training', 101),
 ('crash', 98),
 ('mile', 96),
 ('landing', 91),
 ('captain', 91),
 ('cause', 82),
 ('ground', 80),
 ('flight', 76),
 ('failure', 73),
 ('gear', 70)]
```

As expected, aircraft was the most common word, although it gives no idea about the causes of accidents. The Word Cloud below gives details about the accident reports of F-16 aircraft.



Word Cloud – Years

1991 (32) and 1993(27) were the years with the most accidents. Remember, 1991 was the year of the Desert Storm, so there were a lot of combat action, too much stress on the pilots and the maintenance crew, flying on odd hours and the aircraft started to get older.



A quick note here, when working with numbers, all the numbers should be counted, converted to strings and then generate_from_frequencies method will produce the WordCloud.

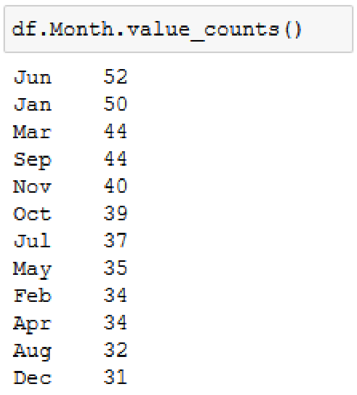
```
counts = df['Year'].value_counts()
```

```
counts.index = counts.index.map(str)
```

```
wordcloud = WordCloud(background_color="white").generate_from_frequencies(counts)
plt.figure(figsize=(15,15))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Word Cloud –Months

This may be a rough analysis, but the results show that June and January are the riskiest months for accidents. The difference between the top of the list and the bottom is huge.



```
df.Month.value_counts()
```

```
Jun    52
Jan    50
Mar    44
Sep    44
Nov    40
Oct    39
Jul    37
May    35
Feb    34
Apr    34
Aug    32
Dec    31
```

Sep
Aug
Jun
Oct
Jul
Mar
May
Apr
Jan
Nov
Dec
Feb

<u>Fatality Classification/Estimation</u>

Machine learning algorithms most often take numeric feature vectors as input, so the first step will be converting text files into numeric feature vectors in order to be able to run the machine learning algorithms. Text vectorization converts each document into a numeric vector.

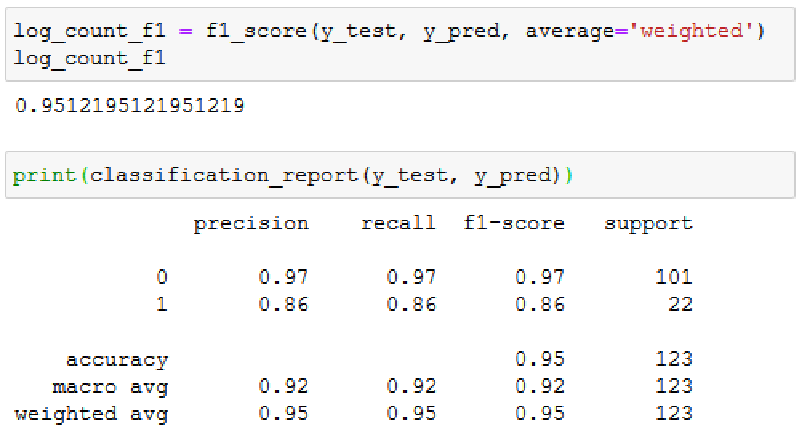
Scikit-learn's <u>CountVectorizer</u> provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary. An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document.

Word counts are a good starting point, but they are considered very basic. An alternative is to calculate word frequencies, and by far the most popular method is called <u>TF-IDF</u>. This is an acronym than stands for "*Term Frequency – Inverse Document*" Frequency which are the components of the resulting scores assigned to each word. The <u>TfidfVectorizer</u> will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents.

Next step will be predicting the fatality by using five different algorithms.

