Machine Learning Engineer Nanodegree

Capstone Project: Human Activity Monitoring System

Anshul Bansal

20/01/2020

I. Definition

Project Overview

With the advent and now ubiquitous presence of smart devices such as cellular phones and fitness watches around us, an exciting new area of data mining research and data mining applications have been opened up. These devices have powerful sensors like GPS sensors, audio sensors, direction and acceleration sensors like accelerometers and gyrometer, which are capable of reading and processing the data quite accurately and in real time.

I have always wondered how the smartphones and fitness watches know when I am running, walking, cycling or just have been lethargic for a while . The project is aimed to demystify this and build a system which can accurately detect the activity using some of the sensor data from the accelerometer and gyrometer in particular.

A lot of previous work has been done in this regard like Human Activity Recognition using LSTMs on Android — TensorFlow for Hackers (Part VI) and this Activity Recognition using Cell Phone Accelerometers, to quote a few.

This has a large number of applications. There is a possibility of providing a highly personalized and customized experience of smart devices like phones and watches. Also it can be determined if the user is following a healthy exercise routine or not. Let's see how accurately our model is able to predict the activity based on the sensor data.Excited to explore this!

Datasetlink - https://archive.ics.uci.edu/ml/machine-learning-databases/00507/

Problem Statement

The task is bucket the user activity in one of the 3 groups (Non-hand-oriented activities- running, jogging, Hand-oriented activities - General like clapping, folding clothes, Hand-oriented activities - Eating like eating food, drinking) which consists of the 18 given activities based on sensor data. In the machine learning domain this is a multi-class classification and a supervised learning problem. The idea is to build a system (model) which takes inputs from sensors like accelerometers and gyrometer present in watches and mobile phones and accurately predict the user activity group.

Evaluation Metrics

Classification model will be evaluated on **Accuracy**. Accuracy is an important factor while detecting the human activity. The system is expected to predict the activity with as high accuracy as possible so that appropriate action can be taken in response by other system dependent on it.

Accuracy measures how often the classifier makes the correct prediction. It is the ratio of the number of correct predictions to the total number of predictions (the number of test data points).

In the case of multiclass classification problem, the accuracy can be calculated from the confusion matrix using the formula below.

Accuracy = sum(diag(matrix)) / sum(matrix))

II. Analysis

Dataset Features and Description

The raw accelerometer and gyroscope sensor data is collected from the smartphone and smartwatch at a rate of 20Hz. It is collected from 51 test subjects as they perform 18 activities for 3 minutes apiece. The sensor data is represented using a 10-second window.

Attribute Information:
subject-id: value from 1600- 1650 that identifies one of the 51 test subjects
activity-code: character between 'A' and 'S' (no 'N') that identifies the activity. The mapping from code to activity is provided in the activity_key.txt file and in our dataset description document.
timestamp: Unix time (integer)
x: represents the sensor reading (accelerometer or gyroscope) for the x dimension
y: represents the sensor reading (accelerometer or gyroscope) for the y dimension
z: represents the sensor reading (accelerometer or gyroscope) for the z dimension

Further the transformed dataset we would be using has following labels.
1. ACTIVITY: specifies the activity performed using one of the activity codes from X{0-9}; Y{0-9}, Z{0-9}: These 30 features collectively show the distribution of values over the
2. x, y, and z axes. We call this a binned distribution. For each axis we determine the range of values in the 10s window (max – min value), divide this range into 10 equal-sized bins, and then record the fraction of values in each bin.
3. {X,Y,Z}AVG: Average sensor value over the window (per axis).

4. {X,Y,Z}PEAK: Time in milliseconds between the peaks in the wave associated with most activities. Heuristically determined (per axis).
5. {X,Y,Z}ABSOLDEV: Average absolute difference between each of the 200 readings and the mean of those values (per axis)
6. {X,Y,Z}STANDDEV: Standard deviation of the 200 values (per axis)
7. {X,Y,Z}VAR: Variance of the values (per axis)
8. XMFCC{0-12}, YMFCC{0-12}, ZMFCC{0-12}: MFCCs represent short-term power spectrum of a wave, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. There are 13 values per axis.
9. {XY, XZ, YZ}COS: The cosine distances between sensor values for pairs of axes (three pairs of axes).
10. {XY, XZ, YZ}COR: The correlation between sensor values for pairs of axes (three pairs of axes).
11. RESULTANT: Average resultant value, computed by squaring each matching x, y, and z value, summing them, taking the square root, and then averaging these values over the 200 readings.
12. Class: This is a very different feature than the rest. It simply is set to the subject-id.

