



AY: 2020/21

CIS7031 - Programming for Data Analysis

Module Leader: Dr Ambikesh Jayal

**Title: Developing and evaluating a prediction model using
various data science techniques**

By

Rohit Saini

MSc Data Science (Internship)

St20183550

Abstract

Activity recognition plays an important role in today's world. In this project I have designed many models which recognize a particular activity using smartphone data. Accelerometer and Gyroscope sensor of the smartphones is used to collect the data. Jupyter notebook is used to clean and for further processing. Different deep learning and machine learning techniques are used to develop a predictive model. The main objective is to classify one of the three activities performed i.e., Walking, Shaking and Dropping. Within the recorded data, I started with analyzing the data using data exploratory analysis, then applied machine learning algorithm like Decision Tree, SVM, Random Forest etc. As the data contains sequential values, I have developed Neural network model using Keras and TensorFlow.

Contents	Page No.
1. Abstract	i
2. Introduction	2
3. Workflow of the Project	3
4. Data Preparation and Pre-processing	4
5. Data Exploration	6
6. Implementing Machine Learning Algorithms	10
7. Implementing Neural Network with TensorFlow and Keras	12
8. Results	17
9. Conclusion and Future Scope	20
10. References	21
11. Appendix	23

Introduction

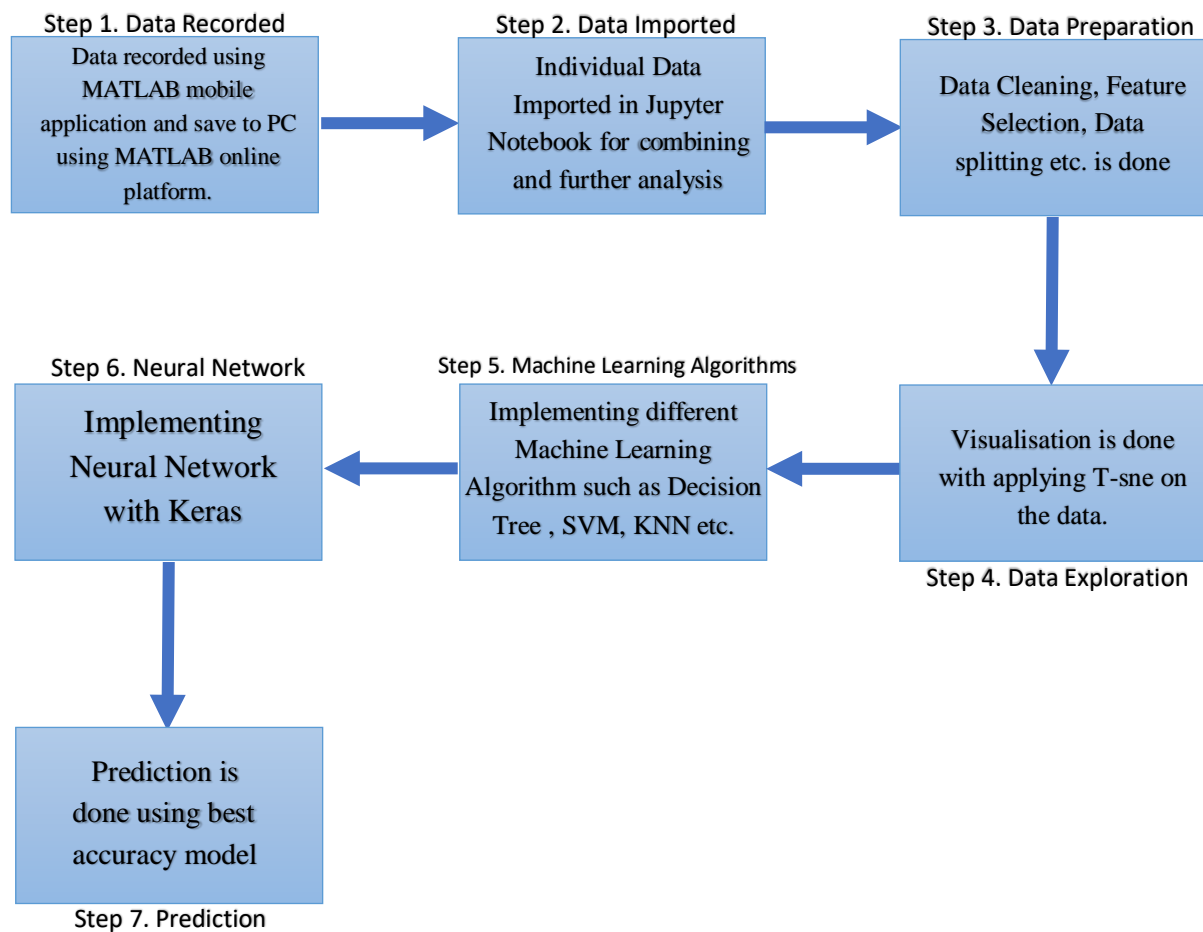
In today's world, Data is everywhere and is affecting almost every type of business. Data science is the process of converting both structured and unstructured data into suitable insights with the help of various techniques. It is all about using the data to solve a particular problem.

Machine learning is a subgroup of Artificial Intelligence. Different algorithms and models are used to analyse and learn data to make useful decisions without human interference. Machine learning gives machines the ability and capability to learn, it allows them to increase the accuracy and score of algorithms by making predictions on their own, but it requires a large amount of data that should also be of good quality (DataFlair Team, 2019).

Deep Learning is a part of machine learning which teaches machines to learn by example. It enables the computer to make a hierarchy of complicated tasks by breaking it into simpler parts. Deep learning models are generally trained by giving them a large set of data and neural network architecture which allows them to learn useful techniques and features directly from the data without being manually extracted (Mathworks.com, 2019). The same neural network-based model can be used for many different applications and problems, but it requires complex data models and is extremely expensive, thus increasing the cost for the users (Rfwireless-world.com, 2012).

Smartphones have become a very important part of human's life. Almost every person carries a smartphone with them all the time. These smartphones have built-in sensors that enable them to collect data and detect the activity done by people. The main scope of this project is to analyse mobile sensor data in the context of activity recognition. The main objective is to create a Machine learning and Deep learning model using smartphone sensor data that classifies whether an individual is walking, shaking something or dropping something. For this project MATLAB mobile application is used to record the data.

Workflow of the Project



These are the main steps that are implemented in this project. Each of these steps has split into several sub steps. Let's have a look on how these steps are implemented.

Data Preparation and Pre-processing

How data was recorded

For the purpose of this project the data was recorded with the help of MATLAB mobile application using the sensors (Accelerometer and Gyroscope) of a smartphone (Iphone 11). The sensors have captured Acceleration, Angular Velocity, Orientation and Magnetic Field of the activities (walking, shaking and dropping) for a duration of two minutes. The MATLAB mobile application generates a MATLAB .mat file which further be required to use in MATLAB desktop application to run and download data as an Excel file. (See Appendix A1)

Importing Data

Individual Excel file of all three activities with all sensor signals was imported in Jupyter notebook with the help of Pandas libraries. Different sensor signals are then combined in a single data frame of a particular activity with adding a label (Activity) column in the dataset. All three activities have given a different label of 1,2 or 3. Finally, all three activities dataset file combine to one single file for cleaning and further analysis. (See Appendix A2)

Data Cleaning

Cleaning of data is an essential step in machine learning because the accuracy of the results depends on the data we use. It is the process to make the data into such a state that the machine can parse it easily. Most of the time there will be discrepancies in the data we captured such as missing data, duplicate data, errors while capturing data, data formats. It could be as simple as leaving out an entry because of missing values or filling up missing values with either mean or median (Thomas, 2018).

For the purpose of this project, data cleaning is done on the combined data sets of all the three activities. Null values and duplicate values are checked with the help of python inbuilt functions. I got 0 duplicate values and 3 null values in the data set. As we are missing only 3 values so we can afford to remove them from the dataset. Now the data is cleaned and ready for features selection. (See Appendix A3)

Feature Selection

Selecting features in machine learning is the process of lowering the total number of input variables or selecting manually or automatically only those features which are relevant for the models. Feature selection minimizes the training time and increases the accuracy of the model (Raheel Shaikh, 2018).

In this project there are total 15 columns in the dataset. The columns are [(Timestamp), (Acc_X), (Acc_Y), (Acc_Z), (AngV_X), (AngV_Y), (AngV_Z), (Mag_X), (Mag_Y), (Mag_Z), (Orin_X), (Orin_Y), (Orin_Z), (Activity Name), (Activity)]. For the machine learning model 'Timestamp' and 'Activity Name' columns are not important as Activity Name is string and the machine learning model only satisfied with integers. So, for input variable 'X', I selected only relevant column i.e., "Acc, AngV, Mag, Orin]" for all three axis X, Y and Z. For dependent variable (y) "Activity" column is selected. (See Appendix A4)

Data Splitting

Data splitting is basically dividing the data into two portions. In machine learning, the purpose of splitting the data into train-test and train-validation is for evaluating the performance of the algorithm. Training dataset is a subset used to train the model, validation data is used to evaluate a model while tuning model hyperparameters and test dataset is used to test the trained model (Machine Learning Mastery, 2017).

In this project, the data is divided into X_train, X_val and X_test with 2354 rows in X_train data, 736 rows in X_test data and 589 rows in X_val data with the help of Sklearn inbuilt 'train_test_split' function. (See Appendix A5)

Data Normalization

Data normalization is a common technique used in machine learning. It is also termed as data transformation. Data Normalization basically means to transforming the data to the range of (0,1) so that it brings all the numeric columns of the data set to a common scale (Team, 2019). In this project all three datasets X_train, X_test, X_val get normalized. (See Appendix A6 and A7)

Data Exploration

With data preparation and pre-processing done, we moved on to data exploration. Data exploratory analysis is basically a visualisation technique that used to analyse and investigate the data.

To find the distribution in the dataset, I performed exploratory data analysis which gave us the data points (frequency of activities) on the training and test dataset.

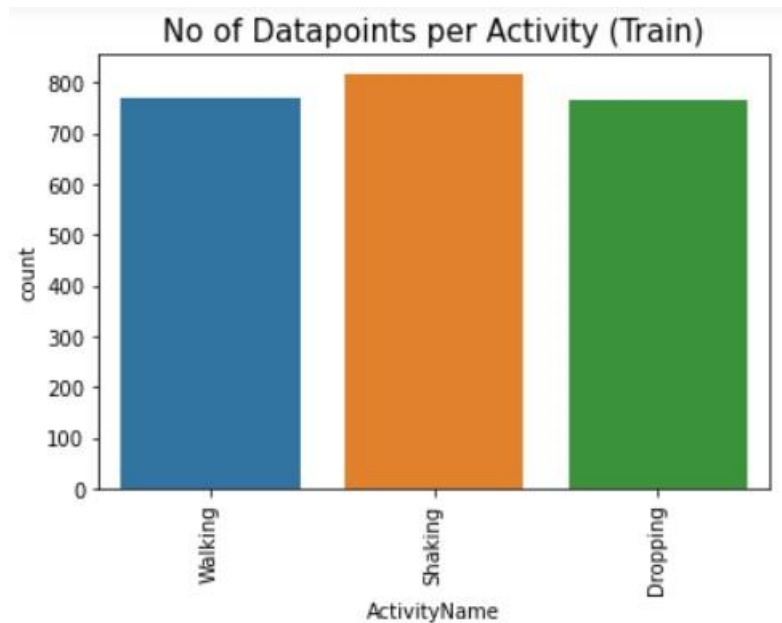


Fig 1

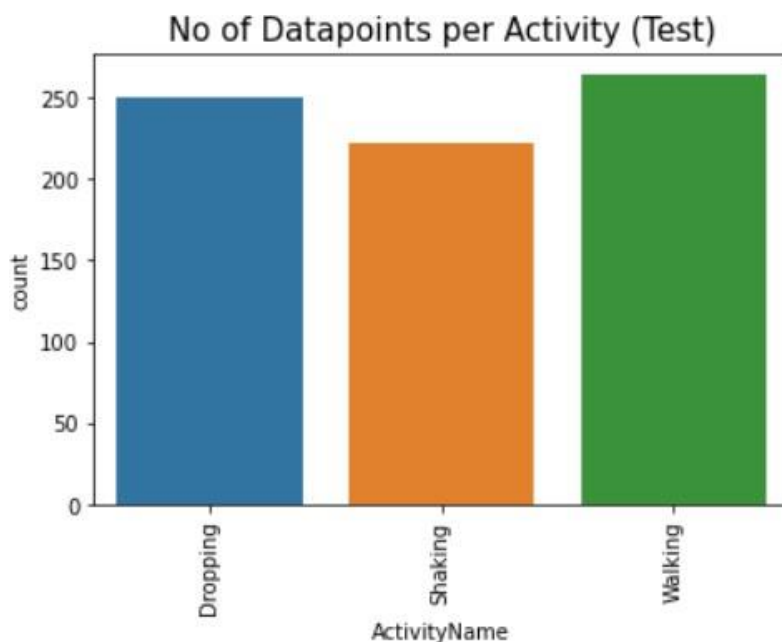


Fig 2

By analysing fig 1 and 2, we can say that the dataset is almost balanced.

