COSC2670 Assignment 2

I certify that this is all my own original work. If I took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in my submission. I will show I agree to this honour code by typing "Yes": **Yes**

Student ID: s3778500

Student Name: Aditya Kakde

Affiliation: Master of Analytics(MC242), RMIT University

Email ID: s3778500@student.rmit.edu.au

Date of Report: 2nd June 2020

Table of Contents

1. Executive Summary

Accelerometers are devices which measure the acceleration of any object in 3-dimensional space along the 3 co-ordinate axes. This effectively helps us to identify the orientation of that object and also track the movement of that object. Because of this utility, the technology has its applications in various domains like medical, engineering, transport, navigation, etc.

For this predictive analysis task, I considered the data obtained from a chest mounted accelerometer. Using this data, I tried to predict the activity of user. To make this prediction, I used 2 different classifiers(classification algorithms), viz., k-Nearest Neighbours and Decision Tree. For each of the model, I identified and selected the optimal number of features and hyper-parameter values. After training and fitting both the models, k-Nearest Neighbours classifier performed better with an overall accuracy of 0.7896.

After looking at the accuracy, we know that it can be improved further. The accuracy can be improved by considering a larger sample to train the model. Also, a different classifier such as Random Forests or Naïve-Bayes can be considered for modelling.

2. Introduction

The problem statement for this analysis can be given as:

"Identify the best classifier to predict the activity of user using accelerometer data."

For this analysis, we will use the data obtained from chest mounted accelerometer. This data has been downloaded from the UCI Machine Learning Repository. The data has been measured using an uncalibrated accelerometer at a frequency of 52Hz. These measurements have been recorded for 15 users. The data is collected in 15 csv files where data for each user is available in a separate csv file. Each file contains following 4 columns:

1. **X**: Represents acceleration values recorded along the X axis.
2. **Y**: Represents acceleration values recorded along the Y axis.
3. **Z**: Represents acceleration values recorded along the Z axis.
4. **Activity**: Represents the activity for which values have been recorded.

For our analysis, we will be following a methodology as outlined in the next section.

3. Methodology

We will conduct this analysis using Python. We will follow below steps to perform this analysis:

3.1. Importing and Preparing the Data

Here, we will first outline the steps to import the data and then prepare it for modelling.

3.1.1. Import and Data Check

We will start by importing the data. Here we will read data from all the 15 csv files and store it in a Pandas data frame. While importing, we will create a new column which we will represent the user for which the measurements have been recorded. After importing, we will check for the dimensions of this data frame to ensure that all the data has been imported. Then we will check the data types for all the columns to ensure that all columns have been assigned the correct data type. After confirming that the data has been imported correctly, we will prepare the data for modelling.

3.1.2. Feature Check

Here, we will start by checking if there are any missing values present in any of the column. Then we will compute the summary statistics for all the numeric features(columns). By doing this, we intend to see if there are any impossible values present for any of the numeric feature. We also intend to check the distribution of values for the numeric features. After checking for impossible values, we will display the unique values present for all the categorical features. The output will show us all the categories present for the categorical features. From this output, we will analyse and fix all the issues like unexpected category, typos, whitespaces, etc.

After completing above steps, we will have a clean data. Next step will be to extract a sample from this data.

3.1.3. Extract Sample

In this step, we will extract a sample of 100,000 records(instances). We cannot be choosing a smaller sample as it is important to capture as much variability as possible while training the model. We also cannot choose a bigger sample as it would be very inefficient to conduct the modelling on such a large data using local computer.

Once we extract the sample of appropriate size, the next step would be to perform encoding on this extracted sample.

3.1.4. Perform Encoding

For implementing any classification algorithm in Python, we need all the data in numeric format. Encoding is the process of converting categorical non-numeric features into numeric format. So, once we have extracted the sample, we will perform encoding on this extracted sample. After this step, all our data should be in numeric format.

Next step would be to split the descriptive and target features.

