SAFE DRIVING CHALLENGE

ML Project Report

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

SUBMITTED BY

NAME OF THE STUDENT	ROLL NO
Ms. M SUSMITHA	17WH1A0591
Ms. M NAGA PRAVALLIKA	18WH5A0517
Ms. T BINDHU BHARGAVI	18WH5A0520

Department of Computer Science and Engineering
BVRIT HYDERABAD
College of Engineering for Women
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

Department of Computer Science and Engineering
BVRIT HYDERABAD
College of Engineering for Women
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)

Bachupally, Hyderabad – 500090

Acknowledgement

Firstly, I would like to express my immense gratitude towards **BVRIT HYDERABAD College of Engineering for Women**, which created a great platform to attain profound technical skills in the field of Computer Science though this industry enabled learning WISE.

I would like to extend my sincere thanks and gratitude to **Dr. K V N Sunitha,** Principal, BVRIT HYDERABAD College of Engineering for Women and WISE team of college for their meticulous planning and conduction of this learning program.

I would also like to extend my sincere thanks to WISE & Team of Talentsprint for enabling us with this unique learning platform.

M Susmitha(17WH1A0591)

M Naga Pravallika(18WH5A0517)

T Bindhu Bhargavi(18WH5A0520)

INDEX

LIST OF FIGURES

ABSTRACT

In this project we introduce a classifier which takes in multidimensional data consisting of real-world measurements of physical, environmental and vehicular continuous features obtained from number of driving sessions. We will show that using Naive Bayes classifier which assumes the data distribution to be Gaussian distribution we can make a prediction weather the driver is alerted or not while driving and achieve reasonable low misclassification rate for the given data. We will inspect how insight into relevant features were obtain by using Principal Component Analysis (PCA) and simple correlation matrix. We were able to obtain a misclassification rate as low as 12.03 % and 27.07 % for the test and training data respectively.

INTRODUCTION

With a training and test set consisting of 33 features from real time measurements test we want to use that information to predict if a certain driver is alerted or not alerted while driving. Here our goal is to construct a binary classifier which will predict a binary target value using the whole or a subset of the 33 features and give a prediction as Predictions = (1 if the driver is alert 0 if the driver is not alert (1) A. Datasets The datasets are gained from the website www.kaggle.com and consist of one training set and one test set. The datasets include measurements from total of 510 real time driving session where each driving session takes 2 minutes. This gives a new measurement of the each of the 33 features every 100ms. The headers in the datasets are listed in table I below. The size of the training set is a measurement set of 510 driving sessions done by 100 people. This results in a 604330×33 as the size of the training set. The test data have fewer observations and has

Problem Statement

Driving while distracted, fatigued or drowsy may lead to accidents. Activities that divert the driver's attention from the road ahead, such as engaging in a conversation with other passengers in the car, making or receiving phone calls, sending or receiving text messages, eating while driving or events outside the car may cause driver distraction. Fatigue and drowsiness can result from driving long hours or from lack of sleep.

The data for this Kaggle challenge shows the results of a number of "trials", each one representing about 2 minutes of sequential data that are recorded every 100ms during a driving session on the road or in a driving simulator. The trials are samples from some 100 drivers of both genders, and of different ages and ethnic backgrounds. The files are structured as follows:

The first column is the Trial ID - each period of around 2 minutes of sequential data has a unique trial ID. For instance, the first 1210 observations represent sequential observations every 100ms, and therefore all have the same trial ID The second column is the observation number - this is a sequentially increasing number within one trial ID The third column has a value X for each row where

X = 1 if the driver is alert

X = 0 if the driver is not alert

The next 8 columns with headers P1, P2 , …….., P8 represent physiological data;

The next 11 columns with headers E1, E2, …….., E11 represent environmental data;

The next 11 columns with headers V1, V2, …….., V11 represent vehicular data;

APPROACH

- Initially, we have analyzed train and test datasets
- Imported the required libraries
- By using Data preprocessing, logistic regression, feature engineering, PCA, Support vector regression, Neural network we have predicted the output.

STATISTICS OF THE CODE

- We have used google Collaboratory to predict the output.

SAFE DRIVING CHALLENGE

1 INTRODUCTION

The objective is to design a classifier that will detect whether the driver is alert or not alert, employing data that are acquired while driving. This report is meant to illustrate the process of building a predictive machine learning model of the Machine Learning.

2 DATASETS

There are 604,329 instances of data in the training dataset and 120,840 instances of data in the test dataset. The data for this challenge shows the results of a number of" trials", each one representing about 2 minutes of sequential data that are recorded every 100ms during a driving session on the road or in a driving simulator. The trials are samples from some 100 drivers of both genders, and of different ages and ethnic backgrounds. The files are structured as follows: The training data was broken into 500 trials, each trial consisted of a sequence of approximately 1200 measurements spaced by 0.1 seconds. Each measurement consisted of 30 features; these features were presented in three sets: physiological (P1...P8), environmental (E1...E11) and vehicular (V1...V11). Each feature was presented as a real number. For each measurement we were also told whether the driver was alert or not at that time (a Boolean label called Is Alert). No more information on the features was available.