Let us see how different types of activities look like. I made some plots for all inputs i.e., Acceleration, Angular Velocity, Orientation and Magnetic Field.

Acceleration Plot

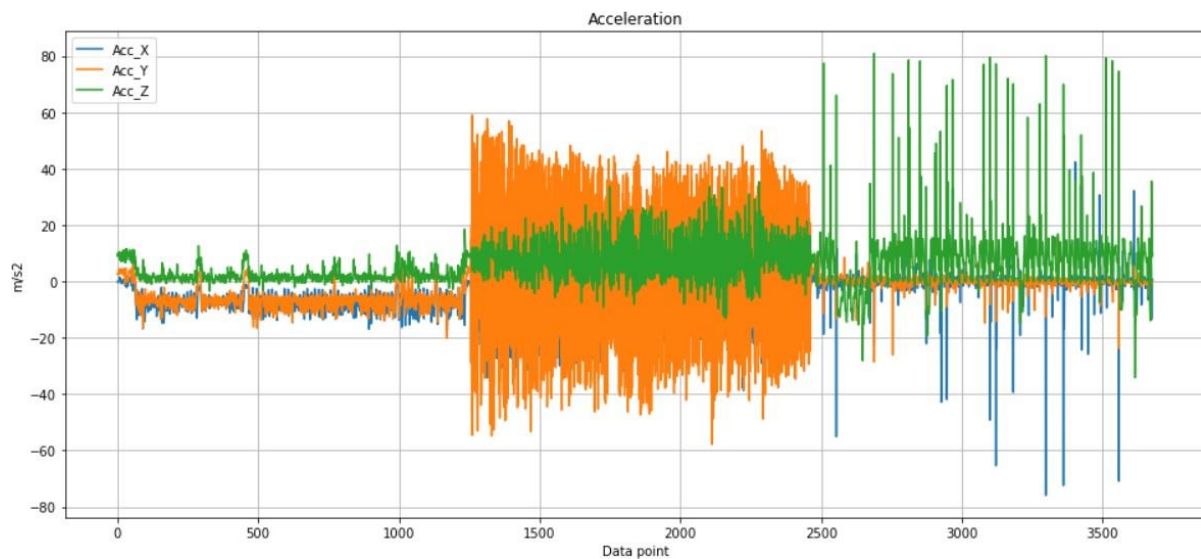


Fig 3

In fig 3 all three activities are combined for comparison between acceleration of the activities. The acceleration is measured by Accelerometer sensor of the smartphone. The type of activities can be classified by observing the data. The first set of waves indicates walking, the middle set of waves indicates shaking and the last one indicates dropping.

After analysing, walking is relaxed than shaking and dropping. Shaking looks much bouncier than other two.

Angular Velocity Plot

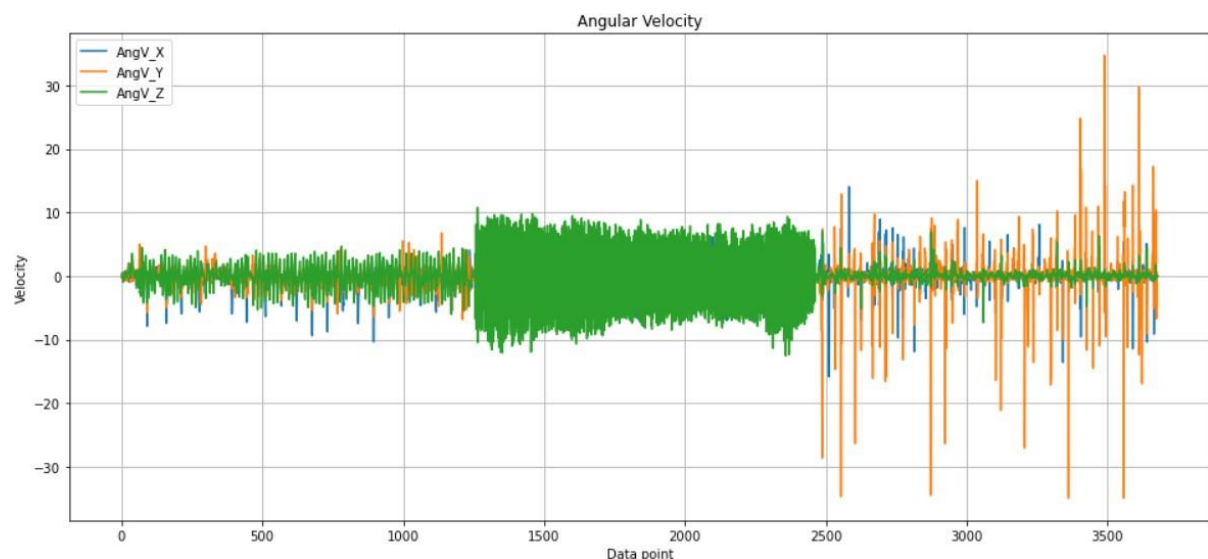


Fig 4

In fig 4 the activities are in the same order as they were before. Angular Velocity basically refers to how an object rotates which means how fast the orientation or angular position can change with the time.

After analysing this graph, we can say that while walking and shaking the assembly is rotating more about the Z axis whereas while dropping it is rotating more about the Y axis.

Magnetic Field Plot

The magnetic sensor in the smartphone is known as magnetometer which measures earth's magnetic field and the direction in which our phone is facing. The x, y and z attributes of magnetometer represents the magnetic field around X, Y and Z axis, respectively.

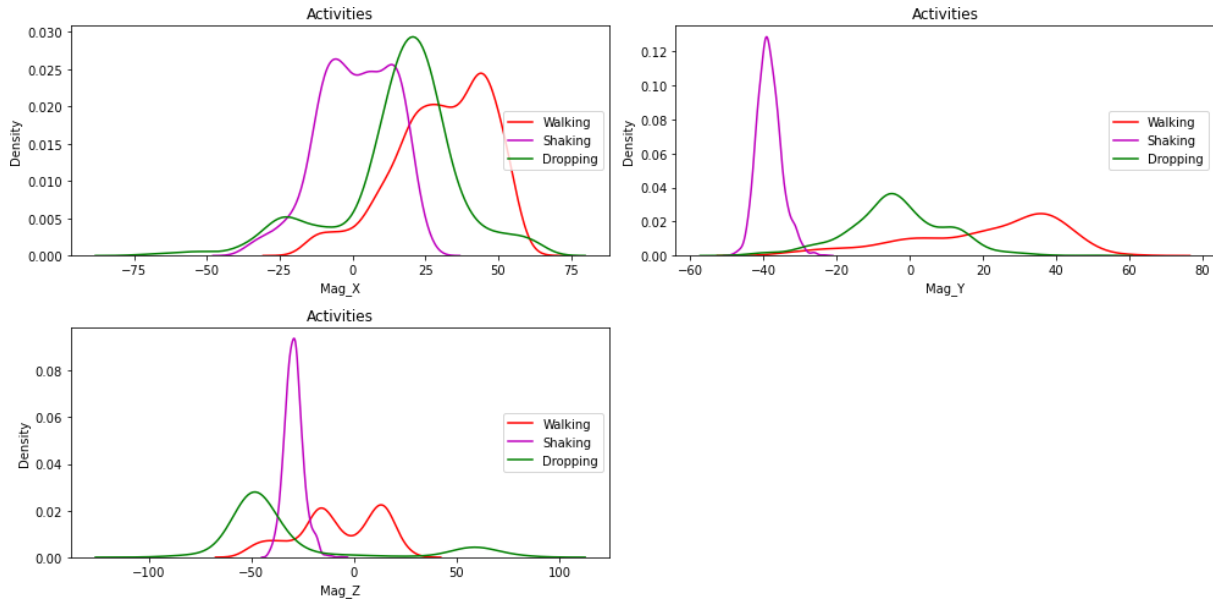


Fig 5

The results from fig. 5 shows that magnetic field around X axis is almost equal while doing all the activities. For Y and Z axis, shaking has more magnetic field than walking and dropping.

Orientation Plot

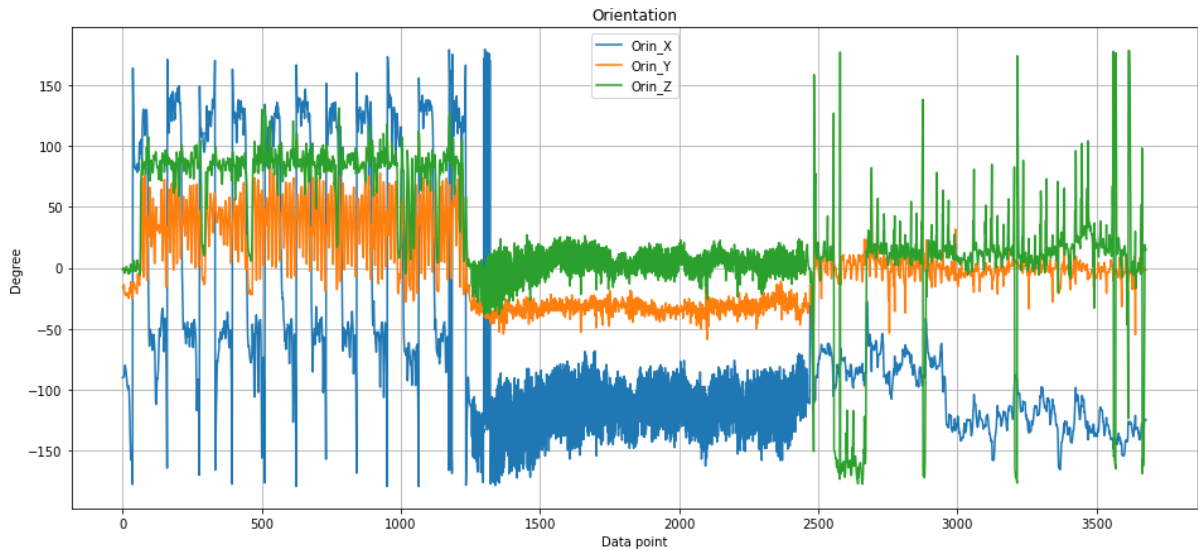


Fig 6

Smartphone measures the orientation angles with geomagnetic field sensor and accelerometer. The Orin X, Y and Z implies Azimuth, Pitch and Roll respectively which further implies degree

of rotation about the Z, X and Y axis, respectively. Fig 6 shows that walking has more degree of rotation than shaking and dropping.

Applying t-SNE on the data

t-SNE stands for t-distributed Stochastic Neighbour Embedding. It is a data exploration technique and helps in visualizing high dimensional data. T-SNE basically tells that how our data is arranged. If we do t-SNE in our classification and the result of t-SNE separates the classes very well. This indicates that it is possible to build the model that will also separate classes (Violante, 2018).

t-SNE has a tuneable parameter known as 'perplexity' which work with number of iterations to balance the data.