3.1.5. Split Descriptive and Target Features

For implementing any classification algorithm in Python, it is important to separate the descriptive and target features. This is because, while passing data to the implementation of any algorithm, it is necessary for the descriptive and target features to be stored separately.

After splitting the descriptive and target features, we will perform scaling on data for the descriptive features.

3.1.6. Perform Scaling

Usually the values in every numeric feature have different ranges. Also, after encoding, the data set contains multiple binary level(0, 1) features. Such difference in the range of values for multiple features lowers the performance of algorithm. To avoid this, we perform scaling. For our data, we will be performing normalisation(type of scaling) on values of all the descriptive features.

After performing all the above steps, our data would be ready for modelling.

3.2. Data Exploration

We will perform this step before modelling to better understand the data. For data exploration, we will use the entire data set which has been cleaned.

3.2.1. Explore each column

Here, we will explore all the features individually using a pie chart, histograms, and a bar chart. This will help us understand the distribution of values for each feature.

3.2.2. Explore relationships between columns

Here, we will explore all the possible relationships between features by plotting the scatter plots. Results of this exploration will help us comprehend the relationships between all the features.

The next step would be to perform predictive modelling.

3.3. Predictive Modelling

For performing this step in Python, we will use the Scikit-Learn package.

In this step, we will train and evaluate 2 models. The 2 models would be k-Nearest Neighbours and Decision Tree. For modelling, we will be using the descriptive and target feature data which we will be creating from the extracted sample of 100,000 instances.

3.3.1. Split Train and Test Data

First step of predictive modelling is to split the data into 2 parts, viz., train and test. We will split the data in the ratio of 70:30. So, we will have 70,000 instances for training the model and 30,000 instances for testing the model. We are splitting the data in this particular ratio so that we have enough instances to train and test the models. If we increase the instances in train data, we will not have enough instances to test the data. Similarly, if consider more instances for testing the data, we will have unnecessary more instances to test the model and there will not be enough instances left to train the data.

We will use the training data when selecting features, tuning the hyper-parameters, and training the model. Once the model has been trained using optimal number of features and hyper-parameter values, we will use the test data to evaluate the performance of our model.

After splitting the data into train and test, we will have 4 data sets using which we will perform the predictive modelling. These 4 data sets would be train descriptive feature data, test descriptive feature data, train target data and test target data.

3.3.2. Feature Selection, Hyper-parameter Tuning and Training the Model

In this step, we will identify the optimal number of features to be used. We will do this by ranking the features according to their importance. We will calculate these importance values using below 2 methods:

1. **F-score:** This method is based on statistical F-score of features. This method ranks the features based on their relationship with the target feature.
2. **Mutual Information:** This method is based on entropies of features. This method also uses the relationship of features with the target feature.

After calculating the importance values for all the features, we will train the model using different number of top ranked features. From these trained models, we will select the features using which the model performed

best. Then we will perform hyper-parameter tuning. While doing this, we will train and fit the model for various hyper-parameter values. We will then compare model performances for all these values and chose the ones which give best performance. This way we will find the optimal number of features and the hyper-parameter values.

The above process is iterative where we perform the same task of fitting the model for different values. To implement this in Python, the Scikit-Learn package has a functionality called pipe. Here, we mention all the possible values for number of features and hyper-parameters. We also mention the method for ranking the features. The pipe then fits the model for all the combinations of these values and returns the best model.

In this analysis, we will use pipe to perform the feature selection and hyper-parameter tuning for both the k-Nearest Neighbours and the Decision Tree models. After performing this step, we will have the optimal values for hyper-parameters and the number of features to be considered for both the models.

3.3.3. Model Evaluation

Once we have both the models trained with optimal features and the hyper-parameter values, we will fit these models on the test data. Then using various performance metrics, we will evaluate both these models and compare their performances. Based on their performances, we will select the best performing model.

Now that we have outlined the methodology, we will implement this in Python and check the results.

