CSE3505

Foundations of Data Analytics

J Component Report

A project report titled

Reducing Commercial Aviation Fatalities

By

19BLC1017 Priyanshu Prasad
19BLC1012 KPS Shivratna
19BLC1133 Ujjwal Gupta

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMPUTER ENGENEERING

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Submitted to

Dr. R. KARTHIK

DEC 2021

School of Electronics Engineering

DECLARATION BY THE CANDIDATE

I hereby declare that the Report entitled "**Reducing Commercial Aviation Fatalities"** submitted by me to VIT Chennai is a record of bonafide work undertaken by me under the supervision of **Dr. R. Karthik, Senior Assistant Professor, SENSE, VIT Chennai.**

Signature of the Candidate

Chennai

05/12/2021.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. R. Karthik,** School of Electronics Engineering for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Sivasubramanian. A,** Dean of the School of Electronics Engineering (SENSE), VIT University Chennai, for extending the facilities of the School towards our project and for his unstinting support.

We express our thanks to our **Head of The Department Dr. Vetrivelan. P (for B.Tech-ECE)** for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the courses till date.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

BONAFIDE CERTIFICATE

Certified that this project report entitled "**Reducing Commercial Aviation Fatalities"** is a bonafide work of **PRIYANSHU PRASAD (19BLC1017)** , **KPS SHIVRATNA (19BLC1012) and UJJWAL GUPTA (19BLC1133)** carried out the "J"-Project work under my supervision and guidance for CSE 3505 Foundation of Data Analytics.

Dr.R.Karthik

School of Electronics Engineering

VIT University, Chennai

Chennai – 600 127.

TABLE OF CONTENTS

INTRODUCTION

Abstract

Reducing commercial aviation fatalities is just one of the complex problems that have been solving for business, government, and military leaders for many years. Most flight-related fatalities stem from a loss of “airplane state awareness”. That is ineffective attention management on the part of pilots who may be distracted, sleepy, or in other dangerous cognitive states.

- A large part of the training given to the pilot involves the physiological aspects which are required while flying an airplane.
- This is important because one of the important abilities required for pilots is to multitask, the ability to concentrate, and the ability to pay attention to all these tasks.
- All of these may help to reduce pilot induced flight fatalities.
- Airplane state awareness (ASA) is a pilot performance attribute wherein the pilot should be able to realize and respond quickly to any change of state of the airplane.
- Loss of airplane state awareness may lead to many dangerous situations and may result in loss of airplane control wherein an extreme deviation from the intended flight path may occur.
- Loss of ASA is mainly due to loss of attention on the part of pilots who may be distracted, sleepy, or in other dangerous cognitive states.
- Due to the stressful environment, while flying, the possibility of the loss of awareness is common.

Our focus will be on predicting one of these cognitive states and helps the pilot to manage flights effectively.

Problem Statement and Objective

We are provided with real physiological data from pilots who were subjected to various distracting events. The pilots experienced distractions and resulted in one of the following three cognitive states:

- Channelized Attention (CA): This occurs when the pilot is focusing only on one task without giving any attention to other tasks.
- Diverted Attention (DA): The state of having one’s attention diverted by actions or thought processes associated with a decision. This is induced by having the subjects perform a display monitoring task.
- Startle/Surprise (SS): This is the response to a sudden unexpected stimulus. In aviation, this can be defined as an uncontrollable automatic reflex or reaction caused due to exposure to a sudden intense event that violated a pilot’s expectations.

The aim is to build a model that can estimate the state of mind of the pilot in real-time using the physiological data given. **When the pilot enters into any one of the above mentioned dangerous cognitive states, he/she should be alerted, thereby preventing any possible accident.**

Requirements and proposed system

We have taken the data from a Kaggle competition (https://www.kaggle.com/c/reducing-commercial-aviation-fatalities/data?select=test.csv). The size of the dataset is 6.32 GB, there should be sufficient disk space on the system to load this dataset.