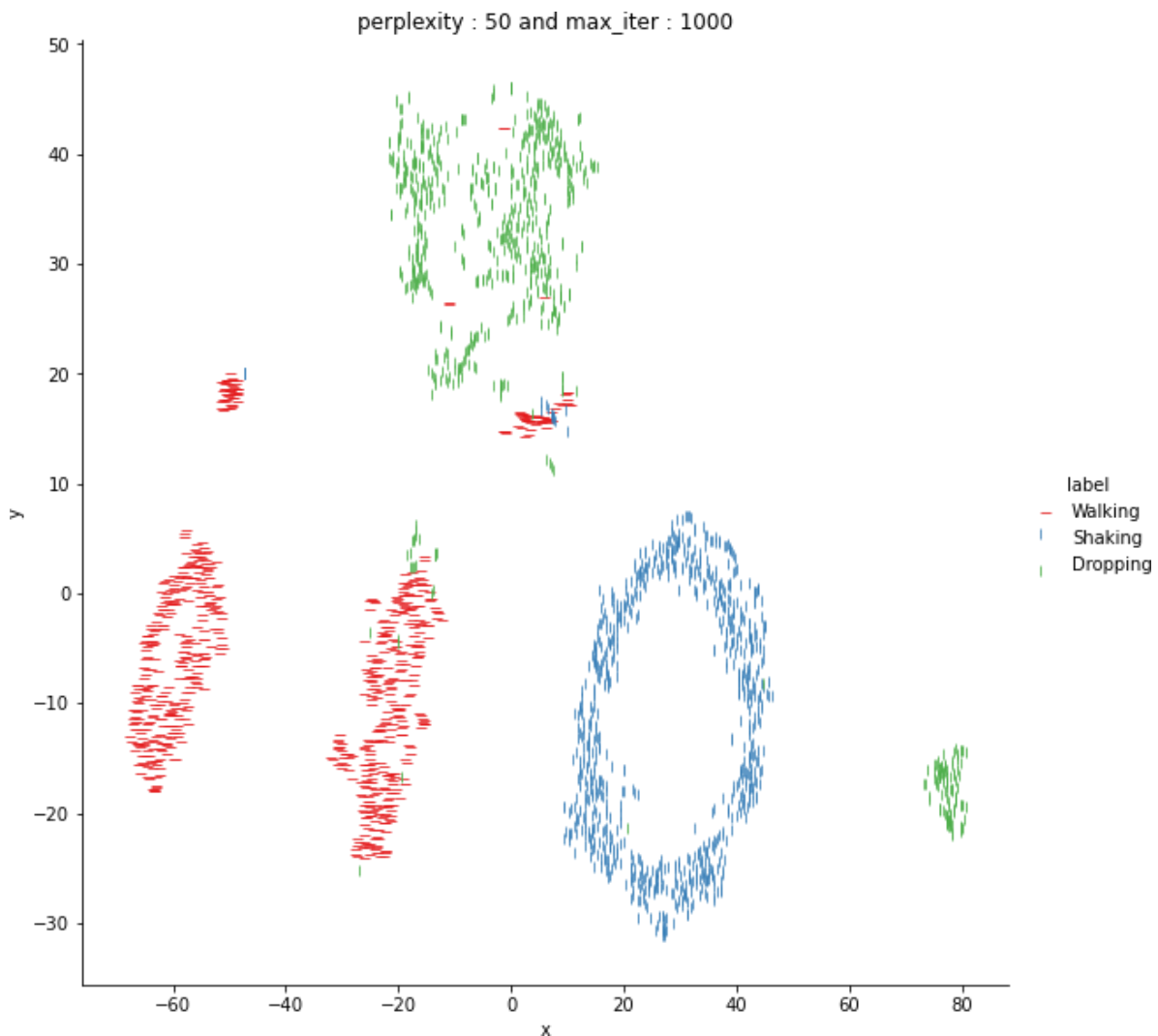


Fig 7

In this project t-SNE is done with perplexity 2,5,10,20 and 50 with 1000 iterations. Fig 7 shows the output of t-SNE with perplexity 50 and iteration 1000. It separates the activities well means it is possible to create and evaluate the prediction model with this data. (For every perplexity visualisation See Appendix A8)

Implementing Machine Learning Algorithm

With exploratory data analysis done, we moved on to implementing machine learning models on the dataset. Different algorithm is implemented to evaluate the accuracy and to choose which one is best for prediction. Sklearn packages are used to train the dataset and evaluate the model. Following is the algorithm implemented:

Decision Tree

Decision tree is the most commonly used algorithm under supervised learning that can be used for both regression and classification. It is a set of rules that can be used to classify the data. It is simple to interpret and understand, able to handle both categorical data and numerical data, it also requires less data preparation (Gupta, 2017). Multi-class classification can be done by using DecisionTreeClassifier

In this project for building the Decision tree model, 'sklearn.tree' package is used to import the DecisionTreeClassifier and for evaluating the accuracy, 'sklearn.metrics' package is used to import the accuracy_score. After training the model with X_train and Y_train, I got the accuracy of 95.5% with DecisionTreeClassifier. (See Appendix A9 and A10)

Logistic Regression

Another technique used in machine learning primarily for classification purposes is Logistic regression. Types of logistic regression are Multinomial logistic regression and Binary logistic regression (www.tutorialspoint.com, n.d.).

Binary logistic regression is designed only for two class problems whereas Multinomial logistic regression is a modified version of logistic regression that can be used for multi class problems. We can modify our logistic regression to multinomial logistic regression by setting the 'multi_class' and 'solver' argument to 'multinomial' and 'lbfgs' respectively.

In this project for building a multinomial logistic regression model, 'sklearn.linear_model' package is used to import 'LogisticRegression'. After training the model using X_train and Y_train, I got the accuracy of 93.2% using this model. (See Appendix A9 and A10)

K-Nearest Neighbors

K-Nearest Neighbors commonly known as KNN is an easy and simple algorithm which is used to implement both classification and regression problems. KNN can be used for multiclass classification. KNN is also known as a lazy algorithm because of inability to learn anything in the training period. As this algorithm does not requires any training before prediction, more data can be added into the dataset which will not affect the accuracy of the algorithm (Harrison, 2019).

For the purpose of this project, "sklearn.neighbors" package is used to import "KNeighborsClassifier". After training the model using X_train and Y_train, I got the accuracy of 98.5% using this model. (See Appendix A9 and A10)

SVM

SVM is known as Support Vector Machine. It is another supervised machine learning algorithm which can be used for both regression and classification problems. The normal SVM supports binary classification with two classes and does not support the multiclass classification. For multiclass classification some strategies must be applied. For multi-class classification, One-vs-Rest (OVR) is a great method for using binary SVM algorithm. It splits the multi-class dataset into several binary classification problems (Brownlee, 2020).

In this project, “sklearn.svm” package is used to import “SVC (support vector classifier)” and for training the multiclass dataset, the decision input shape is changed to “OVR”. After training the model using X_train and Y_train, I got the accuracy of 97.8% using SVM model. (See Appendix A9 and A10)

Random Forest

The Random Forest is basically an extension of decision tree algorithm and works on ensemble method. Random forest creates multiple different decision trees on the data, combined them at the end of the process to get a balanced and accurate prediction. Random forest has almost the same parameters as a decision tree and does not over fit as the number of trees increases (Wikipedia Contributors, 2019).

In this project, “sklearn.ensemble” package is used to import “RandomForestClassifier”. After training the model using X_train and Y_train, I got the accuracy of 96% using this model. (See Appendix A9 and A10)

Implementing Neural Network with TensorFlow and Keras

Neural Network are a Machine learning technique with layered structures. There are many layers in the neural network. The first and last layer is known as the input and output layer and the layers between these two layers are known as hidden layers or neurons.

TensorFlow

TensorFlow is an python library which is used for fast computing and implementation. TensorFlow can be used to create deep learning models directly or by using any other wrapper libraries. TensorFlow offers many different levels of ideas and concepts to build, compile and train neural network models. TensorFlow provides both high and low level APIs and gives more controls and flexibility with Keras (Choudhury, 2019).

Keras

Keras is a high level library in neural network that runs on the top of TensorFlow. Keras is a easy to understand API which has an easy interface. Keras API is developed for fast prototyping and implementation. As compared to TensorFlow, only high-level API is provided by Keras (Choudhury, 2019).

In this project two Neural network model is implemented using both TensorFlow and keras. The first model has only one hidden layer whereas the second model has 4 hidden layers. Both models are tuned for batch size, number of epochs and optimizer input.

Neural Network Model 1

The main steps for implementing neural network model in this project are as follows:

1. Importing the libraries

Frameworks plays an important role in data science. Frameworks are basically a set of libraries and packages which helps to make the programming experience simple. For the purpose of this project many libraries and packages are used to implement the neural network model. Fig 8 shows the packages and libraries used.

```
In [73]: import tensorflow
         from tensorflow import keras
         from tensorflow.keras import layers
         from keras.models import Sequential
         from keras.layers import Dense
         from sklearn.metrics import accuracy_score
         from keras.wrappers.scikit_learn import KerasClassifier
         from sklearn.model_selection import cross_val_score
         from sklearn.model_selection import KFold
         from sklearn.model_selection import GridSearchCV
```

Fig 8

2. Creating and Compiling the model

```
In [74]: def simplemodel(n_inputs=12, n_outputs=3, optimizerinput='adam'):

    model = Sequential()

    model.add(layers.Dense(n_inputs, input_dim=n_inputs, kernel_initializer='normal', activation='relu'))

    model.add(layers.Dense(2000, activation='relu'))

    model.add(layers.Dense(4, activation='softmax'))

    model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizerinput, metrics=['accuracy'])

    return model
```

Fig 9

- Simplemodel function is defined with 12 inputs (as there are 12 columns in X_train) and 3 outputs (as there are 3 activities to predict).
- Sequential model is initialized with an input layer, a single hidden layer and an output layer.
- For hidden layer and input layer activation “Relu” is used and for output layer, activation “softmax” is used.
- Relu function is used as better performance can be achieved using this.
- Softmax is used in output layer as this is a multiclassification model having 3 classes.
- As it is a classification problem the model is compiled using loss=sparse_categorical_crossentropy and metrics = accuracy.

3. Tuning batch size and number of epochs using Hyperparameter tuning.

Hyperparameter Tuning is done to tune the number of epochs and batch size using keras classifier and GridSearchCV by defining grid search parameters as epochs= [10, 50, 100, 500] and batch size= [10, 20, 40, 60, 80, 100]. Sklearn’s model_selection package has a function GridSearchCV which helps in looping through predefined parameters which fits the estimator on the training set. At last, GridSearchCV gives the best parameters out of the given listed hyperparameters. (See Appendix A11)

```
param_grid={'batch_size': [10, 20, 40, 60, 80, 100],
            'epochs': [10, 50, 100, 500]}
Best: 0.415915 using {'batch_size': 10, 'epochs': 100}
0.013866 (0.011004) with: {'batch_size': 10, 'epochs': 10}
0.237344 (0.291031) with: {'batch_size': 10, 'epochs': 50}
0.415915 (0.404127) with: {'batch_size': 10, 'epochs': 100}
0.253656 (0.310717) with: {'batch_size': 10, 'epochs': 500}
0.006797 (0.006043) with: {'batch_size': 20, 'epochs': 10}
0.344918 (0.252869) with: {'batch_size': 20, 'epochs': 50}
0.272967 (0.356669) with: {'batch_size': 20, 'epochs': 100}
0.265347 (0.327230) with: {'batch_size': 20, 'epochs': 500}
0.006797 (0.006043) with: {'batch_size': 40, 'epochs': 10}
0.104395 (0.112833) with: {'batch_size': 40, 'epochs': 50}
0.287607 (0.277986) with: {'batch_size': 40, 'epochs': 100}
0.271332 (0.345822) with: {'batch_size': 40, 'epochs': 500}
0.006797 (0.006043) with: {'batch_size': 60, 'epochs': 10}
0.053261 (0.039525) with: {'batch_size': 60, 'epochs': 50}
0.201981 (0.199403) with: {'batch_size': 60, 'epochs': 100}
0.237344 (0.291031) with: {'batch_size': 80, 'epochs': 10}
0.237344 (0.291031) with: {'batch_size': 80, 'epochs': 50}
0.237344 (0.291031) with: {'batch_size': 80, 'epochs': 100}
0.237344 (0.291031) with: {'batch_size': 80, 'epochs': 500}
0.237344 (0.291031) with: {'batch_size': 100, 'epochs': 10}
0.237344 (0.291031) with: {'batch_size': 100, 'epochs': 50}
0.237344 (0.291031) with: {'batch_size': 100, 'epochs': 100}
0.237344 (0.291031) with: {'batch_size': 100, 'epochs': 500}
```

Fig 10

Fig 10. Shows that from the above given parameters we got batch_size=10 and epochs=100 as the best parameters for training our model.