4. Results

In this section, we will look at the results for every step performed in above outlined methodology.

4.1. Importing and Preparing the Data

Below are the results of steps carried out for importing and preparing the data:

4.1.1. Import and Data Check

Once the data was imported from 15 csv files and stored in a data frame *df_user_dat*, we checked the dimensions and first 5 rows for this data frame. We also checked the data types assigned for all the columns.

```
Out[2]: (1926896, 5)
```

Out[3]:

	X	Y	Z	Activity	User
0	1502	2215	2153	1	user_1
1	1667	2072	2047	1	user_1
2	1611	1957	1906	1	user_1
3	1601	1939	1831	1	user_1
4	1643	1965	1879	1	user_1

```
Out[4]: X            int64
        Y            int64
        Z            int64
        Activity     int64
        User        object
        dtype: object
```

From above output, we have around 2 million records and 5 features. Here, *Activity* is our target feature. Also, *X*, *Y* and *Z* are continuous descriptive features, *User* is categorical(nominal) descriptive feature and *Activity* is categorical(nominal) target feature.

4.1.2. Feature Check

While performing the feature check, we started by checking for null values in all the columns (features).

```
Out[5]: X           0
        Y           0
        Z           0
        Activity    0
        User        0
        dtype: int64
```

From above output, we confirmed that no null values were present in the data. Next, we calculated the summary statistics for all the continuous features to check for impossible values. We also displayed the unique values present in all categorical features.

Out[6]:

	X	Y	Z
count	1926896.00	1926896.00	1926896.00
mean	1987.65	2382.52	1970.60
std	111.36	100.32	94.46
min	282.00	2.00	1.00
25%	1904.00	2337.00	1918.00
50%	1992.00	2367.00	1988.00
75%	2076.00	2413.00	2032.00
max	3828.00	4095.00	4095.00

```
Unique values for feature Activity :
[1 2 3 4 5 6 7 0]

Unique values for feature User :
['user_1' 'user_2' 'user_3' 'user_4' 'user_5' 'user_6' 'user_7' 'user_8'
 'user_9' 'user_10' 'user_11' 'user_12' 'user_13' 'user_14' 'user_15']
```

From above output, all the continuous features seemed to have appropriate values. For categorical features, only *User* feature had expected values. In *Activity* feature, 0 is an unexpected value. This is because, as per the description provided by the data provider, the data is recorded only for 7 activities. We checked the count of instances where *Activity* was 0.

```
Out[8]: 3719
```

As seen in the output, out of around 2 million instances, only 3719 instances had *Activity* value as 0.

So, we dropped the instances where *Activity* was 0.

4.1.3. Extract Sample

Once the data had been cleaned, we extracted a sample of 100,000 records and stored it in a new data frame *df_user_sample*. While extracting the sample, we set the random state to 111 so that we get same instances every time we run the code and the results could be reproduced.

```
Out[12]: (100000, 5)
```

As seen in the output, sample of 100,000 instances had been successfully extracted.

4.1.4. Perform Encoding

Here, we performed encoding on *User* feature as this was to only non-numeric categorical feature. After performing the one-hot encoding, we checked the columns names to see if required columns had been created. We also checked the dimensions to confirm this.

```
Out[14]: Index(['X', 'Y', 'Z', 'Activity', 'User_user_1', 'User_user_10',
                'User_user_11', 'User_user_12', 'User_user_13', 'User_user_14',
                'User_user_15', 'User_user_2', 'User_user_3', 'User_user_4',
                'User_user_5', 'User_user_6', 'User_user_7', 'User_user_8',
                'User_user_9'],
               dtype='object')
```

```
Out[15]: (100000, 19)
```

From above output, columns for all the users had been successfully created. We knew from the description of this data that it was collected for 15 users. So, from the dimensions we can confirm that 14(*User* column reused of 15th user) new columns have been successfully added.