We are provided with three csv files which are train, test and sample submission file. The train data is used for training our model, our model will be tested on the test dataset. Sample_submission.csv is provided to submit the final output in the CSV format.

The training data consist of three experiments: CA, DA, and SS. The output is one of the four labels: Baseline(no event), CA, DA, or SS. For example, if the experiment is CA, the output is either CA or Baseline(no event). The test data is taken from a full flight simulator. Here the experiment is called LOFT or Line Oriented Flight Training where the training of the pilot is carried out in a flight simulator, which artificially creates the environment of a real flight. In the test data, the experiment is given as LOFT and the output can be one of the four states at a given time. To predict the state of a pilot, physiological data are required. We have data from four sensors — EEG, ECG, Respiration, Galvanic skin response.

```
print(list(enumerate(train.columns)))
print()
print(list(enumerate(test.columns)))
```

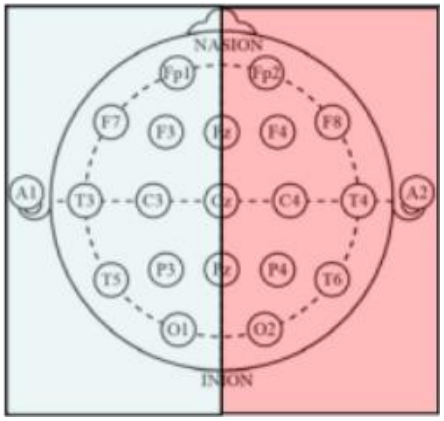
```
[(0, 'crew'), (1, 'experiment'), (2, 'time'), (3, 'seat'), (4, 'eeg_fp1'), (5, 'eeg_f7'), (6, 'eeg_f
8'), (7, 'eeg_t4'), (8, 'eeg_t6'), (9, 'eeg_t5'), (10, 'eeg_t3'), (11, 'eeg_fp2'), (12, 'eeg_o1'), (13,
'eeg_p3'), (14, 'eeg_pz'), (15, 'eeg_f3'), (16, 'eeg_fz'), (17, 'eeg_f4'), (18, 'eeg_c4'), (19, 'eeg_p
4'), (20, 'eeg_poz'), (21, 'eeg_c3'), (22, 'eeg_cz'), (23, 'eeg_o2'), (24, 'ecg'), (25, 'r'), (26, 'gs
r'), (27, 'event')]

[(0, 'id'), (1, 'crew'), (2, 'experiment'), (3, 'time'), (4, 'seat'), (5, 'eeg_fp1'), (6, 'eeg_f7'),
(7, 'eeg_f8'), (8, 'eeg_t4'), (9, 'eeg_t6'), (10, 'eeg_t5'), (11, 'eeg_t3'), (12, 'eeg_fp2'), (13, 'eeg
_o1'), (14, 'eeg_p3'), (15, 'eeg_pz'), (16, 'eeg_f3'), (17, 'eeg_fz'), (18, 'eeg_f4'), (19, 'eeg_c4'),
(20, 'eeg_p4'), (21, 'eeg_poz'), (22, 'eeg_c3'), (23, 'eeg_cz'), (24, 'eeg_o2'), (25, 'ecg'), (26,
'r'), (27, 'gsr')]
```

Let's analyse each attribute of the dataset.

- Id: Unique identifier for crew+time combination. A pilot with a particular time into the experiment is represented using an id. So for each id, we need to predict the state
- Crew: Unique id for a pair or pilot
- Experiment: For training, it will be either CA or DA or SS. For testing, it will be LOFT
- Time: Seconds into the experiment
- Seat: Seat of the pilot- 0 means left, 1 means right

EEG (Electroencephalogram) — This is the summation of all activities on the surface of the brain. Data from 20 electrodes are given to us. Each electrode lead is placed near a particular part of the brain (prefrontal(fp), temporal(t), frontal(f), parietal(p), occipital(o), central(c)). The odd numbers in the representation indicate that the electrode is placed on the left side of the brain, even numbers indicate the right side, and z indicate the middle region.



