Mini project report

This project is to use decision tree model to evaluate the performance of the dataset.

Information of the dataset:

The dataset contains 2060 rows and 2049 columns, from [feature_0] to [feature_2047], with dtype ‘float64’ and a [label] column, with dtype ‘int64’.

The [label] column has 3 class, namely 0,1,2. The ‘0’ class contains 783 records, ‘1’ class contains 920 records and ‘3’ class contains 357 records.

Data preprocessing:

```
In [5]: 1 df.describe()
```

Out[5]:

	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	...	feature_2039
count	2060.000000	2060.000000	2060.000000	2060.000000	2060.000000	2060.000000	2060.000000	2060.000000	2060.000000	2060.000000	...	2060.000000
mean	0.160921	0.270409	0.208184	0.204598	0.241283	0.234374	0.246521	0.267722	0.226706	0.208153	...	0.240621
std	0.060174	0.081587	0.123492	0.332334	0.119511	0.608519	0.061969	0.701613	0.708157	0.090262	...	0.720848
min	0.064219	0.112422	0.119411	0.000000	0.131731	0.000000	0.144053	0.000000	0.000000	0.042953	...	0.000000
25%	0.119265	0.223384	0.151960	0.062676	0.165306	0.013066	0.199476	0.026164	0.004395	0.147668	...	0.008080
50%	0.153173	0.263307	0.164523	0.104471	0.199482	0.036835	0.234758	0.049078	0.010298	0.194815	...	0.018348
75%	0.193926	0.313065	0.191397	0.161176	0.258664	0.086375	0.289554	0.088395	0.020641	0.252502	...	0.038293
max	0.411141	0.673628	1.298625	1.287342	1.279045	2.285157	0.666712	2.662355	2.571561	0.875835	...	2.662868

8 rows × 2049 columns

```
In [6]: 1 print(df.isna().sum())
```

```
feature_0       0
feature_1       0
feature_2       0
feature_3       0
feature_4       0
feature_5       0
feature_6       0
feature_7       0
feature_8       0
feature_9       0
feature_10      0
feature_11      0
feature_12      0
feature_13      0
feature_14      0
feature_15      0
feature_16      0
feature_17      0
feature_18      0
```

From the first table, we can see that the sample mean of the feature columns is very close. The range of then mean is between 0.1 to 0.2. Besides, from the [min] and [max] row, we can see that the minimum and maximum tuples are close to the sample mean. There is no large dataset and the data are very close to each other.

From the second table, we can see that there is no ‘nan’ value from each of the feature columns. Therefore, the original dataset is very clean and there is no necessity to preprocess the data.

Model creation, training & prediction:

From the dataset, there are 2048 feature attributes and 1 label attribute. The decision tree model can only read the value in numpy array instead of a big table. So, we should first split the dataset into X_col, the data with feature columns only, and y_col, the data with label column only. Then, convert the X_col and y_col table into numpy array, X and y, so that the model can read the data easily.

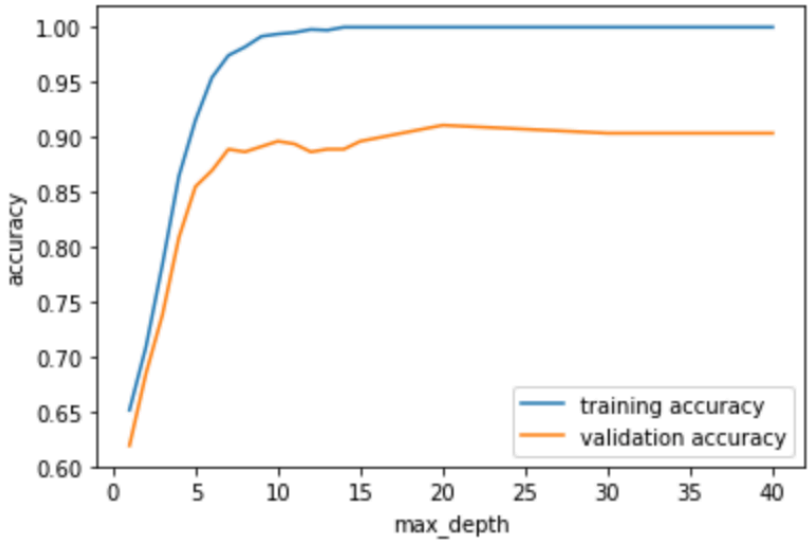
Second, we should split the data into training set, validation set and test set using train_test_split. The size of training set is 1442, validation set is 412 and test set is 206. The ratio is 7:2:1. This is to ensure 70% of the data (train set) is used to train and fit into the model, 20% of the data (validation set) is used evaluate the model and fine tune the model hyperparameters, 10% of the data (test set) is to evaluate the final model. The variable for training set is X_train, y_train, the validation set is X_valid, y_valid, the testing set is X_test, y_test.