3 EXISTED MODEL

In order to summarize existed work and formulate a plan in order to build an outperformed machine learning predictive model. Similar machine learning techniques are applied to this dataset. The techniques most participants used limited to Nave Bayes, Logistic Regression, Support Vector Machine, Neural Network, and Random Forest. But the performances of their models are totally different, as they preprocessed the original data in different ways, especially in their feature engineering.

Thus, I will mainly focus on the feature engineering methods applied by the participants, instead of how they choose parameters of algorithms in the summary part.

The highest score (AUC = 0.861151) was reached by a logistic regression model. As the dataset consists of sequential data recorded every 100ms for 2 minutes in each trial, the partitions of the data by trials (Trial ID) rather than randomly partition. The Means and Standard Deviations of each trial were computed as new features (include the target feature Is Alert). After- wards, feature selection based on diagnostics of the logistic regression was conducted and three strong features were chosen for modelling (sdE5, V11, and E9). How- ever, this model applies future observation (The mean and standard deviation can only be calculated when a trial is finished), thus inapplicable for real-life situations. A running Mean and Standard deviation were applied to training instead and the AUC has dropped slightly, from 0.861151 to 0.849245). We focus on the instances at the initial moment the driver lost alertness, the dataset is reduced significantly in this way and he highlighted the factors change significantly between status change for feature

selection. E4, E5, E6, E7, E8, E9, E10, P6, V4, V6, V10, and V11 are selected for building a Neural Network. This model reaches an AUC of 0.84953 & also attempts to aggregate data from each trial and calculate means and standard deviations as additional features. After tossing up correlated feature and other feature engineering, a logistic regression model trained from feature selected data reaches an AUC of 0.80779. Fourier generates around 600 new features to the dataset (The inverse, the square, and the cube of each features, all the combinations of 2 columns, time interval variables). It reaches the highest AUC by applying forward search to select predictive features. A Nave Bayes model trained by these selected features reach an AUC of 0.844. We trained an epsilon-SVR, RBF kernel model with parameters c = 2, g = 1/30, and p = 0.1, which reaches an AUC of 0.839 and applies a random forest with 199 trees and min node size of 25, the correlated features are tossed out beforehand. This predictive model reaches an AUC of 0.81410.

3.1 SUMMARY OF EXISTED MODEL

An important feature for this dataset is that it contains sequential data. For each trial, the dataset records data every 100ms. Thus, all the participants shuffle the dataset by trials for the purpose of preserving this sequential feature. Aggregating data within a trial to generate means and standard deviations as new features for modelling is proofed as a useful method of data preprocessing. Another useful method of data preprocessing is to choose the instances close to the moment the driver lost alertness, which re- duce time to train the models significantly.

Multiple methods of feature selection are applied, the mean/standard deviation of existed features, inverse, the square, the cube, and a combination of 2 columns are viewed as potentially useful new features. Correlated, remain constant features are always tossed out. As for the choice of predictive machine learning algorithms, there is no valid proof that one algorithm out-performs all the others in this specific situation. Generally, Nave Bayes, Logistic Regression, Random Forest, Support Vector Machine, and Neural Network all reach a good performance in this case.

4 MODEL BUILDING PLANS

Even though many existed models have already had a decent performance, it's still possible to improve the model. A plan for building a new predictive model is outlined in this section.

4.1 GAP IDENTFICATION

The predictive model with the highest AUC value is trained from 20% of the training dataset. What's more, the means and standard deviations of each trial are future observation features. Those make this predictive model inapplicable to a real-life situation. An AUC value of 0.861151 also means there are still rooms for improvement. Another noticeable point within most of the existed work is that most of the models are evaluated by either AUC score or classification accuracy. For this specific situation, it's obviously more important to identify those not alert instances as driving while not alert can be deadly. Failing to identify 'not alert' can lead to worse consequences compare to failing to identify 'alert'. Thus, true negative rate (TN / (TN + FP)) can also be a valuable measure of evaluation as it shows the percentage of 'not alert' instances successfully identified. Furthermore, as all the models' classification accuracies are above 50%, which makes building an ensemble model to reach a better performance possible as if the recalls

and the specificities of all the models can reach above 50% at the same time for all the models.

4.2 MODEL BUILDING PLAN

Firstly, those existed models with good performance will be reproduced, includes the way they preprocess the data and the parameters they choose to build predictive models. Secondly, a local evaluation will be conducted on these models. The recalls and specificity will be used for evaluation, apart from classification accuracy and AUC score. Then to group those models with recall and specificity both higher than 50% to build an ensemble model, aims to reach a better performance than all the existed models.