4. Tuning optimization algorithm using Hyperparameter tuning.

To tune optimization algorithm, hyperparameter tuning is done by defining Grid Search parameter with different optimizer input. (See Appendix A12)

```

Best: 0.343367 using {'optimizerinput': 'Adam'}
0.284644 (0.337840) with: {'optimizerinput': 'RMSprop'}
0.006797 (0.006043) with: {'optimizerinput': 'Adagrad'}
0.008157 (0.005768) with: {'optimizerinput': 'Adadelat'}
0.343367 (0.407473) with: {'optimizerinput': 'Adam'}
0.114187 (0.137869) with: {'optimizerinput': 'Adamax'}
0.227281 (0.267939) with: {'optimizerinput': 'Nadam'}

```

Fig 11

Fig 11 shows that after tuning our model for optimization algorithm we got Optimizer input = 'Adam' as best to use in our model.

5. Training and Evaluating the model

Training the Model

```

In [80]: model=simplemodel(len(X_train.columns),n_outputs=3, optimizerinput='Adam');
history = model.fit(X_train, y_train, verbose=1, epochs=100, batch_size=10, validation_data=(X_val, y_val))
Epoch 94/100

```

Fig 12

Evaluating the Model

```

In [82]: from numpy import argmax
loss, acc = model.evaluate(X_test, y_test, verbose=0)

print('Loss =',loss)
print('Accuracy =',acc)

Loss = 0.3024035394191742
Accuracy = 0.9130434989929199

```

Fig 13

I trained the model using the best parameter that I got from Hyperparameter tuning i.e., Batch size = '10', epochs = '100' and optimizer input = 'Adam' and got the accuracy of 91.3 % by evaluating the model with X_test and y_test.

Neural Network Model 2

In this model I have added more hidden layers to evaluate the accuracy and then compare it with the Model 1. The steps are similar as model 1 only the output is changed.

1. Defining and compiling the model

Implement Neural Network 2 (Adding more Hidden Layers) ¶

```
In [87]: def simplemodel2(n_inputs=12, n_outputs=3, optimizerinput='adam'):

    network = Sequential()

    network.add(layers.Dense(n_inputs, input_dim=n_inputs, kernel_initializer='normal', activation='relu'))
    network.add(layers.Dense(16, activation='relu'))
    network.add(layers.Dense(6, activation='relu'))
    network.add(layers.Dense(4, activation='relu'))
    network.add(layers.Dense(20, activation='relu'))
    network.add(layers.Dense(4, activation='softmax'))

    network.compile(loss='sparse_categorical_crossentropy', optimizer=optimizerinput, metrics=['accuracy'])

    return network
```

Fig 14

In fig 14, the model is defined and compile same as model 1 with adding more hidden layers.

2. Hyperparameter tuning for number of epochs and batch size.

In the second model, I defined grid search parameters as epochs= [10, 50, 100] and batch size= [10, 20, 40, 60, 80, 100]. (See Appendix A13)

```
param_grid = {'batch_size': [10, 20, 40, 60, 80, 100],
              'epochs': [10, 50, 100]}

Best: 0.159598 using {'batch_size': 20, 'epochs': 100}
0.006797 (0.006043) with: {'batch_size': 10, 'epochs': 10}
0.007613 (0.006006) with: {'batch_size': 10, 'epochs': 50}
0.141379 (0.184396) with: {'batch_size': 10, 'epochs': 100}
0.006797 (0.006043) with: {'batch_size': 20, 'epochs': 10}
0.006797 (0.006043) with: {'batch_size': 20, 'epochs': 50}
0.159598 (0.215407) with: {'batch_size': 20, 'epochs': 100}
0.006797 (0.006043) with: {'batch_size': 40, 'epochs': 10}
0.007613 (0.006006) with: {'batch_size': 40, 'epochs': 50}
0.149810 (0.202137) with: {'batch_size': 40, 'epochs': 100}
0.006797 (0.006043) with: {'batch_size': 60, 'epochs': 10}
0.006797 (0.006043) with: {'batch_size': 60, 'epochs': 50}
0.006797 (0.006043) with: {'batch_size': 60, 'epochs': 100}
0.006797 (0.006043) with: {'batch_size': 80, 'epochs': 10}
0.006797 (0.006043) with: {'batch_size': 80, 'epochs': 50}
0.006797 (0.006043) with: {'batch_size': 80, 'epochs': 100}
0.006797 (0.006043) with: {'batch_size': 100, 'epochs': 10}
0.006797 (0.006043) with: {'batch_size': 100, 'epochs': 50}
```

Fig 15

Fig 15. Shows that from the given parameters we got batch_size=20 and epochs=100 as the best parameters for training our model.

3. Hyperparameter tuning to tune optimization algorithm.

```
184/184 [=====] - 0s 2ms/step -  
Best: 0.171546 using {'optimizerinput': 'Adam'}  
0.087547 (0.111221) with: {'optimizerinput': 'RMSprop'}  
0.004078 (0.005768) with: {'optimizerinput': 'Adagrad'}  
0.004078 (0.005768) with: {'optimizerinput': 'Adadelat'}  
0.171546 (0.194841) with: {'optimizerinput': 'Adam'}  
0.016313 (0.014033) with: {'optimizerinput': 'Adamax'}  
0.126692 (0.148725) with: {'optimizerinput': 'Nadam'}
```

Fig 16

Fig 16 shows that after tuning our model for optimization algorithm we got Optimizer input = 'Adam' as best to use in our model.

4. Training and Evaluating the model

Training the model

```
In [90]: network=simplemodel2(len(X_train.columns),n_outputs=3, optimizerinput='Adam');  
history = network.fit(X_train, y_train, verbose=1, epochs=100, batch_size=20, validation_data=(X_val, y_val))
```

Fig 17

Evaluating the model

```
In [91]: from numpy import argmax  
loss, acc = network.evaluate(X_test, y_test, verbose=0)  
  
print('Loss =',loss)  
print('Accuracy =',acc)  
  
Loss = 0.09552818536758423  
Accuracy = 0.96875
```

Fig 18

I trained the model using the best parameter that I got from Hyperparameter tuning i.e., Batch size = '20', epochs = '100' and optimizer input = 'Adam' and got the accuracy of 96.8 % by evaluating the model with X_test and y_test.

Results

Graph for Train Loss vs Validation Loss (Model 1)

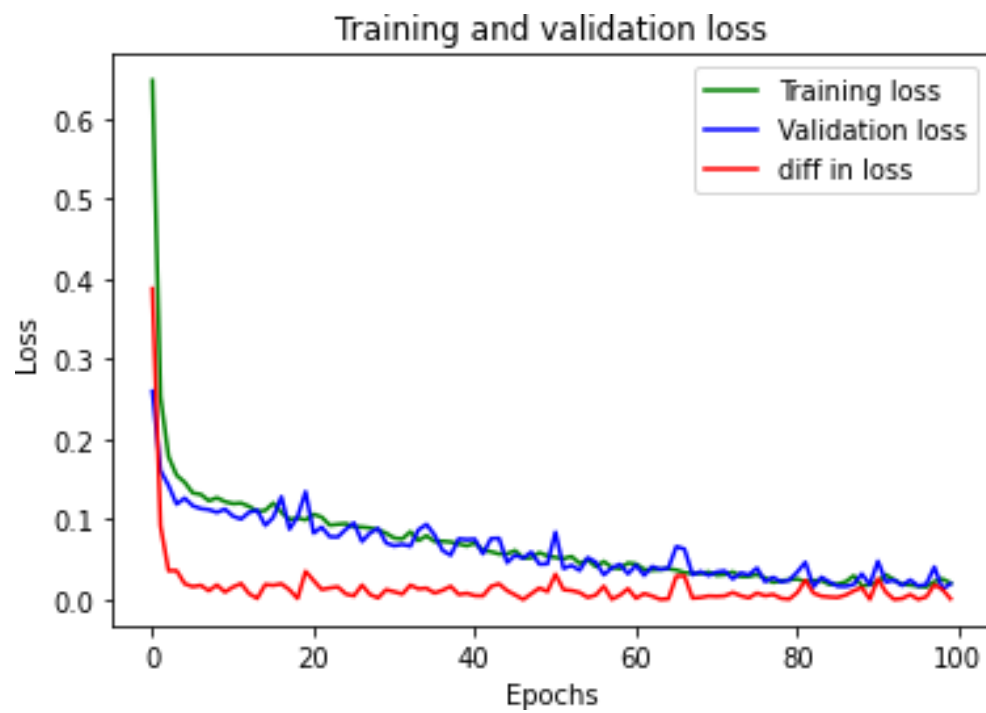


Fig 19

Graph for and Train Accuracy vs Validation Accuracy (Model 1)

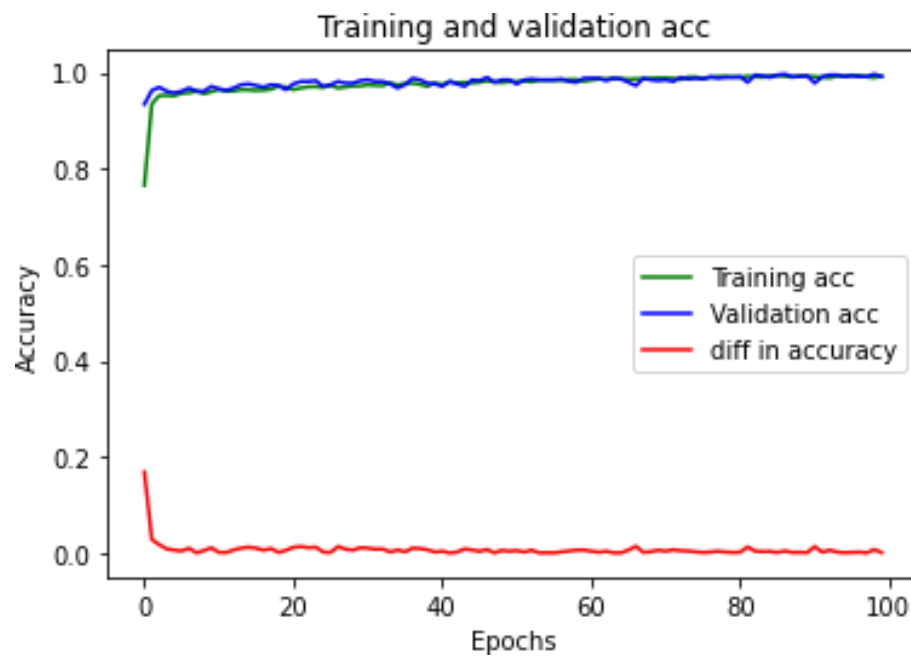


Fig 20

Fig 19 and 20 depicts a clear correlation between training loss and the validation loss. They both tend to reduce and remain at a constant value. Both training accuracy and the validation accuracy increases and stays similar to each other. This means that this model is trained very well.

Graph for Train Accuracy vs Validation Accuracy (Model 2)

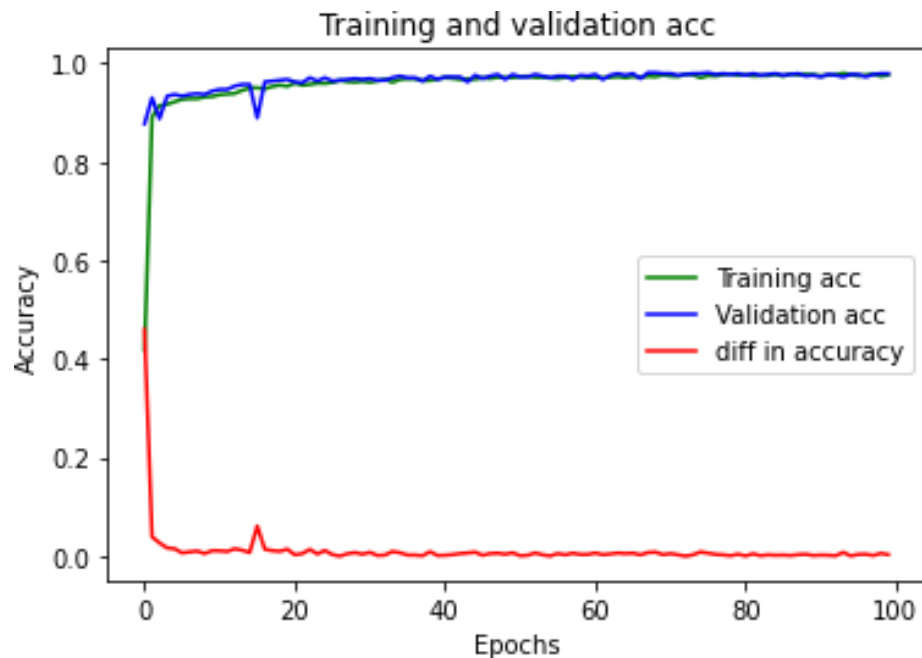


Fig 21

Graph for Train Loss vs Validation Loss (Model 2)