In this report, we would use random forest classifier as the model to evaluate the dataset. The reason of choosing random forest is that the performance of this model is much better than decision tree classifier and produce a higher accuracy. To build the random forest, we need to first consider the number of trees to build. In this case, we set n_estimators = 100, that is to build 100 trees. These trees will be built completely independently from each other, and the algorithm will make different random choices for each tree to make sure the trees are distinct. To make prediction using random forest, the algorithm provides a probability for each possible output label. Then, the probability predicted by all the trees are averaged, and the class with the highest probability is predicted. As we can see that, the training accuracy of the model is 99.3% and the testing accuracy is 90.3%

```
random_forest = RandomForestClassifier(n_estimators = 100,  max_depth = 10,random_state = 2, n_jobs = -1)
random_forest.fit(X_train,y_train)
print(f'train set accuracy: {random_forest.score(X_train,y_train)}')
print(f'test set accuracy: {random_forest.score(X_test, y_test)}')
```

```
train set accuracy: 0.9937586685159501
test set accuracy: 0.9029126213592233
```

We use the training set to fit and train the model. There are some parameters in RandomForestClassifier, including n_estimators, max_depth and n_jobs. In this report, we set n_estimators to 100. This is to create 100 trees randomly and predict the highest probability. We also set n_jobs = -1. This is to use all the cores in the computer so that the model can run faster. Here, we set the max_features to default value because the model has already performed quite well if we set the max_features to default. Finally, we set max_depth to 10 (see the reason below).



In the project, we use for loop to set the max_depth in a list. Then, we create a random forest model using the parameter as mentioned above and set the max_depth equal to [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,20,30,40]. Therefore, the accuracy of model with different max_depth is plotted above. We found that accuracy of training and validation set will not change after max_depth = 20. When the max_depth = 10, the validation accuracy can achieve above 90% and the training accuracy is good enough, not exactly equal to 1. This means that the training set is not overfitted.

Cross-validation:

```
from sklearn.model_selection import KFold
KFold = KFold(n_splits = 10, shuffle=True, random_state=66)
new_cvs2 = cross_val_score(random_forest,X_valid,y_valid,cv = KFold)
print(f'Cross validation score: {new_cvs2}')
print(f'Average cross validation score: {new_cvs2.mean()}')
```

```
Cross validation score: [0.95238095 0.95238095 0.95121951 0.90243902 0.92682927 0.87804878
 0.87804878 0.87804878 0.80487805 0.87804878]
Average cross validation score: 0.9002322880371662
```

In this project, we use the validation set and create the k-fold cross-validation model. We set n_splits to 10, which refer to the 10-fold. When performing 10-fold cross-validation, the data is first partitioned into 10 parts of (approximately) equal size, called folds. Next, a sequence of models is trained. The first model is trained using the first fold as the test set, and the remaining folds (2–10) are used as the training set. The model is built using the data in folds 2–10, and then the accuracy is evaluated on fold 1. Then another model is built. This process is repeated until using folds 10 as test sets. For each of these 10 splits of the data into training and test sets, we compute the accuracy. At last, we have collected 10 accuracy values.

Besides, we set the shuffle parameter of KFold to true and fix the random_state. This is to ensure that each fold contain suitable amount of sample from class '0', class '1' and class '2', so that the test set is informative. The average cross-validation score of the 10 folds is 0.90. This shows that the model performs quite well on unseen data and hence prevent the overfitting problem.

Model evaluation:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.25, random_state = 123)

new_forest = RandomForestClassifier(n_estimators = 100,  max_depth = 10,random_state = 2, n_jobs = -1)
new_forest.fit(X_train,y_train)
print(f'train set accuracy: {new_forest.score(X_train,y_train)}')
print(f'test set accuracy: {new_forest.score(X_test,y_test)}')
```

```
train set accuracy: 0.9948220064724919
test set accuracy: 0.912621359223301
```

To evaluate the performance of the model, we build a new model with the same parameters that we used previously. We split the original dataset to training set and testing set and fit the training set into new_forest. We get the training accuracy = 99.5% and testing accuracy = 91.3%. We further evaluate the model using the classification report, as shown below:

	precision	recall	f1-score	support
0	0.96	0.88	0.92	188
1	0.87	0.98	0.92	243
2	0.97	0.80	0.88	84
accuracy			0.91	515
macro avg	0.93	0.88	0.90	515
weighted avg	0.92	0.91	0.91	515

From the table, it contains precision, which measures how many of the samples predicted as positive are actually positive; recall, which measures how many of the positive samples are captured by the positive predictions. From the table above, we can see that the precision and recall of the three class '0', '1' and '2' are quite high, about 0.8 to 0.9. This shows that the model does not produce many false positive and avoid any false negative. From the table, the f-score column takes the precision and recall into account. The f-score of class '0' and '1' is 0.92 while that of class '2' is 0.88. This also shows that the predictive performance of the model is good enough.

Final model