5 SOLUTION DEVELOPMENT

Python is used as the developing environment for this project. Scikit-learn is the machine learning tool applied. Missing data is identified as 0 throughout all the dataset.

5.1 FIRST MODEL

The first predictive model is built by the data preprocessing method. We were concerned that using the entire data set would create too much noise and lead to inaccuracies in the model. The final goal of the system is to detect the change in the driver from alert to not alert so that the car can self-correct or alert the driver. So, we decided to just focus on the data at the initial moment when the driver lost alertness. According to this, I subset the dataset to the moment when the driver lost alertness. The rows with the feature 'Is Alert' == 0 and the last rows with the feature 'Is Alert'== 0 are chosen, along with 5 rows before and after each (100ms of time between each observation, 5 rows before means focus on the data recorded 0.5s before and after the driver lost alertness).After sub setting, 37421 instances without duplication are chosen to build the predictive model.

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.decomposition import PCA
    from sklearn import preprocessing
    from sklearn.linear_model import LogisticRegressionCV
    from sklearn.naive_bayes import GaussianNB
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import confusion_matrix
    from sklearn.metrics import roc_auc_score

    from sklearn.svm import LinearSVC # Linear Support Vector Classigication
    from sklearn.svm import NuSVC

    from sklearn.model_selection import GridSearchCV, StratifiedKFold, train_test_split
```

```
    train = pd.read_csv('/content/drive/My Drive/train.csv')
    test = pd.read_csv('/content/drive/My Drive/test.csv')
    exp = pd.read_csv('/content/drive/My Drive/example_submission.csv')
```

```
[ ] train.info()
    test.info()
    train.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 604329 entries, 0 to 604328
Data columns (total 33 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   TrialID  604329 non-null  int64
 1   ObsNum   604329 non-null  int64
 2   IsAlert  604329 non-null  int64
```

Data preprocessing

+ Code + Text

```
[ ] new_df = pd.DataFrame()

[ ] noalert = train.index[train['IsAlert'] == 0]
    turnnoalert = train.iloc[noalert-1][train['IsAlert'] == 1]

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: UserWarning: Boolean Series key will be reindexed to match DataFrame index.

[ ] for i in turnnoalert.index: # Return 500ms before and after the moment
        new_df = new_df.append(train[i-5:i+6])

[ ] new_df = new_df.drop_duplicates()

[ ] new_df
```

	TrialID	ObsNum	IsAlert	P1	P2	P3	P4	P5	P6	P7	P8	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	V1	V2	V3	V4	V5	V
604323	510	1193	1	32.0303	7.68265	800	75.0000	0.081731	680	88.2353	0	17.807	222.11	0	0	0.016379	323	2	2	1	64	0.0	91.40	0.175	240	3.01875	0	179
604324	510	1194	1	32.0051	10.13240	800	75.0000	0.081731	680	88.2353	0	17.807	222.11	0	0	0.016379	322	2	2	1	64	0.0	91.51	0.280	240	3.01875	0	180
604325	510	1195	1	32.0393	12.45040	800	75.0000	0.081731	680	88.2353	0	17.807	222.11	0	0	0.016379	322	2	2	1	64	0.0	91.51	0.280	240	3.01875	0	180
604326	510	1196	1	32.0762	10.06180	800	75.0000	0.081731	680	88.2353	0	17.807	222.11	0	0	0.016379	322	2	2	1	64	0.0	91.56	0.175	240	3.01875	0	180
604327	510	1197	1	32.1154	17.84500	800	75.0000	0.081731	680	88.2353	0	17.807	222.11	0	0	0.016379	322	2	2	1	64	0.0	91.56	0.175	240	3.01875	0	180
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
604314	510	1184	0	31.9667	9.17322	688	87.2093	0.089515	684	87.7193	0	17.807	222.11	0	0	0.016379	327	2	2	1	64	0.0	90.94	0.175	240	3.01875	0	178

5.1.1 FEATURE ENGINEERING

There are 30 features included in the dataset, thus filter those features with higher impact could not only save computational resources but also potentially improve the performance of the predictive model. Principle Component Analysis (PCA) is applied as the feature engineering technique in this case. For PCA, the dataset is standardized firstly, then the fraction of variances of each feature is calculated to identify those features have higher impact on the result.

▾ Feature Engineering

```
[ ] pcadata = new_df.drop(columns = ['TrialID', 'ObsNum', 'IsAlert']) # Drop unneccessary columns
```

▾ Standarization