For more information this file can be referenced - Transaction / Regular Paper Title.

Data Exploration

The files are spread across phone and watch folder and further segregated as gyro and accelerometer folder. As a first step, I had to assemble all the files present in these folders in a single data frame.

Based on preliminary data analysis following were the observed meta information.

- Total number of records: 75,099
- Total number of features: 93
- Total number of categories: 18
- Number of Blank rows: None
- Number of Phone readings:
- Number of Watch readings:
- Number of gyrometer readings:
- Number of accelerometer readings:
- Type of data: Float and String and mix of positive and negative values.

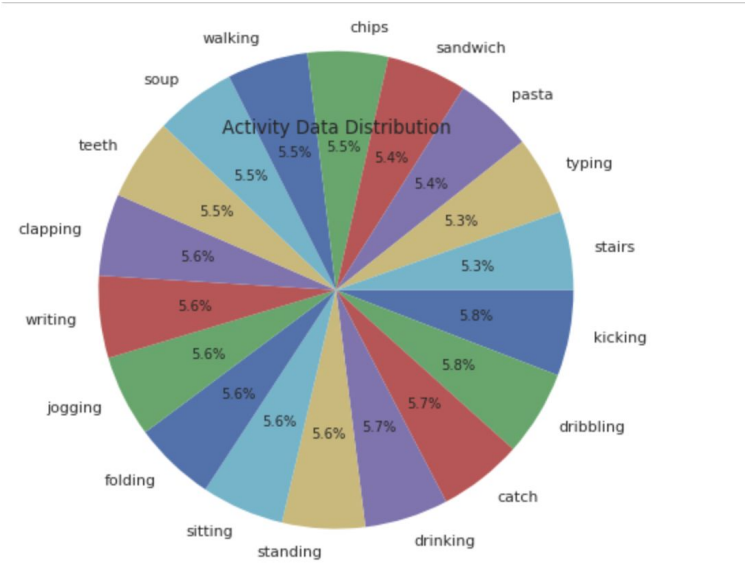
	"ACTIVITY"	"X0"	"X1"	"X2"	"X3"	"X4"	"X5"	"X6"	"X7"	"X8"	...	"ZMFCC11"	"ZMFCC12"	"XYCOS"	"XZCOS"	"YZCOS"	"XYCOR"	"XZCOR"	"
0	b'A'	0.170	0.315	0.435	0.06	0.010	0.0	0.01	0.0	0.0	...	0.433341	0.427773	-0.188714	0.141920	-0.062945	-0.204853	0.140685	-
1	b'A'	0.125	0.310	0.485	0.08	0.000	0.0	0.00	0.0	0.0	...	0.386114	0.381152	-0.371368	0.198086	0.442477	-0.372734	0.199274	
2	b'A'	0.150	0.320	0.485	0.04	0.005	0.0	0.00	0.0	0.0	...	0.405939	0.400722	-0.522678	-0.038708	0.519698	-0.517238	-0.038878	
3	b'A'	0.190	0.220	0.430	0.16	0.000	0.0	0.00	0.0	0.0	...	0.426505	0.421024	-0.642675	0.057995	0.128622	-0.643005	0.057987	
4	b'A'	0.205	0.240	0.370	0.17	0.015	0.0	0.00	0.0	0.0	...	0.469978	0.463939	-0.603190	-0.031606	0.137222	-0.603988	-0.031405	