4.1.5. Split Descriptive and Target Features

After encoding the *User* feature, we separated the descriptive and target features. We stored the descriptive features in data frame named *Data* and the target features in a NumPy array named *Target*.

```
Dimensions of descriptive feature data: (100000, 18)
Dimensions of Target feature data: (100000,)
```

From the above output, descriptive and target features had been successfully separated and were then available in *Data* and *Target*, respectively.

4.1.6. Perform Scaling

As outlined in the methodology, our last step in preparing the data was scaling the data for descriptive features. After performing this step, all the data for descriptive features was be normalized and available in a 2-dimensional NumPy array.

```
Datatype for descriptive feature data: <class 'numpy.ndarray'>

Datatype for target feature data: <class 'numpy.ndarray'>
```

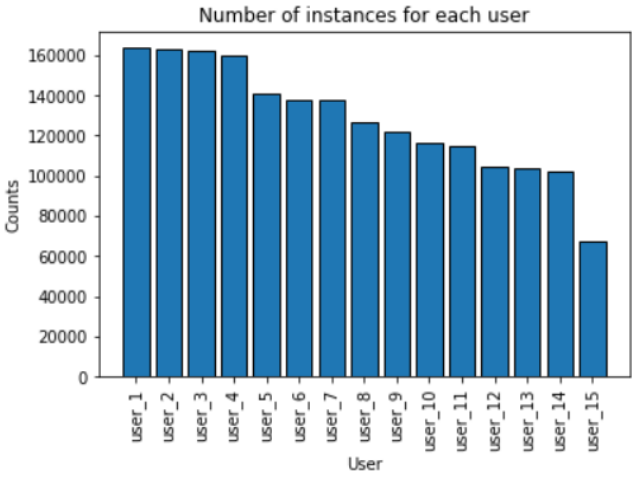
Now that we have discussed the results for importing and preparing the data, we will explore and analyse the results for Data Exploration.

4.2. Data Exploration

We will start by looking at the results which we got after exploring each column.

4.2.1. Explore each column

For *Activity* feature, we saw that most of the instances were of activity 1 and 2. There were very few instances which represent activity 5, 6 and 7. This showed us that data was collected from the user during their working hours. The distribution of activities in above pie chart more or less represented the same activity distribution that an individual would follow in the working hours if he/she were employed on a desk job.



Next, from the above bar chart representing the number of measurements for each user, we saw that we have much more measurements recorded for user 1, 2 and 3. We could expect that measurements for these users will dominate the results of our predictive modelling.

After plotting the histograms for values in X, Y and Z features, we observed that for X, Y and Z acceleration values, variation was limited. Majority of the values were between 1500 to 3000. These histograms confirmed that we did not have any unexpected values measured which might be incorrect.

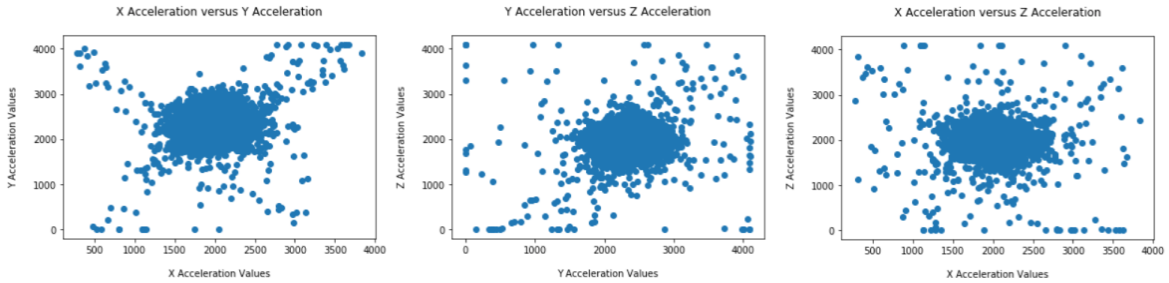
Next, we explore the relationships in between above features by defining and analysing few hypotheses.