```
[ ] X_scaled = preprocessing.scale(pcadata)
    pcadata.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 37421 entries, 604323 to 604318
Data columns (total 30 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   P1      37421 non-null  float64
 1   P2      37421 non-null  float64
 2   P3      37421 non-null  int64
 3   P4      37421 non-null  float64
 4   P5      37421 non-null  float64
 5   P6      37421 non-null  int64
 6   P7      37421 non-null  float64
 7   P8      37421 non-null  int64
 8   E1      37421 non-null  float64
 9   E2      37421 non-null  float64
 10  E3      37421 non-null  int64
 11  E4      37421 non-null  int64
 12  E5      37421 non-null  float64
 13  E6      37421 non-null  int64
 14  E7      37421 non-null  int64
 15  E8      37421 non-null  int64
```

· Principle Component Analysis

```
[ ] pca=PCA()
    pca.fit(X_scaled)
    X_pca=pca.transform(X_scaled)
```

```
[ ] #let's check the shape of X_pca array
    print ("shape of X_pca", X_pca.shape)
```

```
shape of X_pca (37421, 30)
```

```
[ ] #%%
    # Scree Plot
    y = pca.explained_variance_ratio_

    x = np.array([i for i in range(1, len(y)+1)])

    plt.plot(x, y, 'r-x')
```

```
[<matplotlib.lines.Line2D at 0x7f3816730828>]
```

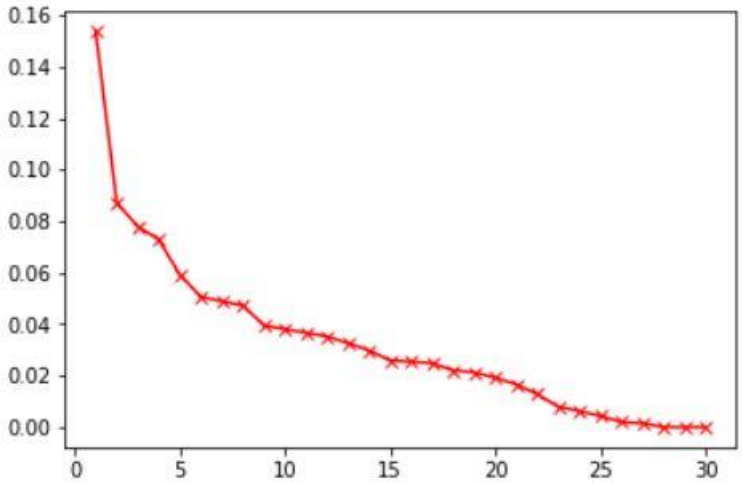


図 1: Scree Plot of 30 features

We can see that the first 14 attributes contribute 80.95% of the total variance, the number of features

```
[ ] # Filter principle components
    sum(pca.explained_variance_ratio_[:14])
    X_pca = X_pca[:,:14] # Slice the first 14 features
    X_pca.shape
```

```
(37421, 14)
```

```
[ ] X_pca # Training features
```

```
array([[-1.13162853,  0.31998431,  0.26156361, ...,  1.02433172,
         0.70943025, -0.18363423],
       [-1.19089936,  0.24896352,  0.27907797, ...,  0.74432583,
         0.81958233, -0.25328004],
       [-1.19688915,  0.18795301,  0.28791467, ...,  0.49758852,
         0.90339316, -0.27264132],
       ...,
       [-1.15137249,  0.33964321,  0.32035685, ...,  0.96243759,
         0.66141226, -0.15012464],
       [-1.11689356,  0.29833156,  0.32107332, ...,  0.81488074,
         0.71706924, -0.1312757 ],
       [-1.14711508,  0.20941089,  0.3323469 , ...,  0.55608563,
         0.87515175, -0.21948589]])
```

```
[ ] new_df.IsAlert # Target features
```

```
604323    1
604324    1
604325    1
604326    1
604327    1
         ..
604314    0
604315    0
604316    0
```

selected for modelling is decreased from 30 to 14 in this way.

5.1.2 MODELLING

As the size of subset is relatively smaller, stratified 10-fold cross-validation is applied as data evaluation method to make full use of the dataset. Naive Bayes, Logistic Regression, Random Forest, Support Vector Machine, and Neural Network models are built from this dataset. Gaussian Nave Bayes model performs a validation ac- curacy of 61.74%. Logistic Regression with optimization algorithm of 'liblinear' reaches a validation accuracy of 64.6%. Multiple models in Support Vector Machine family include Linear-SVC, Nu-SVC, C-SVC are applied as well. Their validation accuracies varied from 65% to 78%

A Neural Network with 5 neurons in the first hidden layer and 2 neurons with the second hidden layers reaches a validation accuracy of 65.01%.

I tried to use neural networks with different architectures, another neural network with 5 hidden layers and 14, 14, 12, 10, 5 neurons in each layer. The activation function is also changed from RELU to logistic regression. Unfortunately, the performance of the new neural network does not change mach.

Modeling

```
[ ] # Logistic Regression Cross Validation
    clf_CV = LogisticRegressionCV(cv=10, random_state=0, solver = 'liblinear') # Model Setting
    clf_CV.fit(X_pca, new_df.IsAlert) # Model Fitting
```