Logistic Regression

Classification report and the f1-score of the Logistic Regression classification for the data prepared by using the CountVectorizer method is shown below:



```
log_count_f1 = f1_score(y_test, y_pred, average='weighted')
log_count_f1
```

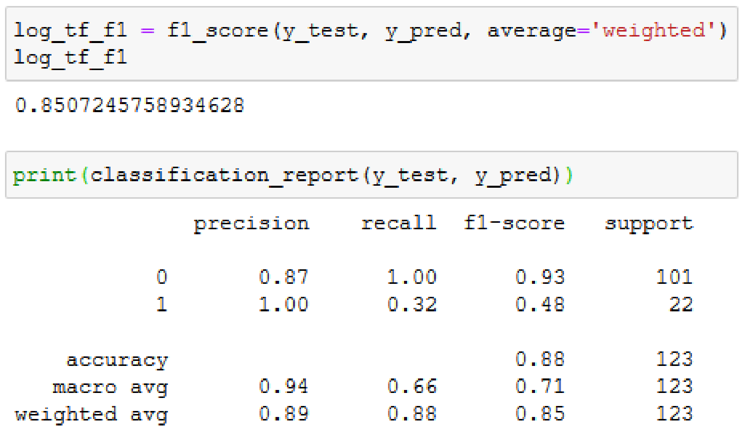
```
0.9512195121951219
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       101
           1       0.86      0.86      0.86        22

    accuracy                           0.95       123
   macro avg       0.92      0.92      0.92       123
weighted avg       0.95      0.95      0.95       123
```

f1 Score is defined as the weighted average of Precision and Recall and it takes both false positives and false negatives into account. Especially if there is an uneven class distribution (in our case, 394 to 98), f1 score is considered more useful than the accuracy.

Classification report and the f1-score of the Logistic Regression classification for the data prepared by using the TfIdfVectorizer method is shown below:



```
log_tf_f1 = f1_score(y_test, y_pred, average='weighted')
log_tf_f1
```

```
0.8507245758934628
```

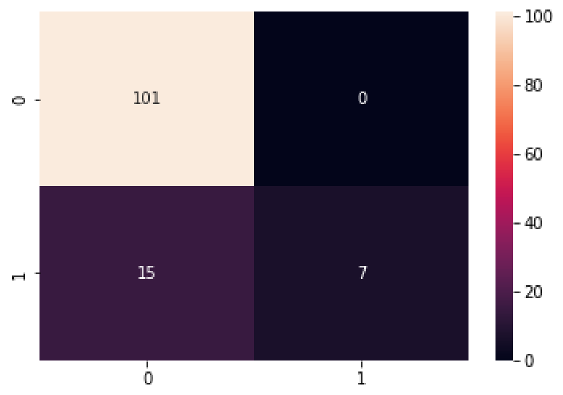
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      1.00      0.93       101
           1       1.00      0.32      0.48        22

    accuracy                           0.88       123
   macro avg       0.94      0.66      0.71       123
weighted avg       0.89      0.88      0.85       123
```

When Logistic Regression is applied to the data prepared by TfIdfVectorizer, the f1-score for fatality dropped from 0.86 to 0.48. The Recall value (which is the ratio of correctly predicted positive observations to the all observations in actual class) dropped to 0.32, which can be seen in the confusion matrix below. The algorithm correctly predicted 7 fatalities, but also falsely predicted 15 more, although those pilots ejected safely. f1-score being the weighted average of Precision and Recall, poor Recall score negatively affected the f1-score.

```
y_pred = model.predict(X_test_tf_idf_word)

cnf_matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(cnf_matrix,annot=True,fmt="d")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef8d257188>
```



Naïve Bayes

Classification report and the f1-score of the Naïve Bayes classification for the data prepared by using the CountVectorizer method is shown below:

```
nb_count_f1 = f1_score(y_test, y_pred, average='weighted')
nb_count_f1
```

```
0.9246876859012493
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.97      0.96       101
           1       0.84      0.73      0.78        22

    accuracy                           0.93       123
   macro avg       0.89      0.85      0.87       123
weighted avg       0.92      0.93      0.92       123
```

Classification report and the f1-score of the Naïve Bayes classification for the data prepared by using the TfIdfVectorizer method is shown below:

```
nb_tf_f1 = f1_score(y_test, y_pred, average='weighted')
nb_tf_f1
```

```
0.7404907084785133
```

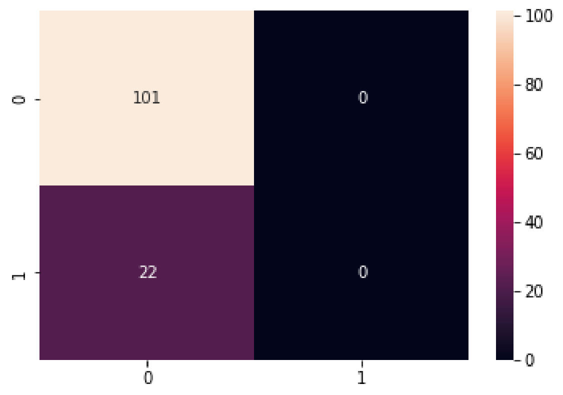
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      1.00      0.90       101
           1       0.00      0.00      0.00        22

    accuracy                           0.82       123
   macro avg       0.41      0.50      0.45       123
weighted avg       0.67      0.82      0.74       123
```

This time, the results from the TfIdfVectorizer are even worse. As shown in the confusion matrix below, the algorithm’s predictions for the fatalities were all wrong, although the Precision value for safe ejections were around 82 %.

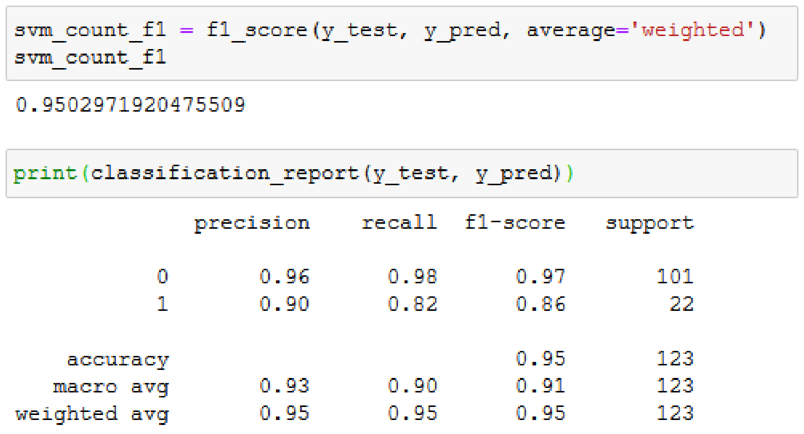
```
y_pred = model.predict(X_test_tf_idf_word)

cnf_matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(cnf_matrix,annot=True,fmt="d")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef8d251d48>
```



Support Vector Machines (SVM)

Classification report and the f1-score of the SVM classification for the data prepared by using the CountVectorizer method is shown below:



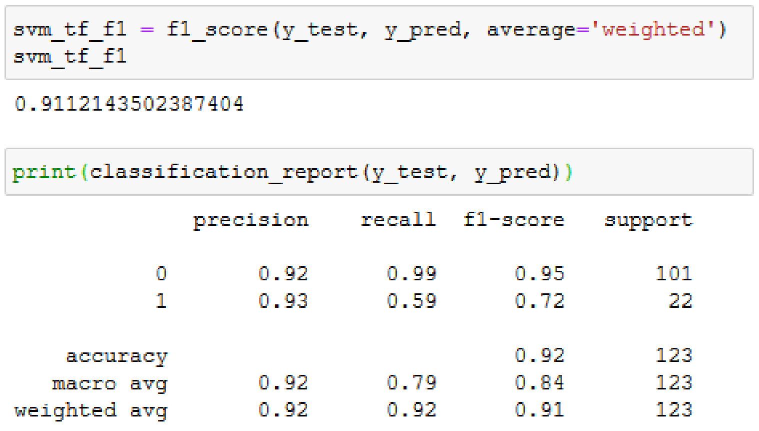
```
svm_count_f1 = f1_score(y_test, y_pred, average='weighted')
svm_count_f1
```

```
0.9502971920475509
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       101
           1       0.90      0.82      0.86        22

    accuracy                           0.95       123
   macro avg       0.93      0.90      0.91       123
weighted avg       0.95      0.95      0.95       123
```