4.2.2. Explore relationships between columns

Hypothesis 1: Considering the data for all the activities together, we cannot expect to see any significant correlation between the acceleration values for X, Y and Z.

Out[26]:

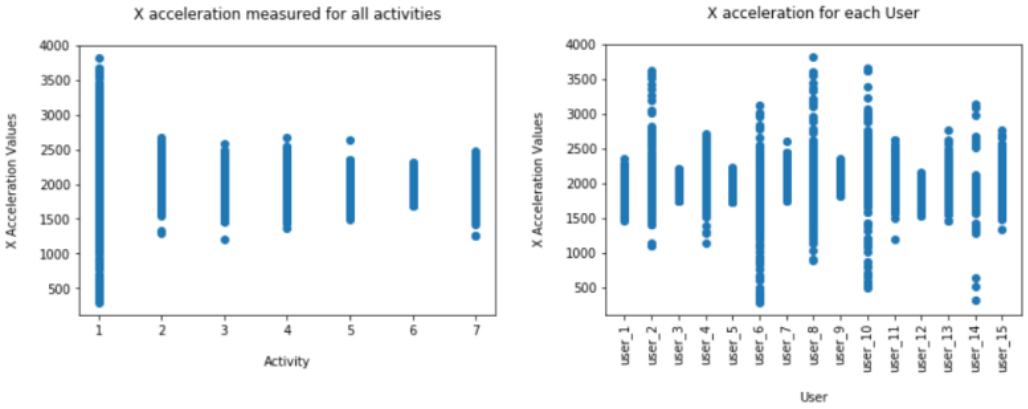
	X	Y	Z
X	1.00	0.36	0.01
Y	0.36	1.00	0.35
Z	0.01	0.35	1.00



As seen in the above results, for X and Y, we could see slight correlation on the edges for few values, but for majority of the values we did not have any correlation. This justified the correlation value of 0.36. For Y and Z, there was a slight upward trend visible for few values, but majority of them did not follow any trend. This justified the correlation value of 0.35. For X and Z, the correlation value was 0.01. Even looking at the scatter plot, we could see any correlation at all.

So, we concluded that our hypothesis 1 was correct.

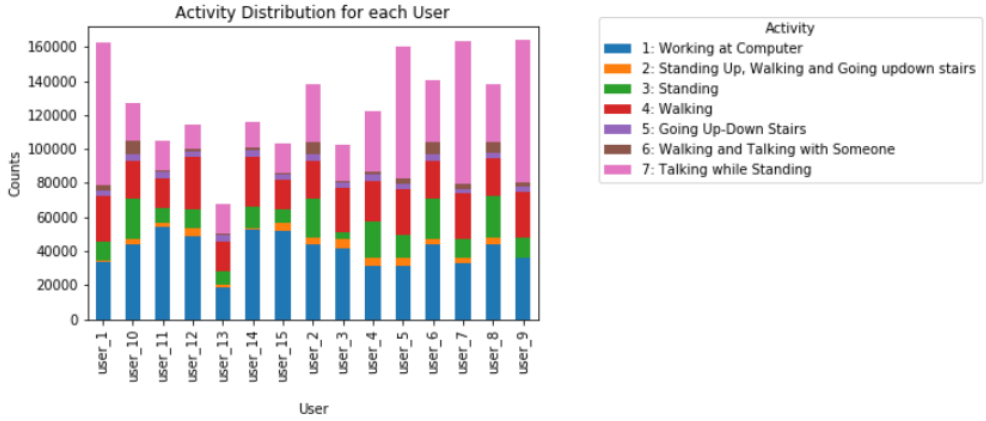
Hypothesis 2: The variation in X acceleration values might be different for each activity and each user based on the movements involved in every case.



As seen in the above output, we observed that for activity 1, the variation in X acceleration values was more which indicated that this activity involves more movement. Also, for users 6, 8 and 10, the variation of these values was more which indicated that these users performed more movements. So, we conclude that our 2nd hypothesis was correct.