Position of electrodes in the scalp

- Eeg_f7: Data from the electrode near the prefrontal portion — left side
- Eeg_f8: Data from the electrode near the frontal area — right side
- Eeg_t4: Data from the electrode near the temporal area — right side
- Eeg_t6: Data from the electrode near the temporal area — right side
- Eeg_t5: Data from the electrode near the temporal area — left side
- Eeg_t3: Data from the electrode near the temporal area — left side
- Eeg_fp2: Data from the electrode near the prefrontal area — right side
- Eeg_o1: Data from the electrode near the occipital area — left side
- Eeg_p3: Data from the electrode near the parietal area — left side
- Eeg_pz: Data from the electrode near the parietal area — middle region
- Eeg_f3: Data from the electrode near the frontal area — left side
- Eeg_fz: Data from the electrode near the frontal area — middle region
- Eeg_f4: Data from the electrode near the frontal area — right side
- Eeg_c4: Data from the electrode near the central area — right side
- Eeg_p4: Data from the electrode near the parietal area — right side
- Eeg_poz: Data from the electrode near the parietal-occipital junction — Middle region
- Eeg_c3: Data from the electrode near the central area — left side
- Eeg_cz: Data from the electrode near the central area — middle region
- Eeg_o2: Data from the electrode near the occipital area — right side
- Ecg: Three-point electrocardiogram (ECG) signal — It measures the electrical activity of the heart (sensor output is in microvolts)
- R: Respiration sensor — It measures the rise and fall of the chest (Sensor output is in microvolts)
- Gsr: Galvanic skin response — The measure of electrodermal activity (Sensor output is in microvolts)
- Event: The output which is to be predicted — The state of the pilot at a given time. It will be either baseline (A no event) or SS(B) or CA(C)or DA(D)

As this is a multiclass classification problem wherein, for each id (for a particular crew at a particular time), we will predict the state of the pilot as belonging to one of the four given classes. The memory usage of train and test data is huge, hence we have optimized the memory usage of the train and test data.

```
Memory usage of dataframe is 1039.79 MB
Memory usage after optimization is: 241.38 MB
Decreased by 76.8%
Memory usage of dataframe is 3837.77 MB
Memory usage after optimization is: 942.31 MB
Decreased by 75.4%
```

We were able to reduce the memory usage of the train and test data by 75% approximately.

Given all the attributes, we will predict the probability of occurrence of each event.
We will use techniques such as hyper parameter tuning and cross validation for finding the best fit for our model. For building the model we will be utilizing algorithms like Decision Tree, LightGBM for classification and ranking.

Module description

We have used python as the programming language and Jupyter notebook for writing the code.
For **exploratory data analysis** and data wrangling the following libraries have been used:

- Numpy - A Python library used for working with arrays. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- Pandas - Mainly used for **data analysis**. Pandas allows importing data from various file formats such as comma-separated values, JSON, SQL, and Microsoft Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features.
- Matplotlib - Matplotlib is a comprehensive library for **creating static, animated, and interactive visualizations in Python**. Matplotlib makes easy things easy and hard things possible. Create publication quality plots. Make interactive figures that can zoom, pan, update.
- Seaborn - Seaborn is an open-source Python library built on top of matplotlib. It is used **for data visualization and exploratory data analysis**. Seaborn works easily with dataframes and the Pandas library. The graphs created can also be customized easily.
- Itertools - **Provides various functions that work on iterators to produce complex iterators**. This module works as a fast, memory-efficient tool that is used either by themselves or in combination to form iterator algebra.

For **data wrangling** the following libraries have been used:

- Scipy - **Used for scientific computing and technical computing**. It is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. Here we have used this library for detecting the outliers in the data by the Z-score method.