Classification report and the f1-score of the SVM classification for the data prepared by using the TfIdfVectorizer method is shown below:



```
svm_tf_f1 = f1_score(y_test, y_pred, average='weighted')
svm_tf_f1
```

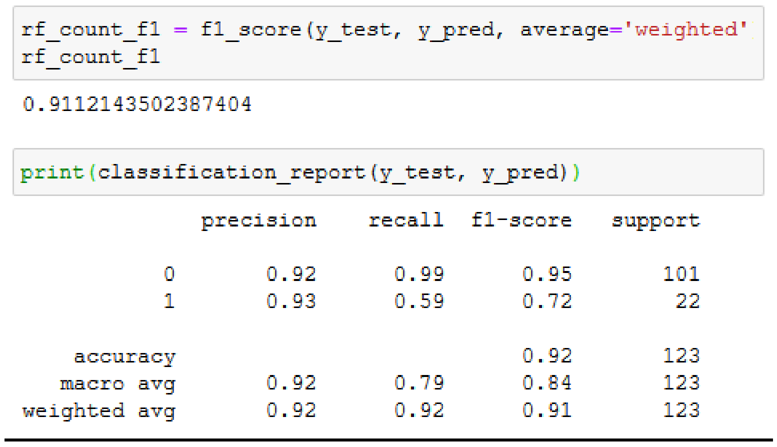
```
0.9112143502387404
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95       101
           1       0.93      0.59      0.72        22

    accuracy                           0.92       123
   macro avg       0.92      0.79      0.84       123
weighted avg       0.92      0.92      0.91       123
```

Random Forest Algorithm

Classification report and the f1-score of the Random Forest classification for the data prepared by using the CountVectorizer method is shown below:



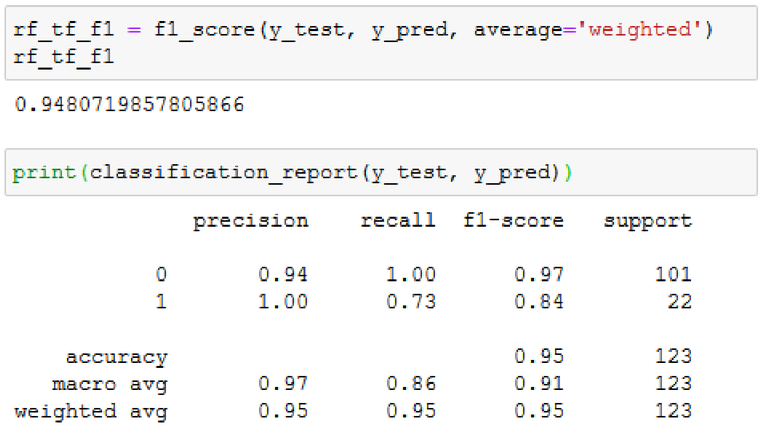
```
rf_count_f1 = f1_score(y_test, y_pred, average='weighted'
rf_count_f1
```

```
0.9112143502387404
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95       101
           1       0.93      0.59      0.72        22

    accuracy                           0.92       123
   macro avg       0.92      0.79      0.84       123
weighted avg       0.92      0.92      0.91       123
```

Classification report and the f1-score of the Random Forest classification for the data prepared by using the TfIdfVectorizer method is shown below:



```
rf_tf_f1 = f1_score(y_test, y_pred, average='weighted')
rf_tf_f1
```