Similarly, we plotted a scatter plot to check how are the Y and Z acceleration values related to Activity and User. From the plots we obtained similar results as above. For Y acceleration values, the variations were more for users 2, 6 and 10. For Z acceleration values, the variations were more for users 2, 6 and 8.

Hypothesis 3: As we observed earlier that the measurements were recorded during the office hours, the activity distribution for each user should more or less follow the same distribution.



As seen in above output, the distribution varied slightly from what we expected. The proportion of working at computer activity was as expected the highest for most number of users. But we also observed that the proportion for talking while standing activity was unexpectedly high for few users. So, we concluded that our 3rd hypothesis was only partially correct.

4.3. Predictive Modelling

In this section, we will look at the results obtained while performing the predictive modelling.

4.3.1. Split Train and Test Data

As outlined in the methodology, we had split the descriptive and target feature data in 2 parts, viz., train and test and stored this separated data in separate NumPy arrays *train_data*, *test_data*, *train_target* and *test_target.*

```
For Train Data:              For Test Data:
Rows: 70000 , Columns: 18    Rows: 30000 , Columns: 18

For Train Target:            For Test Target:
Rows: 70000                  Rows: 30000
```

From the above output we confirmed that the data had been split into train and test data in ratio of 70:30.

4.3.2. Feature Selection, Hyper-parameter Tuning and Training the Model

We carried out this step separately for both the k-Nearest Neighbours and Decision Tree models. As we used the pipe from Scikit-Learn package to identify the optimal values, we ranked the features using both f-score and mutual information techniques.

For both the models, we tried the fitting using 3, 10 and 18 features. The idea here was to first fit the models using only the X, Y and Z acceleration values. Then fit the model using these acceleration values along with half(7 users) of the users. And finally fit the model using all the features(X, Y and Z accelerations and all users).

When fitting the models, we used 7-fold stratified k-fold validation to evaluate the models for all feature combinations and hyper-parameter values. As we used 70,000 instances for training, by using 7-folds, we got

10,000 instances in every fold which is optimal to evaluate the model. We used the stratified version to maintain the frequency distribution of all categories in the target feature across all the 7 folds(partitions).

1. For k-Nearest neighbours

For this model, we chose to consider 1, 5, 10, 50 and 100 number of neighbours. The idea here was to find a balance between over-fitting and underfitting. By choosing values 1, 5 and 10 we tried to avoid over-fitting and consider minimal neighbours. For values 50 and 100, we ensured that model is not underfitted and that we considered more variation to be captured as we had 70,000 data points.

While choosing the value of p for Minkowski distance, we used 1 and 2. With these values we tried to calculate the distance using both Manhattan and Euclidean method.

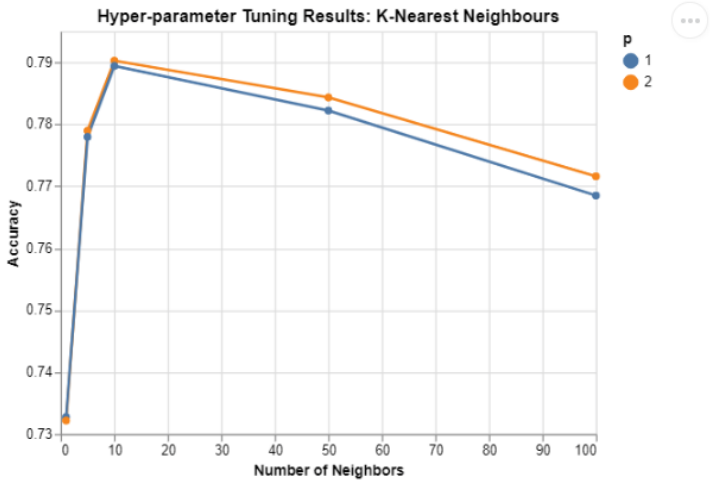
After defining all the above values, we fitted all the possible models using the pipe.

```
Out[43]: {'fselector__k': 18,
          'fselector__score_func':
          <function sklearn.feature_selection._univariate_selection.f_classif(X, y)>,
          'knn__n_neighbors': 10,
          'knn__p': 2}
```

The k-Nearest Neighbours model performed best for above values. From the above output, the model returned best performance by using all the 18 features. The optimal method to rank and select these features was f-score. Also, the model returned best performance where number of neighbours considered was 10 and the distance was calculated using the Euclidean method(p = 2).