Z Score to detect Outliers

```
from scipy import stats
```

```
z=np.abs(stats.zscore(train)) # calculating z score
z
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float16)
```

- train_test_split - The train_test_split function is for splitting a single dataset for two different purposes: training and testing. The testing subset is for building your model. The testing subset is for using the model on unknown data to evaluate the performance of the model. train_test_split is a function in **Sklearn model selection** for splitting data arrays into **two subsets**: for training data and for testing data. With this function, you don't need to divide the dataset manually. By default, Sklearn **train_test_split** will make random partitions for the two subsets. However, you can also specify a random state for the operation.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train, y, test_size=0.4, random_state=25)
print(f'Number of rows in X_train: {X_train.shape[0]}')
print(f'Number of rows in X_test: {X_test.shape[0]}')
```

```
Number of rows in X_train: 2920452
Number of rows in X_test: 1946969
```

- ExtraTreesClassifier - ExtraTreesClassifier is an ensemble learning method fundamentally based on decision trees. ExtraTreesClassifier, like RandomForest, randomizes certain decisions and subsets of data to minimize over-learning from the data and overfitting. We have used ExtraTreesClassifier for finding the top 10 important features in our training data, which can be used in model building and thereby reducing the dimensionality of the train data.

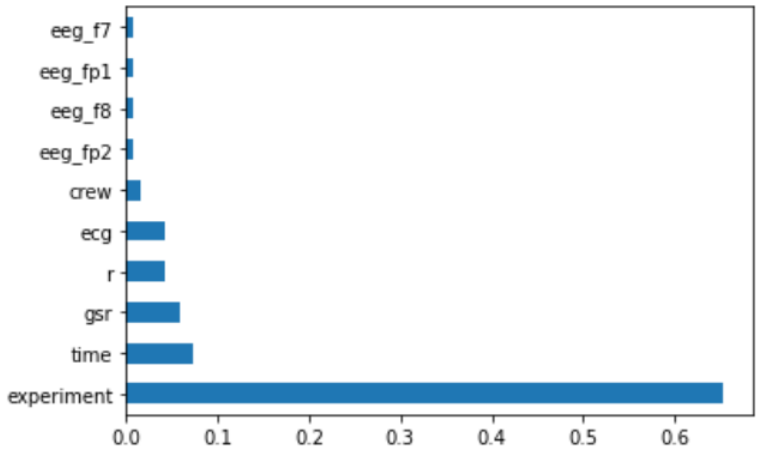
```
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
model.fit(train,y)
```

```
ExtraTreesClassifier()
```

```
print(model.feature_importances_)
```

```
[0.01531078 0.65336739 0.07294627 0.00400903 0.00748646 0.00718524
 0.00773869 0.00464369 0.00420283 0.00408866 0.00507419 0.00788043
 0.00427289 0.00411466 0.0056306  0.00664355 0.00591453 0.00626319
 0.00441403 0.00442632 0.00486016 0.00462933 0.00539573 0.00437058
 0.04270581 0.04304584 0.05937912]
```

```
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=train.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```



For **model building and evaluation** the following libraries have been utilized:

- Sklearn - It is the most **useful and robust library for machine learning in Python**. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.
- GridSearchCV - GridSearchCV is a library function that is a member of sklearn's model_selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters. In our code, we have utilized this function to find our best parameters for the Decision Trees Classifier. We can observe in the below snippet that the best 'criterion' for the decision tree classifier is 'entropy'.

```
dt_grid=GridSearchCV(estimator=DecisionTreeClassifier(),param_grid={'criterion':["gini", "entropy"]},cv=3)
```

```
dt_grid.fit(X_train,y_train)
```

```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy']})
```

```
dt_grid.best_params_
```

```
{'criterion': 'entropy'}
```

- DecisionTreeClassifier - It is a Supervised Machine Learning where the data is continuously split according to a certain parameter. In this method a set of training examples is broken down into smaller and smaller subsets while at the same time an associated decision tree gets incrementally developed. At the end of the learning process, a decision tree covering the training set is returned. The key idea is to use a decision tree to partition the data space into cluster (or dense) regions and empty (or sparse) regions.
- lightgbm - Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks. Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So, when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms. Also, it is surprisingly very fast, hence the word ‘Light’.