```
LogisticRegressionCV(Cs=10, class_weight=None, cv=10, dual=False,
                     fit_intercept=True, intercept_scaling=1.0, l1_ratios=None,
                     max_iter=100, multi_class='auto', n_jobs=None,
                     penalty='l2', random_state=0, refit=True, scoring=None,
                     solver='liblinear', tol=0.0001, verbose=0)
```

```
[ ] # Evalution

    clf_CV.score(X_pca, new_df.IsAlert)
```

```
0.6460276315437856
```

```
[ ] # Confustion matrix evaluation

    clf_cv_pred = clf_CV.predict(X_pca)
    len(clf_cv_pred == 1)
    #sum(clf_cv_pred == 0)
```

```
37421
```

```
[ ] sum(clf_cv_pred == 0)
```

```
5276
```

```
[ ] confusion_matrix(new_df.IsAlert, clf_cv_pred)
```

```
array([[ 4066, 12036],
       [ 1210, 20109]])
```

```
[ ] sum(new_df.IsAlert == 1)
```

```
21319
```

```
[ ] roc_auc_score(new_df.IsAlert, clf_cv_pred)
```

```
0.5978791659850288
```

Try Another Models

NAIVE-Bayes

```
[ ] # Cross Validated Naive Bayes Model
    skf = StratifiedKFold(n_splits=10)
    params = {}
    nb = GaussianNB()
    gs = GridSearchCV(nb, cv=skf, param_grid=params, return_train_score=True)
```

```
x_train, x_test, y_train, y_test = train_test_split(X_pca, new_df.IsAlert, random_state = 42)
```

```
gs.fit(x_train, y_train)
gs.cv_results_
gs.score(X_pca, new_df.IsAlert)
gs_predict = gs.predict(X_pca)
```

```
confusion_matrix(new_df.IsAlert, gs_predict)
```

```
array([[ 4026, 12076],
       [ 2181, 19138]])
```

```
roc_auc_score(new_df.IsAlert, gs_predict)
#nb.fit(x_train, y_train)
#nb.score(x_test, y_test)
```

```
0.5738639710706295
```

```
# Cross Validated  Random Forest
grid_forest = RandomForestClassifier(random_state = 42)
hyperparams = [{'n_estimators':[5, 10 ,50]}]
grid_search = GridSearchCV(grid_forest, hyperparams, cv = 10, scoring = 'neg_mean_squared_error')
```

```
grid_search.fit(x_train, y_train)
```

```
GridSearchCV(cv=10, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False, random_state=42,
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_estimators': [5, 10, 50]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

```
selected_model = grid_search.best_estimator_
```

```
print ('Grid winner', selected_model.score(x_test, y_test))
selected_model.score(X_pca, new_df.IsAlert)
```

```
Grid winner 0.7658187259512612
0.9414499879746666
```

```
randomfpred = grid_search.predict(X_pca)
```

Neural Network

```
from sklearn.neural_network import MLPClassifier
```

```
nn1 = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2), random_state=1)
nngridsearch = GridSearchCV(nn1, cv = skf, param_grid = params, return_train_score = True)
nngridsearch.fit(x_train, y_train)
```

```
nngridsearch.cv_results_
nngridsearch.score(X_pca, new_df.IsAlert)
nn_predict = nngridsearch.predict(X_pca)
```

```
confusion_matrix(new_df.IsAlert, nn_predict)
```

```
array([[ 4205, 11897],
       [ 1205, 20114]])
```

```
roc_auc_score(new_df.IsAlert, nn_predict)
```

```
0.6023126662815139
```

Model	Accuracy	recall	Specificity	AUC Sc
Logistic Regression	64.60%	94.32%	25.25%	0.5978
Nave Bayes	61.74%	89.47%	25.02%	0.5724
Random Forest	93.54%	97%	90.08%	0.9354
Linear-SVC	64.55%	94.85%	24.44%	0.5958
Nu-SVC	78.63%	92.36%	60.45%	0.7640
C-SVC	67.55%	98.13%	27.06%	0.626
Neural Network1	65.01%	94.31%	26.21%	0.6026
Neural Network2	65.96%	97.02%	24.83%	0.6092

Performances of algorithm on PCA dataset

It can be found from the performance diagram that all the models perform pretty well on predicting those drivers 'in alert'. However, most models cannot reach a decent result when it comes to identifying drivers not in alert, which is more important in this specific situation. On the other hand, the Random Forest model outer form all other models, especially when it comes to specificity, which makes it a part of our final ensemble model. The Nu-SVC model reaches a specificity of more than 50% as well, which means it can also be part of an ensemble model.