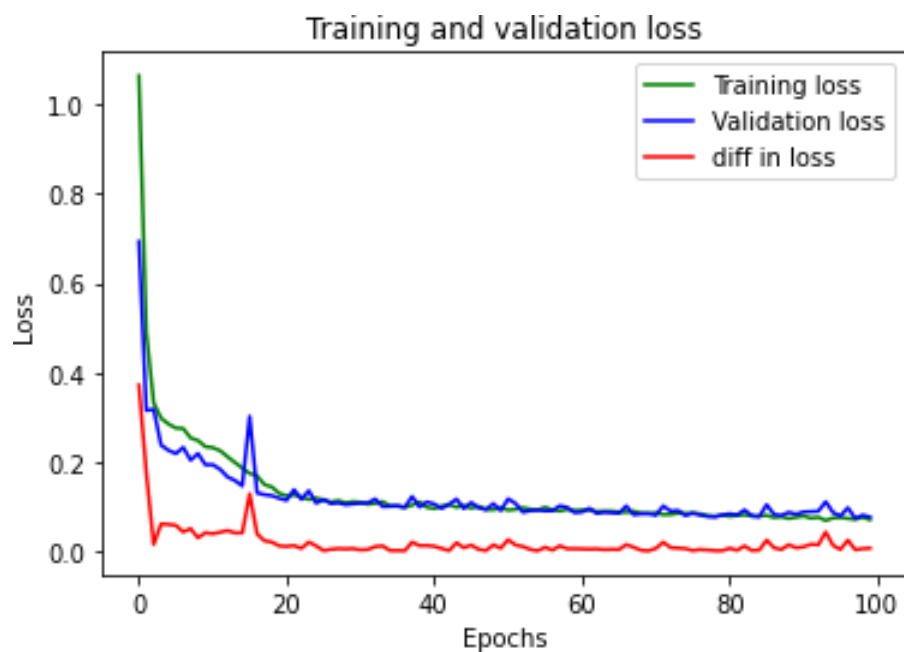


Fig 22

According to Fig 21 and 22, this model is trained well as there is no overfitting and underfitting occurs. Both training accuracy and validation accuracy increases and stays similar to each other. Training loss and accuracy loss also co relating to each other.

Accuracy results of all the models

Model	Accuracy
Decision Tree	95.5%
KNN	98.5%
SVM	97.8%
Random Forest	96%
Logistic Regression	93.2%
Neural Network 1	91.3%
Neural Network 2	96.8%

As from above table we can see that the accuracy of all the models is over 90%. We can predict value from any of the model but as KNN has the highest accuracy (98.5%) among all then we should select this for predicting our data.

Predicting only KNN model as it has the highest accuracy ¶

```
In [106]: clf=KNeighborsClassifier()
knn_model = clf.fit(X_train, y_train)
knn_model = knn_model.predict(X_test.head(10))

print(y_test.head(10))
print("The predictions are")
print(knn_model)
np.savetxt('X_test_predicted.csv', knn_model, delimiter=',')
```

```
3596    3
2826    3
1792    2
1794    2
2099    2
1552    2
3102    3
772     1
872     1
2876    3
Name: Activity, dtype: int64
The predictions are
[3 3 2 2 2 2 3 1 1 3]
```

After predicting X_test with KNN we get the first 10 prediction as [3,3,2,2,2,2,3,1,1,3] same as of y_test. That indicates our model is working fine.

Conclusion

Implementation of both standard machine learning algorithms and neural network model predict the classification problem better and accurate. As evident from the above results, all the models have a great accuracy. For machine learning algorithms, KNN has the highest accuracy and for neural network model, adding more hidden layers increases the accuracy of the second model as compared to first model.

Future Scope

I plan to implement the following as a future scope:

1. Inclusion of more activities.
2. Development of android app.

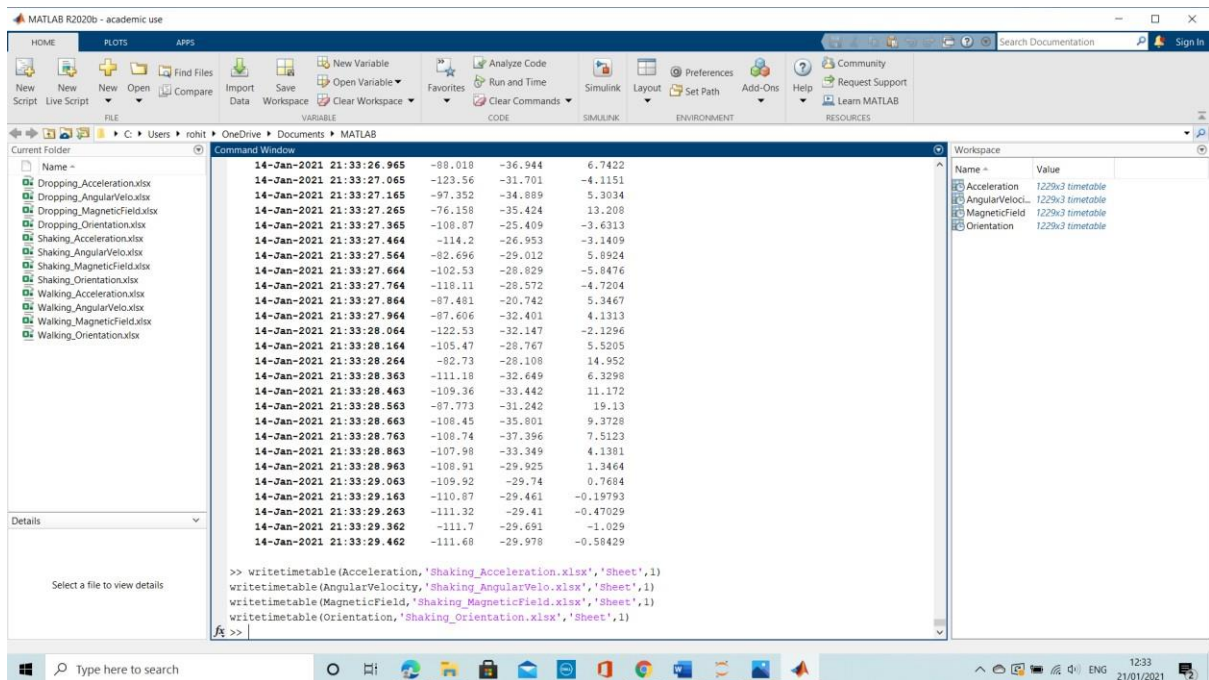
References

1. DataFlair Team (2019). *Advantages and Disadvantages of Machine Learning Language - DataFlair*. [online] DataFlair. Available at: <https://data-flair.training/blogs/advantages-and-disadvantages-of-machine-learning/> [Accessed 18 Jan. 2021].
2. Mathworks.com. (2019). *What Is Deep Learning? | How It Works, Techniques & Applications*. [online] Available at: <https://uk.mathworks.com/discovery/deep-learning.html> [Accessed 18 Jan. 2021].
3. Rfwireless-world.com. (2012). *Advantages of Deep Learning | disadvantages of Deep Learning*. [online] Available at: <https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-Deep-Learning.html> [Accessed 18 Jan. 2021].
4. Thomas, S. (2018). *crazyblog*. [online] Einfochips.com. Available at: <https://www.einfochips.com/blog/data-cleaning-in-machine-learning-best-practices-and-methods/> [Accessed 18 Jan. 2021].
5. Raheel Shaikh (2018). *Feature Selection Techniques in Machine Learning with Python*. [online] Medium. Available at: <https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e> [Accessed 18 Jan. 2021].
6. Machine Learning Mastery. (2017). *What is the Difference Between Test and Validation Datasets?* [online] Available at: <https://machinelearningmastery.com/difference-test-validation-datasets/> [Accessed 18 Jan. 2021].
7. Team, T.A. (2019). *How, When, and Why Should You Normalize / Standardize / Rescale Your Data? – Towards AI — The Best of Tech, Science, and Engineering*. [online] Available at: <https://towardsai.net/p/data-science/how-when-and-why-should-you-normalize-standardize-rescale-your-data-3f083def38ff> [Accessed 18 Jan. 2021].
8. Violante, A. (2018). [online] Towards Data Science. Available at: <https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1> [Accessed 19 Jan. 2021].
9. Gupta, P. (2017). *Decision Trees in Machine Learning*. [online] Towards Data Science. Available at: <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052> [Accessed 19 Jan. 2021].

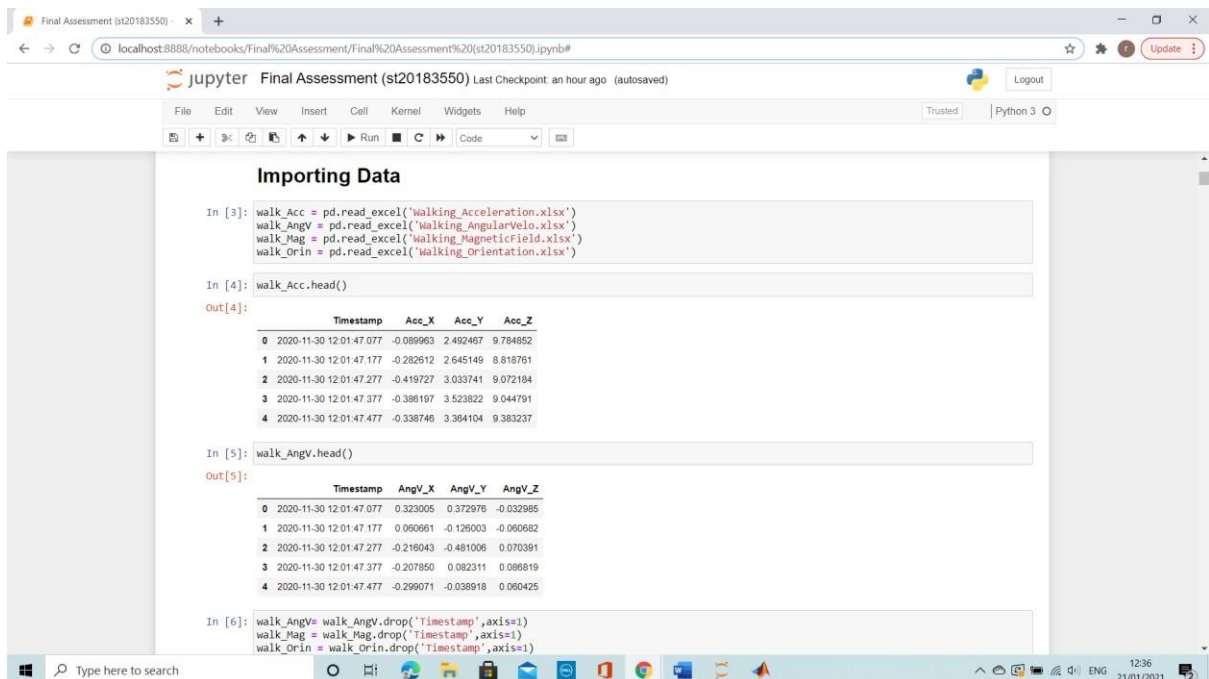
10. www.tutorialspoint.com. (n.d.). *Machine Learning - Logistic Regression - Tutorialspoint*. [online] Available at: https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_logistic_regression.htm#:~:text=Logistic%20regression%20is%20a%20supervised [Accessed 19 Jan. 2021].
11. Harrison, O. (2019). *Machine Learning Basics with the K-Nearest Neighbors Algorithm*. [online] Medium. Available at: <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761#:~:text=Summary-> [Accessed 19 Jan. 2021].
12. Brownlee, J. (2020). *One-vs-Rest and One-vs-One for Multi-Class Classification*. [online] Machine Learning Mastery. Available at: [https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/#:~:text=One%2Dvs%2Drest%20\(OvR%20for%20short%2C%20also%20referred](https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/#:~:text=One%2Dvs%2Drest%20(OvR%20for%20short%2C%20also%20referred) [Accessed 19 Jan. 2021].
13. Wikipedia Contributors (2019). *Random forest*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Random_forest [Accessed 19 Jan. 2021].
14. Choudhury, A. (2019). *TensorFlow vs Keras: Which One Should You Choose*. [online] Analytics India Magazine. Available at: <https://analyticsindiamag.com/tensorflow-vs-keras-which-one-should-you-choose/#:~:text=Keras%20is%20a%20neural%20network> [Accessed 20 Jan. 2021].