```
final_forest = RandomForestClassifier(n_estimators = 100, max_depth = 10, n_jobs = -1,random_state = 74)
final_forest.fit(X,y)
print(final_forest.score(X,y))
```

```
0.9917475728155339
```

This is the final model for the project. We use the original data X for the feature columns and y for the target label column. This is to make use of all the data from the training data, the original data provided, to train the model and we used the parameter that we tuned previously. The final training accuracy of the model is 99.2%.

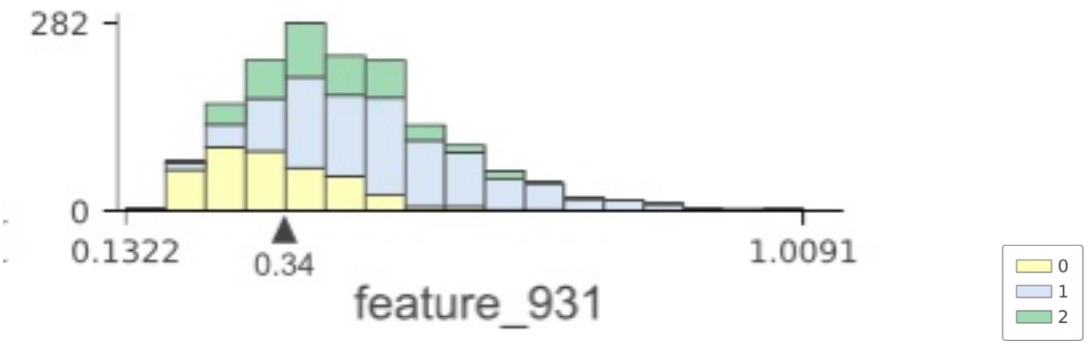
Plotting

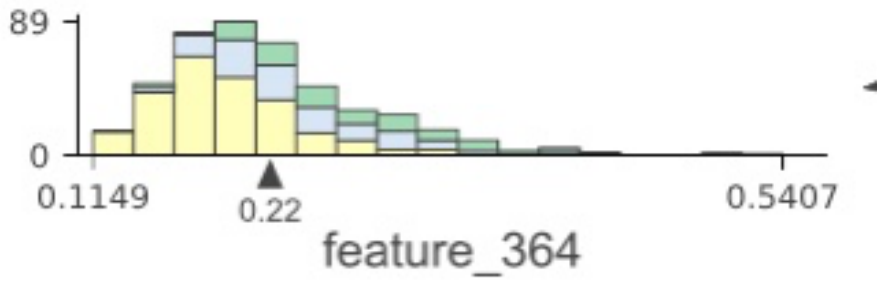
As we used n_estimators = 100 in our final model, we choose the 1st decision tree for the graph plotting.

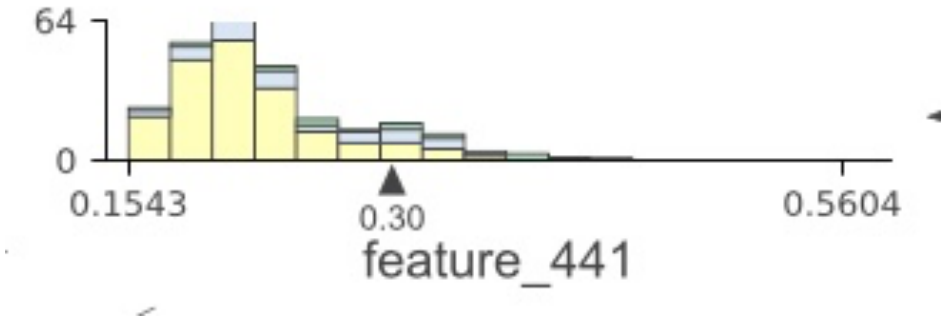
```
final_forest.estimators_[0]
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=10,
                       max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=868452254, splitter='best')
```

Finally, we import the dtreeviz library to plot the histogram for each leaf node. Below are the three splits from the decision tree:







Conclusion

In this report, we use the features and the data provided to build the random forest classifier model. To evaluate the performance of the model and fine tuning the parameters, we use the k-fold cross-validation model and the classification report. The final model for graded is 'final_forest', which used all the training data to train the model.