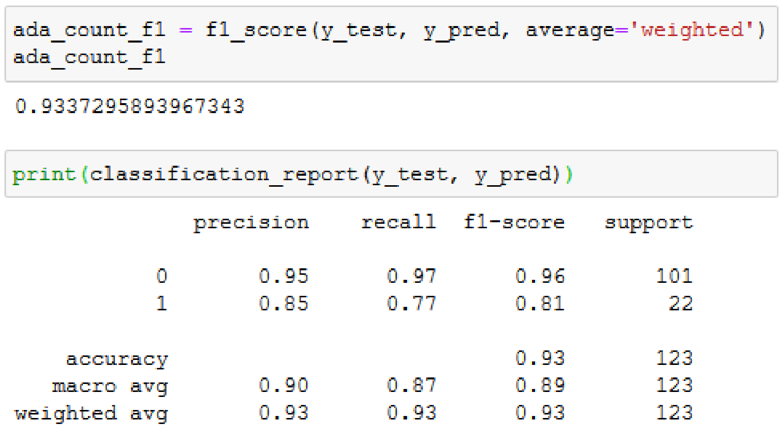
```
0.9480719857805866
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97       101
           1       1.00      0.73      0.84        22

    accuracy                           0.95       123
   macro avg       0.97      0.86      0.91       123
weighted avg       0.95      0.95      0.95       123
```

AdaBoost

Classification report and the f1-score of the AdaBoost classification for the data prepared by using the CountVectorizer method is shown below:



```
ada_count_f1 = f1_score(y_test, y_pred, average='weighted')
ada_count_f1
```

```
0.9337295893967343
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       101
           1       0.85      0.77      0.81        22

    accuracy                           0.93       123
   macro avg       0.90      0.87      0.89       123
weighted avg       0.93      0.93      0.93       123
```

Classification report and the f1-score of the AdaBoost classification for the data prepared by using the TfIdfVectorizer method is shown below:

```
ada_tf_f1 = f1_score(y_test, y_pred, average='weighted')
ada_tf_f1
```

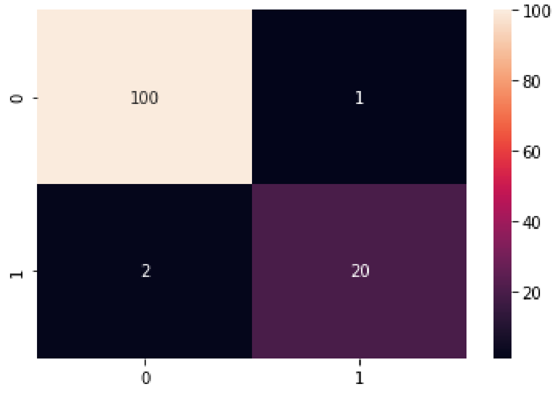
```
0.9753862231026939
```

```
print(classification_report(y_test, y_pred))
```