Appendix

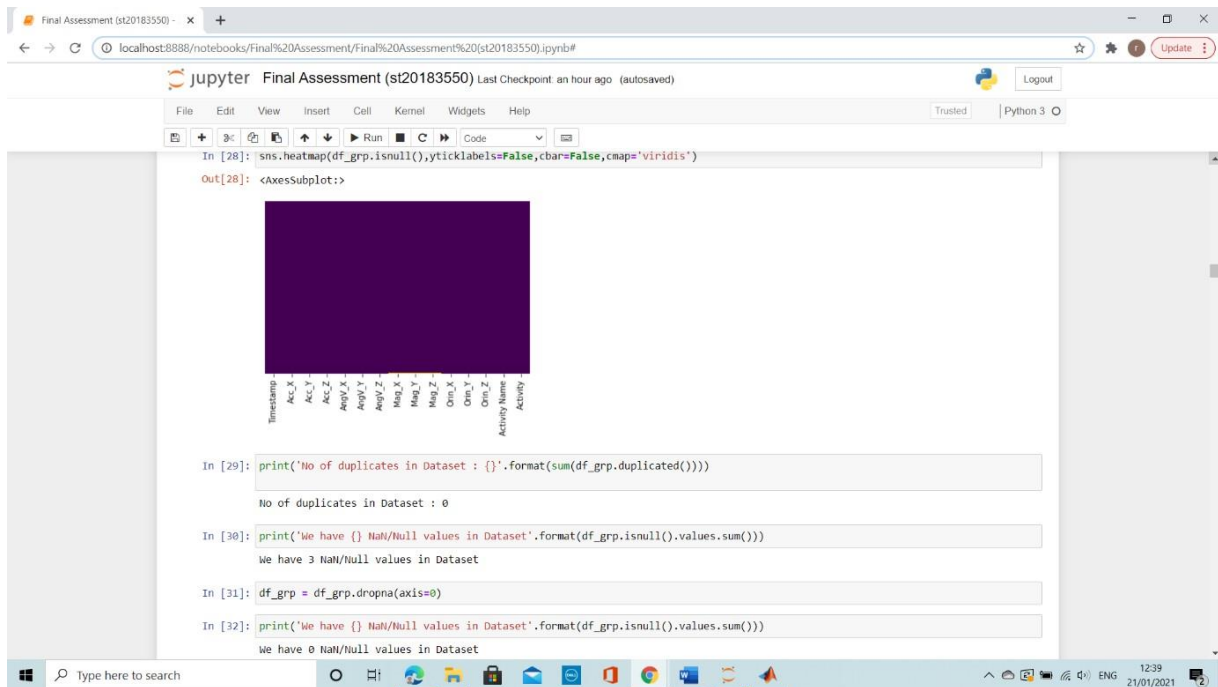
Appendix A1. MATLAB Desktop Application



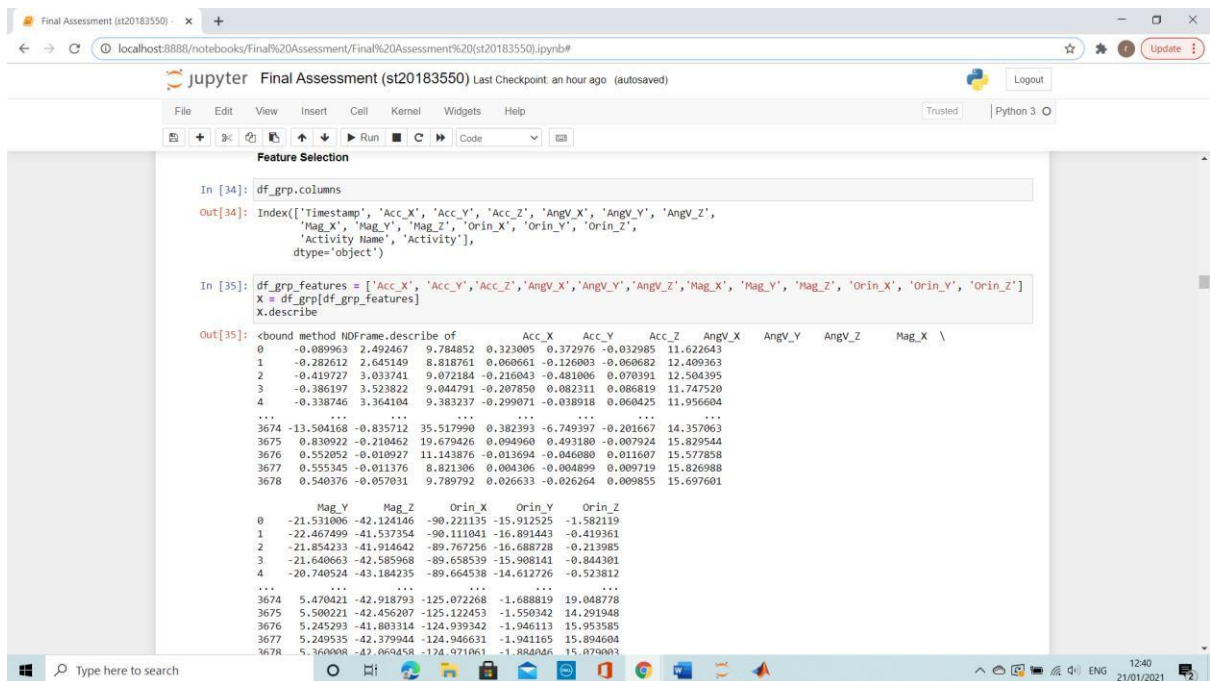
Appendix A2. Importing Data



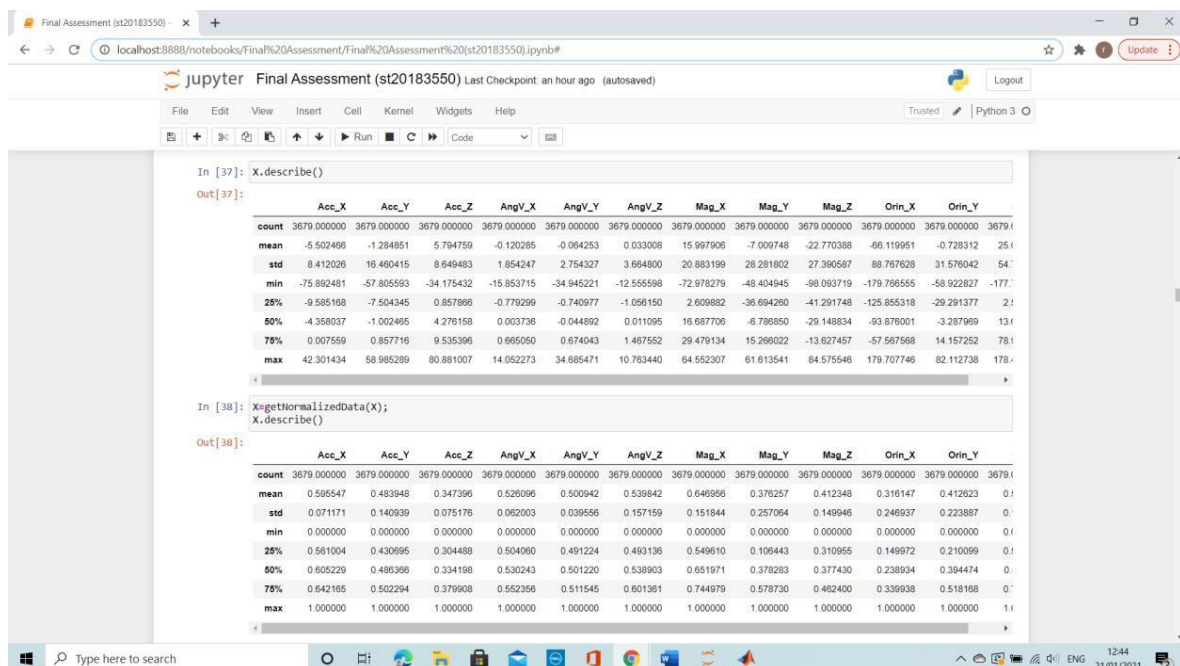
Appendix A3. Data Cleaning



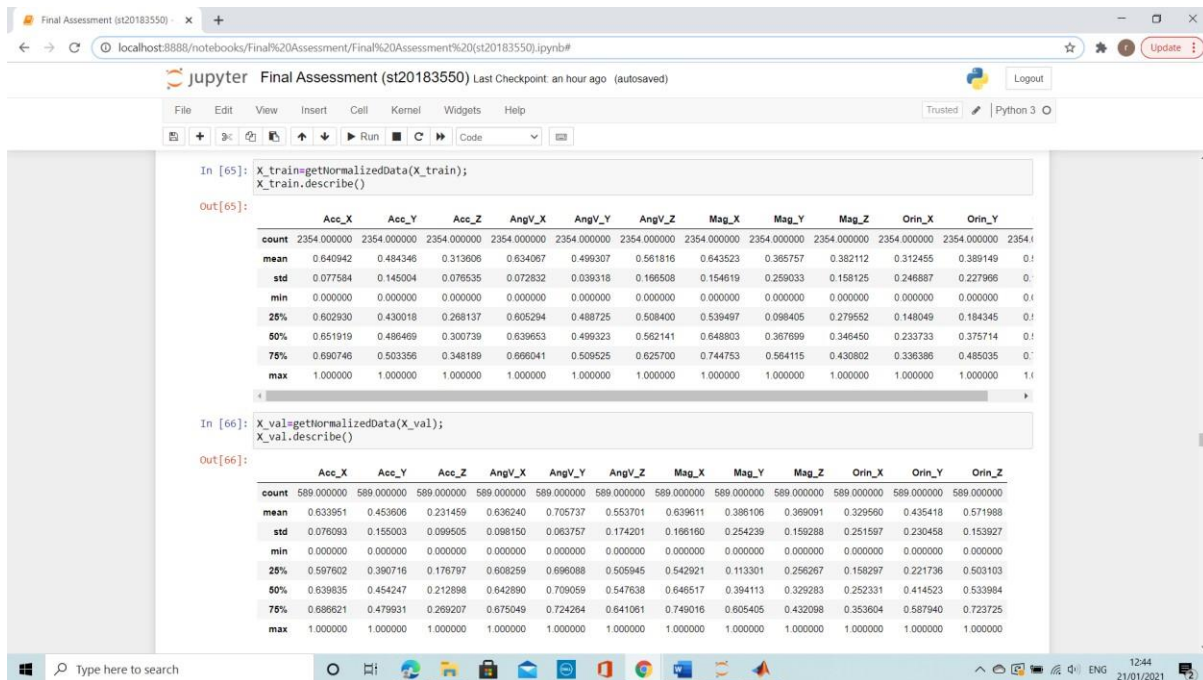
Appendix A4. Feature Selection



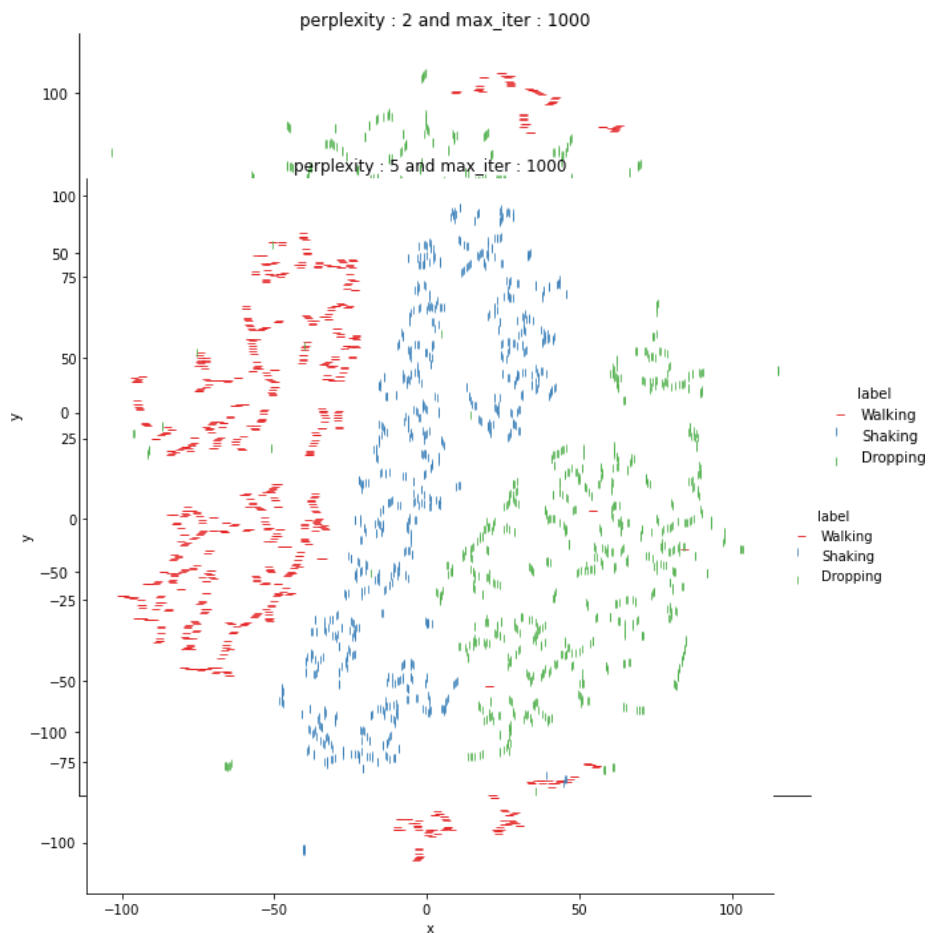
Splitting

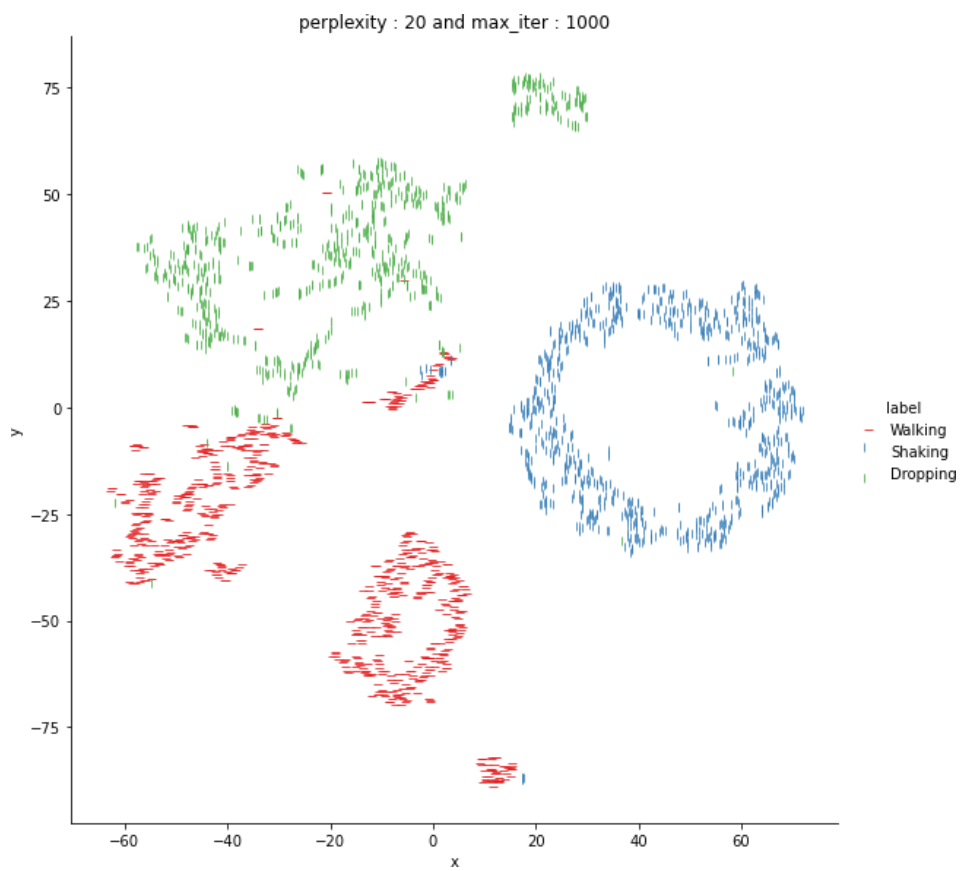
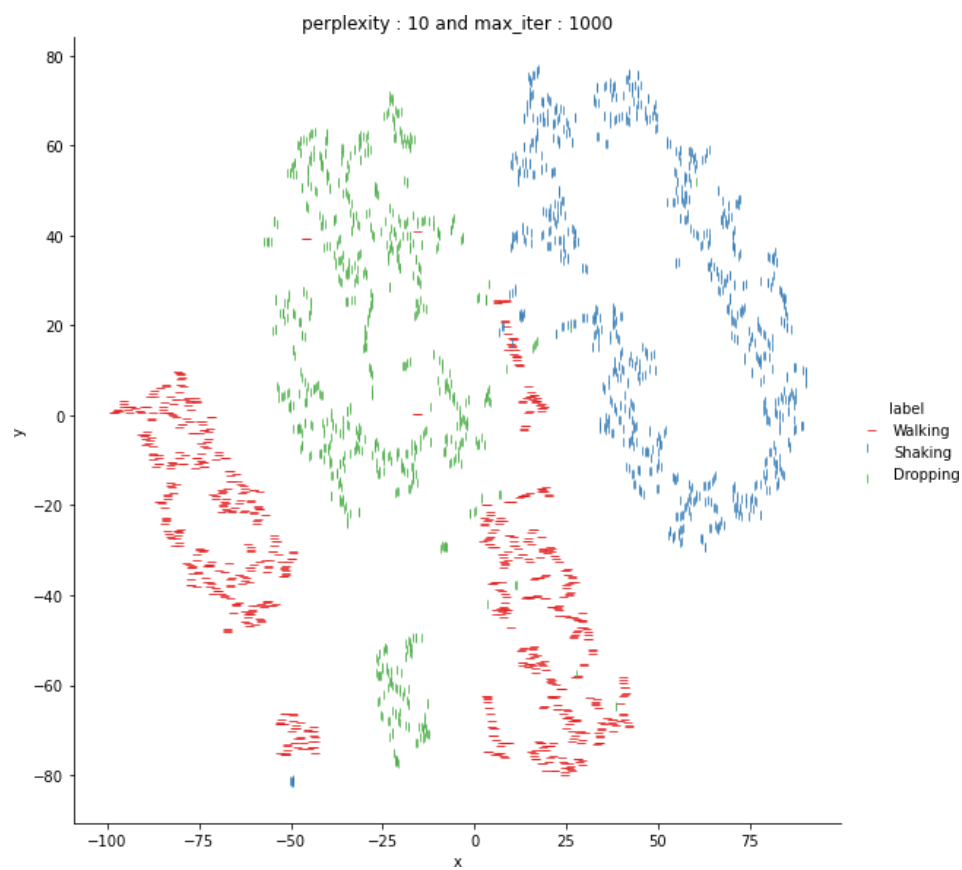


Appendix A7. Data Normalization (X_test and X_val)



Appendix A8. T-SNE visualisation





Appendix A9. Package used in Machine Learning Algorithms

```
Final Assessment (st20183550) - x +
localhost:8888/notebooks/Final%20Assessment/Final%20Assessment%20(st20183550).ipynb#