Results and Discussion

We tried training our data on Decision Tree (with all columns included), Decision Tree (with 10 most important columns) and Light gradient boosting and obtained the following results:

1. Decision Tree (with all columns included)

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier(criterion='entropy')
dt.fit(X_train,y_train)
```

```
DecisionTreeClassifier()
```

```
print('Accuracy of Decision Tree with all features',dt.score(X_test,y_test))
```

```
Accuracy of Decision Tree with all features 0.9993733849896942
```

```
from sklearn import metrics
y_pred=dt.predict(X_test)
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MAE: 0.0017067554747918431
MSE: 0.004923036781787486
RMSE: 0.07016435549328082
```

2. Decision Tree (with 10 most important columns)

```
dt1=DecisionTreeClassifier(criterion='entropy')
dt1.fit(X_train[feature_names],y_train)
```

```
DecisionTreeClassifier()
```

```
print('Accuracy of Decision Tree with top 10 important features',dt.score(X_test,y_test))
```

```
Accuracy of Decision Tree with top 10 important features 0.9993733849896942
```

```
y_pred1=dt.predict(X_test[feature_names])
print('MAE:', metrics.mean_absolute_error(y_test, y_pred1))
print('MSE:', metrics.mean_squared_error(y_test, y_pred1))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred1)))
```

```
MAE: 0.0012470665942806486
MSE: 0.0035819779359609733
RMSE: 0.05984962770110582
```

3. Light Gradient Boosting

```
import lightgbm as lgb
clf = lgb.LGBMClassifier(n_estimators=100,learning_rate=0.05)
```

```
clf.fit(X_train, y_train)
y_pred2 = clf.predict(X_test)
print("Accuracy of lgbm: ", accuracy_score(y_pred2, y_test))
```

```
Accuracy of lgbm:  0.9829098460222017
```

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred2))
print('MSE:', metrics.mean_squared_error(y_test, y_pred2))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred2)))
```

```
MAE: 0.04992015794807211
MSE: 0.14836445778027282
RMSE: 0.3851810714200178
```

Conclusion

In the case of Decision Tree, we can observe that after using the features with higher importance, the accuracy of the model has increased slightly. Also, there is significant dip in Error metrics of the model having features with higher importance. But, having such high accuracy indicates that the Decision Tree algorithm is overfitting the data.

In the case of Light Gradient Boosting method, the accuracy obtained is around 98% which is lesser than both the Decision Tree classifiers implemented before. And the mean error obtained here higher than the previous cases. A model is said to be a good fit if there exists some variability in the model.

Therefore, we can conclude that LGBM is the best model and move forward with it to calculate the probabilities of occurrence of a cognitive state.

```
sub.head() # submission file
```

	id	A	B	C	D
0	0	0.999695	0.000058	0.000209	0.000037
1	1	0.999668	0.000077	0.000217	0.000039
2	2	0.999701	0.000053	0.000209	0.000037
3	3	0.999668	0.000077	0.000217	0.000039
4	4	0.999690	0.000063	0.000209	0.000037

This table represents the probabilities of occurrence of the four cognitive states in which a pilot can be in.

References

- https://medium.com/analytics-vidhya/reducing-commercial-aviation-fatalities-ec338e37900c
- https://medium.com/swlh/decision-tree-classification-de64fc4d5aac
- https://lightgbm.readthedocs.io/en/latest/
- https://www.kaggle.com/c/reducing-commercial-aviation-fatalities/data