To check how did the model perform for other hyper parameter values, we plotted a line graph.

Out[44]:



Looking at the line graph, the performance was at peak for 10 number of neighbours. The performance declined as the number of neighbours were increased. Also, in most of the cases, the distance calculated by Euclidean method(p = 2) gave better performance compared to distance the calculated using Manhattan method(p = 1).

2. For Decision Tree

Here, we first defined the methods using which node impurity was to be calculated as Gini and Entropy. Then, we mentioned the maximum depth values as 3, 5 and 10. The idea behind choosing these values was to find the balance between under-fitting and over-fitting again. Using values 3 and 5 we ensured that we are fitting the model with minimum depth. For value 10, we tried fitting to model to a greater degree.

The same idea was applicable while choosing the minimum number of instances to be present for splitting the node. We chose the values 50, 100 and 500 for this hyper-parameter.

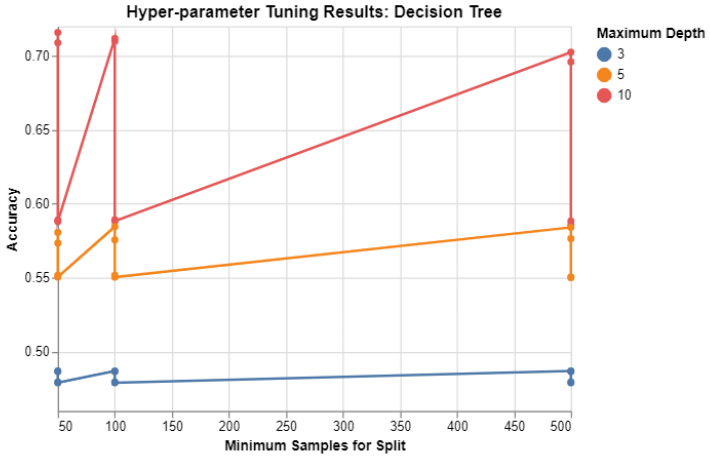
After defining all the above values, we fitted all the possible models using the pipe.

```
Out[46]: {'dt__criterion': 'gini',
          'dt__max_depth': 10,
          'dt__min_samples_split': 50,
          'fselector__k': 10,
          'fselector__score_func':
          <function sklearn.feature_selection.mutual_info.mutual_info_classif(X, y, discrete_features='auto',
         n_neighbors=3, copy=True, random_state=None)>}
```

We got above values for which the model performed best. For decision tree model, it performed best by using just 10 features out of 18 and mutual information method turned out to be optimal while ranking these features. For calculating the node impurity, the model performed best when the node impurity was calculated using the Gini impurity index. Similarly, the optimal value which we got for max depth was 10 and that for the minimum number of instances for split was 50.

To check how did the model perform for other hyper parameter values, we plotted a line graph.

Out[47]:



Looking at the line graph, the performance fluctuates a bit between 50 and 100 number of minimum samples for splits. As we increase this value, initially the performance drops and then gradually increases. Also, in all the cases, the performance for a decision tree of depth 10 is highest. Looking at the plot, we can also say that, the performance increases as the depth of the tree increases.