jupyter Final Assessment (st20183550) Last Checkpoint: an hour ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [70]: from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

In [71]: classifiers = [
    DecisionTreeClassifier(),
    KNeighborsClassifier(4),
    SVC(decision_function_shape='ovr'),
    LogisticRegression(solver='lbfgs', multi_class='multinomial'),
    RandomForestClassifier(n_estimators=10)
]

names = []
scores = []

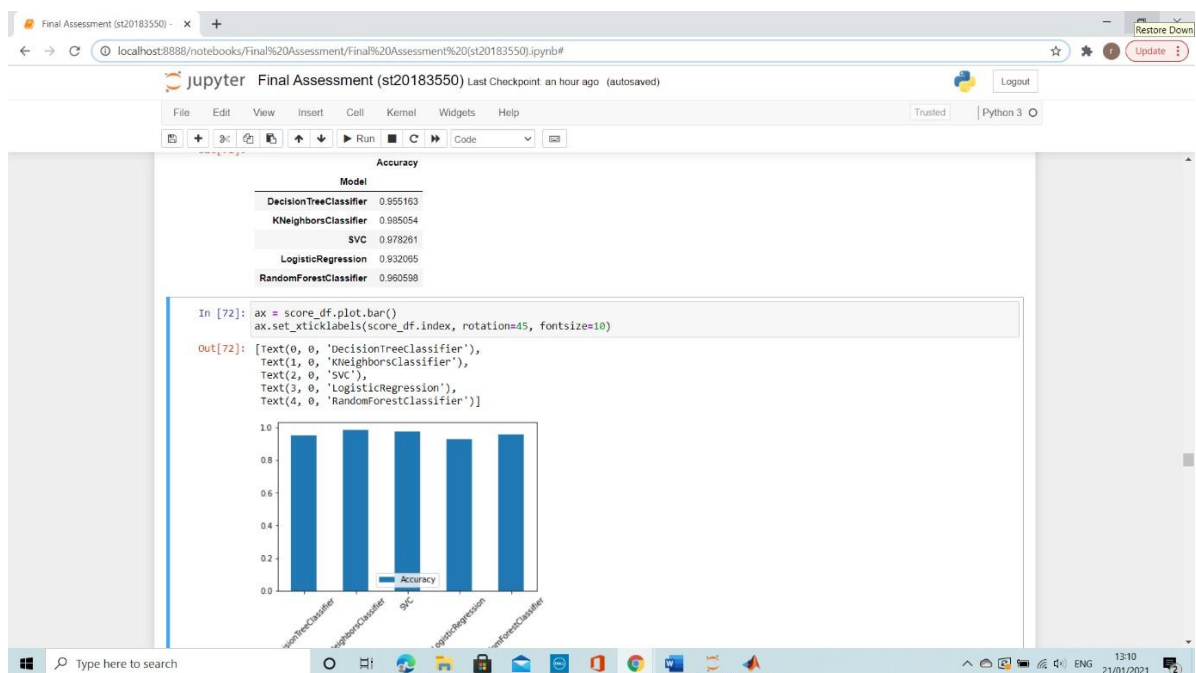
for clf in classifiers:
    clf = clf.fit(X_train, y_train)
    ypred = clf.predict(X_test)
    names.append(clf.__class__.__name__)
    scores.append(accuracy_score(ypred, y_test))

score_df = pd.DataFrame({'Model': names, 'Accuracy': scores}).set_index('Model')
score_df

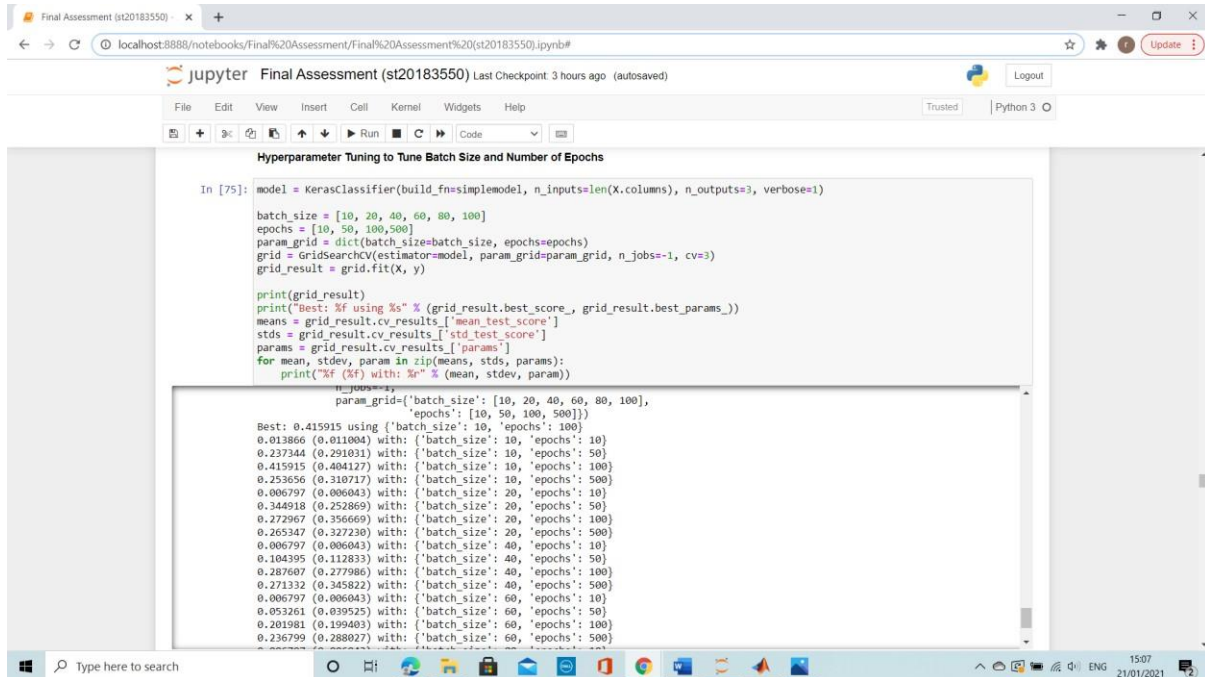
Out [71]:
```