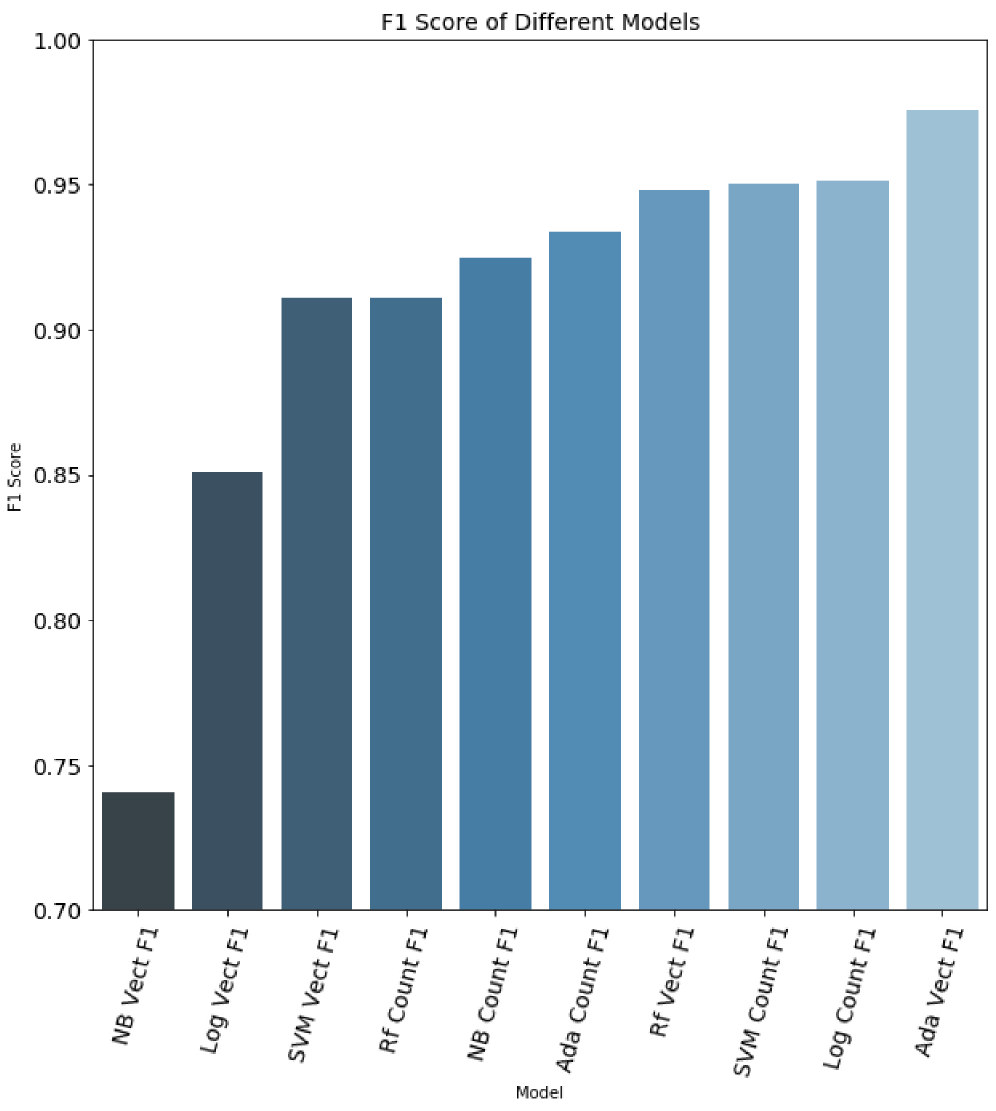
```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       101
           1       0.95      0.91      0.93        22

    accuracy                           0.98       123
   macro avg       0.97      0.95      0.96       123
weighted avg       0.98      0.98      0.98       123
```

Confusion matrix below shows that, the algorithm's predictions for both the fatalities and safe ejections were very accurate.

```
y_pred = model.predict(X_test_tf_idf_word)

cnf_matrix = confusion_matrix(y_test,y_pred)
sns.heatmap(cnf_matrix,annot=True,fmt="d");
```



Comparison of Results



The Barplot above shows f1-scores of the 5 algorithms applied to vectors produced by CountVectorizer and TfidfVectorizer methods. The results can be summarized as:

- AdaBoost for Tfidf vector produced the highest f1-score value of 0.975.
- The first 8 f1-scores were all above 0.91.
- Naïve Bayes method applied to Tfidf produced the lowest f1-score of 0.74. The confusion matrix showed that all the predictions for positive fatality were wrong, whereas all the predictions for safe ejections were correct.

Conclusion

In the first part of the article, the intention was to analyze the F-16 accidents by using WordClouds. The results can be summarized as:

- The human factor (mostly pilots and maintenance crew) was the major causes of the military fighter aircraft accidents and it is difficult remove the human-errors from the crashes. Air Force Safety Center stated that: ““We have aircraft piloted by human beings, designed by human beings and maintained by human beings… we are humans and make mistakes.”
- For a long time, F-16 had a well-known engine problem and some of the crashes in the 1990s can be attributed to this problem – which is solved now.
- During the years of intense combat action, accident rates increased to higher levels.
- F-16 is still in service after 42 years, but it is a much more reliable system now (and also total number of F-16s in USAF decreased), which is demonstrated by the decrease in the accident rate.

In the second part of the article, the focus was on using NLP techniques with the aim of finding details in the accident reports in order to predict whether the pilot was killed during the accident.

- Although the values in the “fatality” column were imbalanced, the classification algorithms produced very high f1-scores and accuracy values.
- Classification algorithms produced similar results for the data prepared by CountVectorizer and TfidfVectorizer methods.
- AdaBoost algorithm produced the highest f1-score (0.975), whereas Naïve Bayes f1-score was 0.74.

You can access to this article and similar ones here.

Photo by pixabay.com