4.3.3. Model Evaluation

Using the optimal features and hyper-parameters, we fitted the model on test data to evaluate the performance. We got below 2 confusion matrices for each model. First matrix is for k-Nearest Neighbours and second is for Decision Tree.

```
array([[9234,   45,   38,  122,    8,    1,   86],
       [ 323,  172,   39,  133,    4,    3,   73],
       [ 145,   22, 1761,  664,   60,   34,  698],
       [ 272,   27,  418, 4309,   32,   19,  515],
       [  37,    4,  191,  328,   99,    9,  135],
       [   8,    1,   91,  143,   32,  164,  312],
       [ 189,   22,  482,  416,   30,  101, 7949]])
```

```
array([[8858,   35,   93,  317,    3,    0,  228],
       [ 355,   66,   28,  197,    2,    1,   98],
       [ 250,    5, 1382,  835,   20,   13,  879],
       [ 514,    7,  428, 3767,   29,   14,  833],
       [  85,    0,  198,  336,   34,    2,  148],
       [  25,    0,  191,  153,    1,   41,  340],
       [ 316,    1,  810,  679,    5,   41, 7337]])
```

When we look at the diagonals for above 2 confusion matrices, the first matrix for k-Nearest Neighbours seems to be performing better. To verify this, we created a classification report which tells us the performance of each model in terms of multiple metrics like f1-score, recall and precision.

	precision	recall	f1-score	support
1	0.90	0.97	0.94	9534
2	0.59	0.23	0.33	747
3	0.58	0.52	0.55	3384
4	0.70	0.77	0.74	5592
5	0.37	0.12	0.19	803
6	0.50	0.22	0.30	751
7	0.81	0.87	0.84	9189
accuracy			0.79	30000
macro avg	0.64	0.53	0.55	30000
weighted avg	0.77	0.79	0.77	30000

	precision	recall	f1-score	support
1	0.85	0.93	0.89	9534
2	0.58	0.09	0.15	747
3	0.44	0.41	0.42	3384
4	0.60	0.67	0.63	5592
5	0.36	0.04	0.08	803
6	0.37	0.05	0.10	751
7	0.74	0.80	0.77	9189
accuracy			0.72	30000
macro avg	0.56	0.43	0.43	30000
weighted avg	0.69	0.72	0.69	30000

As seen in above output, the first classification report for k-Nearest Neighbours demonstrates better performance. To get a final verdict, we displayed the overall accuracy for both the models.

```
Accuracy for K-Nearest Neighbours model: 0.7896
Accuracy for Decision Tree model: 0.7162
```

From above output, the k-Nearest Neighbours model performed better.

5. Discussion

After looking at the confusion matrix and classification report in above results, we know that the performance of both the k-Nearest Neighbours and Decision Tree are not exceptionally good. There is clearly a scope for improvement. Here, we will discuss the limitations for this approach and how can those be fixed.

1. First, for modelling 20 million instances, a sample of 100,000 is not suitable. With this sample size, it is not possible to capture all the variation in the data. Using a bigger sample size might improve the accuracy.
2. Second, due to technical limitations, for tuning the hyper-parameters, we could try only a limited number of values. By trying more values, we might be able to find values which provide better performance.
3. Third, by using different classification models like Random Forests, Naïve-Bayes, Support Vector Machine, etc. the accuracy could have improved.

6. Conclusion

After completing the predictive analysis on the accelerometer data, we found that the k-Nearest Neighbour classifier is performing better than Decision Tree at an overall accuracy of **0.7896**. So, the **k-Nearest Neighbour classifier** can be used for fitting the model on entire data set.

7. References

1. UCI Machine Learning Repository, (2012). Activity Recognition from Single Chest-Mounted Accelerometer Data Set. Available at https://archive.ics.uci.edu/ml/datasets/Activity+Recognition+from+Single+Chest-Mounted+Accelerometer
2. Aksakalli, Vural. (2019). get_search_results() function, Case Study Predicting Income Status[GitHub Repository]. Available at https://www.featureranking.com/tutorials/machine-learning-tutorials/case-study-predicting-income-status/
3. Dr. Yongli Ren, (2020). Module Notes for Practical Data Science. Available on Canvas at https://rmit.instructure.com/courses/67430/modules