Model	Accuracy
DecisionTreeClassifier	0.955163
KNeighborsClassifier	0.985054
SVC	0.978261
LogisticRegression	0.932065
RandomForestClassifier	0.960598

Appendix A10. Accuracy of Machine Learning Algorithms



Appendix A11. Hyperparameter Tuning for Batch Size and Epochs (Model 1)



```
Final Assessment (st20183550) - x +
localhost:8888/notebooks/Final%20Assessment/Final%20Assessment%20(st20183550).ipynb#
jupyter Final Assessment (st20183550) Last Checkpoint 3 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Type here to search

Hyperparameter Tuning to Tune Batch Size and Number of Epochs

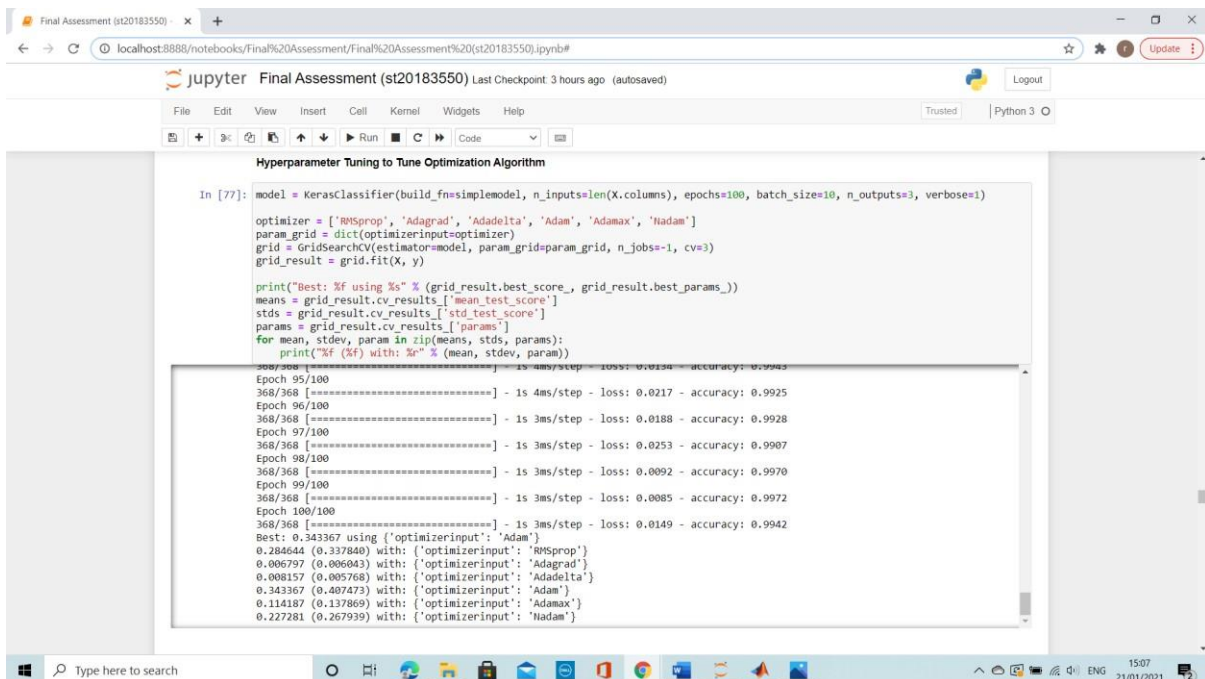
In [75]: model = KerasClassifier(build_fn=simplemodel, n_inputs=len(X.columns), n_outputs=3, verbose=1)

batch_size = [10, 20, 40, 60, 80, 100]
epochs = [10, 50, 100, 500]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X, y)

print(grid_result)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %s" % (mean, stdev, param))

Best: 0.415915 using {'batch_size': 10, 'epochs': 100}
0.013866 (0.011004) with: {'batch_size': 10, 'epochs': 10}
0.237344 (0.291031) with: {'batch_size': 10, 'epochs': 50}
0.415915 (0.404127) with: {'batch_size': 10, 'epochs': 100}
0.253656 (0.310717) with: {'batch_size': 10, 'epochs': 500}
0.006797 (0.006043) with: {'batch_size': 20, 'epochs': 10}
0.344918 (0.252869) with: {'batch_size': 20, 'epochs': 50}
0.272967 (0.356669) with: {'batch_size': 20, 'epochs': 100}
0.265347 (0.327230) with: {'batch_size': 20, 'epochs': 500}
0.006797 (0.006043) with: {'batch_size': 40, 'epochs': 10}
0.104395 (0.112833) with: {'batch_size': 40, 'epochs': 50}
0.287607 (0.277986) with: {'batch_size': 40, 'epochs': 100}
0.271332 (0.345822) with: {'batch_size': 40, 'epochs': 500}
0.006797 (0.006043) with: {'batch_size': 60, 'epochs': 10}
0.053261 (0.039525) with: {'batch_size': 60, 'epochs': 50}
0.201981 (0.199403) with: {'batch_size': 60, 'epochs': 100}
0.236799 (0.288027) with: {'batch_size': 60, 'epochs': 500}
```

Appendix A12. Hyperparameter Tuning for Optimization Algorithm



```
Final Assessment (st20183550) - x +
localhost:8888/notebooks/Final%20Assessment/Final%20Assessment%20(st20183550).ipynb#
jupyter Final Assessment (st20183550) Last Checkpoint 3 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Type here to search

Hyperparameter Tuning to Tune Optimization Algorithm

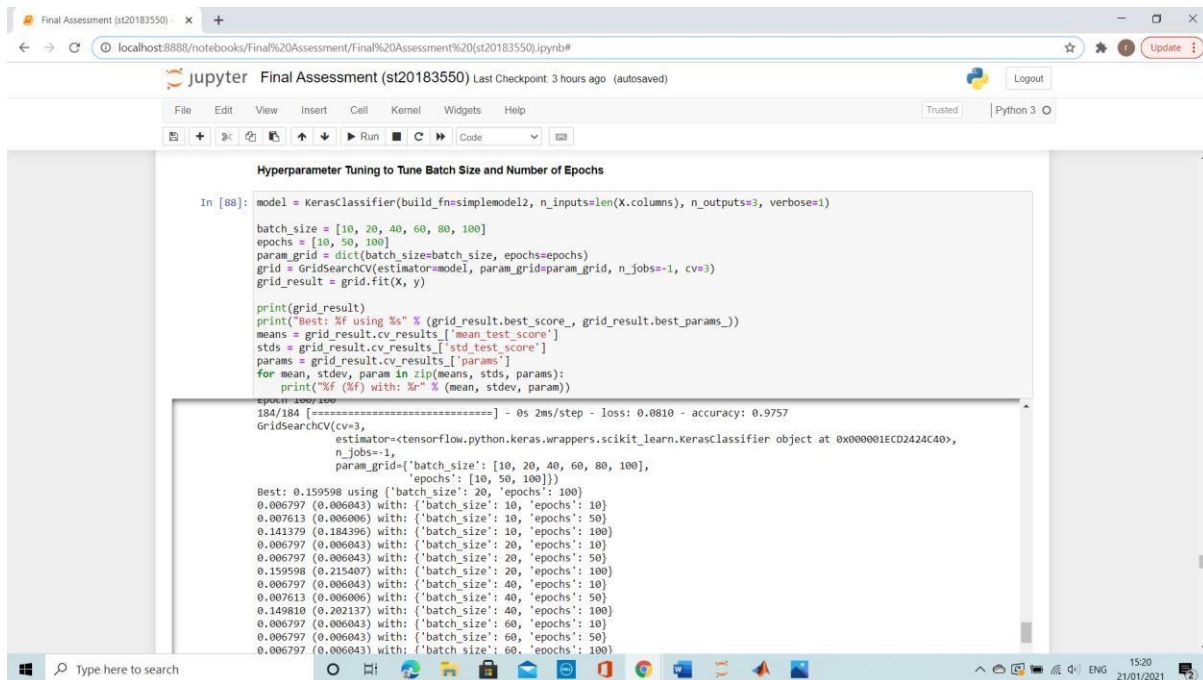
In [77]: model = KerasClassifier(build_fn=simplemodel, n_inputs=len(X.columns), epochs=100, batch_size=10, n_outputs=3, verbose=1)

optimizer = ['RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
param_grid = dict(optimizer=optimizer)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X, y)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %s" % (mean, stdev, param))

Epoch 95/100
368/368 [=====] - 1s 4ms/step - loss: 0.0217 - accuracy: 0.9925
Epoch 96/100
368/368 [=====] - 1s 3ms/step - loss: 0.0188 - accuracy: 0.9928
Epoch 97/100
368/368 [=====] - 1s 3ms/step - loss: 0.0253 - accuracy: 0.9907
Epoch 98/100
368/368 [=====] - 1s 3ms/step - loss: 0.0092 - accuracy: 0.9970
Epoch 99/100
368/368 [=====] - 1s 3ms/step - loss: 0.0085 - accuracy: 0.9972
Epoch 100/100
368/368 [=====] - 1s 3ms/step - loss: 0.0149 - accuracy: 0.9942
Best: 0.343367 using {'optimizerinput': 'Adam'}
0.284644 (0.337840) with: {'optimizerinput': 'RMSprop'}
0.006797 (0.006043) with: {'optimizerinput': 'Adagrad'}
0.008157 (0.005768) with: {'optimizerinput': 'Adadelta'}
0.343367 (0.407473) with: {'optimizerinput': 'Adam'}
0.114187 (0.137869) with: {'optimizerinput': 'Adamax'}
0.227281 (0.267939) with: {'optimizerinput': 'Nadam'}
```

Appendix A13. Hyperparameter Tuning for Batch Size and Epochs (Model 2)



```
Hyperparameter Tuning to Tune Batch Size and Number of Epochs

In [88]: model = KerasClassifier(build_fn=simpleModel2, n_inputs=len(X.columns), n_outputs=3, verbose=1)

batch_size = [10, 20, 40, 60, 80, 100]
epochs = [10, 50, 100]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X, y)

print(grid_result)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %s" % (mean, stdev, param))

epoch 100/200
184/184 [=====] - 0s 2ms/step - loss: 0.0810 - accuracy: 0.9757
GridSearchCV(cv=3,
             estimator=<tensorflow.python.keras.wrappers.scikit_learn.KerasClassifier object at 0x000001ECD2424C40>,
             n_jobs=-1,
             param_grid={'batch_size': [10, 20, 40, 60, 80, 100],
                         'epochs': [10, 50, 100]})
Best: 0.159598 using {'batch_size': 20, 'epochs': 100}
0.006797 (0.006043) with: {'batch_size': 10, 'epochs': 10}
0.007613 (0.006006) with: {'batch_size': 10, 'epochs': 50}
0.141379 (0.184396) with: {'batch_size': 10, 'epochs': 100}
0.006797 (0.006043) with: {'batch_size': 20, 'epochs': 10}
0.006797 (0.006043) with: {'batch_size': 20, 'epochs': 50}
0.159598 (0.215407) with: {'batch_size': 20, 'epochs': 100}
0.006797 (0.006043) with: {'batch_size': 40, 'epochs': 10}
0.007613 (0.006006) with: {'batch_size': 40, 'epochs': 50}
0.149810 (0.202137) with: {'batch_size': 40, 'epochs': 100}
0.006797 (0.006043) with: {'batch_size': 60, 'epochs': 10}
0.006797 (0.006043) with: {'batch_size': 60, 'epochs': 50}
0.006797 (0.006043) with: {'batch_size': 60, 'epochs': 100}
```

Appendix A14. Youtube Video Link - <https://youtu.be/PaaLSQM6PNI>