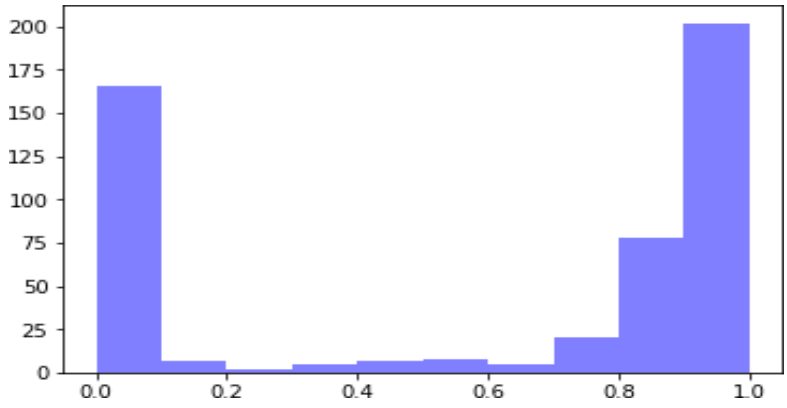
5.2 SECOND MODEL

Unfortunately, we did not get a good predictive ma- chine learning model by the first data preprocessing method (apart from the Random Forest with 50 trees model). I decided to conduct an exploratory analysis on the dataset in order to provide a guidance of data preprocessing

5.2.1 EXPLORATORY ANALYSIS AND FEATURE ENGINEERING ON THE DATASET

We calculate the average Is Alert value per trial and plot the result on a histogram.

図 2: Histogram of mean alertness per trial



It is found that for most drivers, they either stay alert or not alert throughout the 1200ms trial. Thus, the characteristic of each driver, recorded in the mean and standard deviation of each attribute, can be helpful for predictive analysis.

On the other hand, it is impossible to get the mean and standard deviation of a trial at the beginning of each trial, which makes using stable means and standard deviations of each feature unpractical in real-life situation. More- over, using stable means and standard deviations cannot record the change of the driver's behavior within a trial, which may be constantly changing overtime.

For these reasons, we decided to use rolling means and standard deviations of each features as new features in- stead of simply using stable means and standard deviations in order to make full use of the sequential feature.

The rolling window is set to 5, as for every 5 instances (500ms), it calculates the mean and standard deviation for them, then the algorithm drops the first instance and add a new instance, etc.

Use rolling means and std of each attribute as new features

```
train_new = pd.DataFrame() # New df
#train_new
for i in range(0, max(train.TrialID)+1): # Within each trial
    temp_data = train[train['TrialID'] == i] # Create a temporary df for each trial
    for j in list(train)[3:]: # For all the attributes of each trial
#       print (train_2[train_2['TrialID'] == i][j])
        temp_data['m{}'.format(j)] = train[train['TrialID'] == i][j].rolling(window = 5).mean() # Create the Rolling mean
        temp_data['sd{}'.format(j)] = train[train['TrialID'] == i][j].rolling(window = 5).std() # Create the Rolling Std
    train_new = train_new.append(temp_data)
```

```
[ ] train_new = train_new.fillna(0) # Missing Value
```

Data partition

```
[ ] train_new = train_new.drop(columns = ['TrialID', 'ObsNum', 'IsAlert']) # Drop unneccessary columns

    X_train, X_test, y_train, y_test = train_test_split(train_new, train.IsAlert, test_size = 0.2, random_state = 23)
```

```
[ ] train_new
```

	P1	P2	P3	P4	P5	P6	P7	P8	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	34.7406	9.84593	1400	42.8571	0.290601	572	104.8950	0	0.000	0.00	1	-20	0.015875	324	1	1	1	57	0.0	101.96	0.175	752	5.99375	0	2005	0	13.4	0	4
1	34.4215	13.41120	1400	42.8571	0.290601	572	104.8950	0	0.000	0.00	1	-20	0.015875	324	1	1	1	57	0.0	101.98	0.455	752	5.99375	0	2007	0	13.4	0	4
2	34.3447	15.18520	1400	42.8571	0.290601	576	104.1670	0	0.000	0.00	1	-20	0.015875	324	1	1	1	57	0.0	101.97	0.280	752	5.99375	0	2011	0	13.4	0	4

5.2.2 MODELLING

Similarly, we applied algorithms mentioned above to this preprocessed dataset. As the size of the dataset is big enough, we use 80%-20% to train-test split the dataset in- stead of cross-validation. Firstly, we tried Random Forest algorithm, the one per- forms the best in the last feature selected dataset, to see if there's any improvement compare to the other feature selecting method. The Random Forest has 50 trees, the parameters are the same as the one applied before. It reaches a decent performance on the validation dataset, with a validation accuracy of 98.91%. Algorithms of the Support Vector Machine family all fail to converge within a specific period of time. A neural network with four hidden layers, each layer has 90, 70, 50, 30 neurons respectively also applied, reaches a validation accuracy of 80.76%. Furthermore, Nave Bayes and Logistic Regression have not improved much compare to the preview models. Generally, Neural Network and Random Forest performs better than other models in this situation, and Random Forest performs far better than Neural Networks.

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_estimators = 50, random_state = 42)
forest.fit(X_train, y_train)
```

```
forest.score(X_test, y_test)
```

```
0.9891615508083332
```

```
for_predict = forest.predict(X_test)
```

```
# Confusion Matrix
confusion_matrix(y_test, for_predict)
```

```
array([[49663,  1247],
       [   63, 69893]])
```

```
#AUC score
roc_auc_score(y_test, for_predict)
```

```
0.987302614234641
```

```
from sklearn.linear_model import LogisticRegression
```

```
logr = LogisticRegression(random_state=0, solver='saga').fit(X_train, y_train)
```


```
logr = LogisticRegression(random_state=0, solver='saga').fit(X_train, y_train)
```


```
logr.score(X_test, y_test)
```

```
0.6120745288170371
```

```
logr_pred = logr.predict(X_test)
```

```
confusion_matrix(y_test, logr_pred)
```

```
array([[21363, 29547],
       [17340, 52616]])
```

Naïve Bayes

```
from sklearn.naive_bayes import GaussianNB
```

```
nb2 = GaussianNB()
nb2.fit(X_train, y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
nb2.score(X_test, y_test)
```

```
0.6286300531166746
```

```
[ ] confusion_matrix(y_test, nb2_pred)

array([[44303,  6607],
       [38279, 31677]])

[ ] roc_auc_score(y_test, nb2_pred)

0.6615175785943694
```

Neural Network

With four hidden layers 90 - 70 - 50 - 30

```
[ ] from sklearn.neural_network import MLPClassifier

nn2 = MLPClassifier(solver='adam', alpha=1e-5,activation = 'logistic',hidden_layer_sizes=(90, 70, 50, 30), random_state=1)
nn2.fit(X_train, y_train)

[ ] nn2_predict = nn2.predict(X_test)

[ ] nn2.score(X_test, y_test)

0.8347922492677842

[ ] confusion_matrix(y_test, nn2_predict)

[ ] confusion_matrix(y_test, nn2_predict)

array([[41678,  9232],
       [10736, 59220]])

[ ] roc_auc_score(y_test, nn2_predict)

0.8325962434798789
```

Model	Accuracy	recall	Specificity	AUC Sc
Logistic Regression	61.21%	75.21%	41.96%	0.5858
Nave Bayes	62.86%	45.28%	87.02%	0.6615
Random Forest	98.91%	98.58%	97.55%	0.9873
Neural Network	80.76%	96.40%	59.26%	0.7783

Performances of algorithm on rolling mean std dataset

5.2.1COMPARISON OF MODELS

Comparing the performances of models trained from data preprocessed by different methods, it is found that algorithms logistic regression, Support Vector Machine, and nave Bayes are not suitable for this problem. While Neural Network can reach a good performance in the dataset preprocessed by generating time sequential feature, it is not the model fits the dataset the best. The Random Forest Algorithm generates the best result on predictive analysis, either trained from data preprocessed by PCA or from data preprocessed by other feature engineering techniques. Another interesting finding is that most models perform better when it comes to predicting 'alert' drivers than to predicting 'not alert' drivers, apart from the Nave Bayes model. Considering two values are basically equally distributed (alert: 349785, not alert: 254544), it's hard to say one label is over represented than the other, which makes the unbalanced predict result hard to explain.

As a result, I choose three models for local evaluation, which are two Random Forest models and a Neural Net- work Model.

6 Local Evaluation

We use the data'solution.csv' to evaluate the final models.

6.1 MODEL 1

The first model is the Random Forest trained by the data with features selected from PCA.

	Predict = 0	Predict=1
Actual = 0	22571	7343
Actual = 1	63616	27310

AUC = 0.52744

Though this model only reaches an accuracy of 41.28% on the test dataset, it identifies many not alerted drivers correctly. Overall, this model is not good enough, no matter evaluated by which method.

6.2 MODEL 2

The second model is the Random Forest trained from the data with added features of rolling mean and standard deviation.

	Predict = 0	Predict = 1
Actual = 0	16671	13243
Actual = 1	8679	82247

AUC = 0.7309

This model reaches a good performance, with classification accuracy of 81.86% on the test data. It has a good performance in predicting alert drivers, with recall = 90.45%, precision = 86.13% and F1-score = 88.24%. However, for this specific situation. The model is expected to predict 'not alert' drivers precisely, specificity (The percentage of Actual = 0 is predicted correctly) should be the evaluation method we focus on for this rea- son. The specificity of this model only reaches 55.73%, which still has lots of room to improve.

Overall, the AUC value of this model is 0.7309, not as good as the work I referenced, but still an improvement.

6.3 MODEL 3

The third model is the Neural Network trained from the data with added features of rolling mean and standard deviation. It has four hidden layers with 90, 70, 50, 30 neurons in each layer.

We were meant to use the first layer to grab all the original features and the coming layers to process and predict the output, thus the number of neurons for the first layer is

as many as the number of the features.

	Predict = 0	Predict = 1
Actual = 0	12886	17028
Actual = 1	361	90565

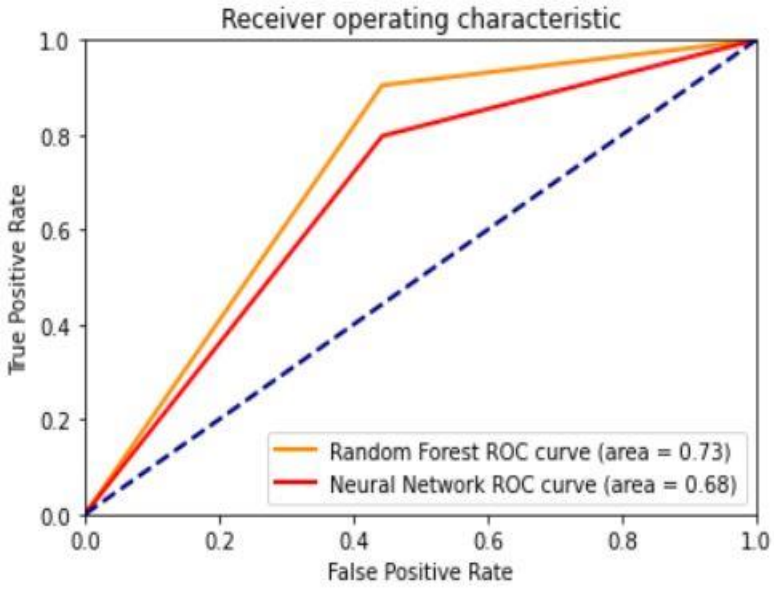
AUC = 0.71340

This model reaches a good performance as well, com- pares to the first model. It successively predicts most of alert drivers (recall = 99.6%, precision = 84.17%, F1- score = 91.24%). However, the model fails to predict many not alert drivers correctly (specificity = 43.08%), which is the more important evaluation method for this predictive model.

6.4 COMPARISON OF MODEL 2 AND MODEL 3

Both two models perform better on predicting alert drivers than identifying not alert drivers as the true positive rate are both higher than their true negative rate in their confusion matrix, though the main goal of this predictive model is to predict not alert drivers. The curve reaches 100% true positive rate firstly is the neural network, the other curve is the random forest. It also can be found that the random forest model performs better than the neural network model. However, the neural network predicts most alert drivers correctly and when it predicts a driver as not alert, it's correct at the most of times.

☒ 3: ROC curve of two models



10

The random forest model reaches a significantly higher result on identifying not alert drivers from all the drivers than the neural network model, though when it predicts a driver as not alert, it gets 34.25% chance of being wrong. The Random Forest model would be a better choice in this situation, but the architecture of the neural network model can be optimized to reach a higher performance.

7 RESULT REFLECTION AND COMPARISON

7.1 RESULT CONCLUSION

This project was meant to build a supervised learning model to predict not alert drivers, the model with the best performance is achieved by Random Forest with 50 trees in it. It predicts 16671 of 29914 not alert drivers correctly in the test data. It reaches a classification accuracy of 81.86% and AUC value of 0.7309.

7.2 RESULT COMPARISON

When it compares to the results of those in the leader- board, there are lots of participants' models reach a higher performance. The best model reaches an AUC value of 0.86115, though it applies means and standard deviations of each trial as new features. Almost 20 participants' models reach AUC scores over 0.8, which is significantly higher than mine. Refers that there is still large room to improve my model.

7.3 DISCUSSION AND FUTURE WORK

The existed predictive model for this problem is far from perfection. There are a few perspectives that can improve the performance of the model.

7.3.1 DATA PREPROCESSING METHOD

The rolling means and the standard deviation is proved to be a good method to preserve the sequential attribute of the data. However, rolling means for every 0.5s could be too short to grab the driving pattern of a driver. Expand the rolling window to produce rolling means and standard deviations in a longer period could be considered as a useful method to introduce the long-term driving patterns of drivers. Better performance is believed can be achieved by model learns not only from drivers' behaviors in a short time (0.5s) but in a long time as well.

7.3.2 MODEL OPTIMIZATION AND SELECTION

Though the neural network model fails to produce a better performance than the random forest model, it is still not convincing that random forest is always the best option for this problem. Neural network still shows great potential to produce good result. Further work could to optimize the architecture of neural networks.

REFERENCE LINK

- **https://www.kaggle.com/c/stayalert/data**

PROJECT LINK

https://github.com/bindhu520/Safe-driving-Challenge-ML-PROECT-