Data Cleaning

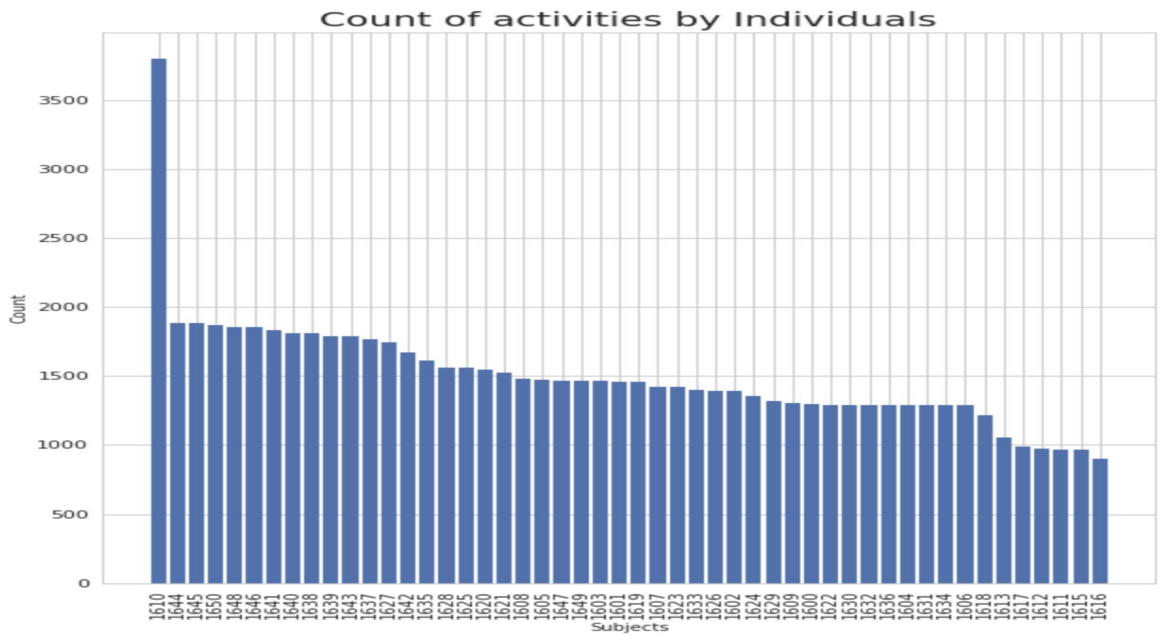
The *ACTIVITY* column and *class* column needs to be formatted to remove unnecessary quotes and variable(b) appended to it. Also the *ACTIViTY* column has the activity code associated with , we create a new column *activity_name* to better visualize the dataset.
The mapping is read from the **activity_key.txt** file.
The mapping is

activty_map = { 'A': 'walking', 'B': 'jogging', 'C': 'stairs', 'D': 'sitting','E': 'standing','F': 'typing', 'G': 'teeth','H': 'soup', 'I': 'chips','J': 'pasta', 'K': 'drinking','L': 'sandwich','M': 'kicking','O': 'catch', 'P': 'dribbling', 'Q': 'writing','R': 'clapping', 'S': 'folding' }

Data Visualisation



The spread of data points across different categories is evenly spread out. We have a balanced dataset, which is not so common.



The individual with subject id as 1610 seems to be very active during the data collection phase while individual with subject id 1616 seems to have not been paid enough to participate.

Algorithms and Techniques

Choosing the right model for any data science task is a tricky job. Based on the previous works and experience we can make an educated move in deciding the correct technique for getting insights from the dataset and thus make decisions. The features in the dataset is numerical data while the label is categorical variable.

Following algorithms were explored for dealing with this classification problem.

1. Linear Learner
2. Linear SVC
3. KNeighborsClassifier

Amazon sagemaker was used for building,training, deployment and inference of the results. Scikit-learn library was used for implementing the different other classifiers. Amazon sagemaker provides out of the box support for the same.

Linear learner works on the principle of one vs all principle. This strategy consists of fitting one classifier per class and is thus computational feasible. Linear learner uses a softmax loss function to train multiclass classifiers. The algorithm learns a set of weights for each class, and predicts a probability for each class. The algorithm is relatively fast and easy to implement. Earlier it was used for binary classification but now sagemaker has hyperparameters which can specified to make it be used for multiclass classifier as well. Thus it became a natural choice to be picked up for creating base model.

As per the past experience and researches done, Support Vector Machines (SVM) are known to perform good in terms of multiclass classification. Linear SVC and KNeighborsClassifier were further explored to improve the accuracy over the base model.

Linear SVC again works on one vs all principle. It is similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and scales better to large numbers of samples .(~75000 in our case) as compared to SVC.

KNeighborsClassifier seemed to be an excellent choice in terms of classification. KNeighborsClassifier is a non-parametric and lazy learning algorithm. What it essentially means is that it doesn't make any assumption on the dataset distribution, which closely mimics the real world data. Also it does not need any training data points for model generation. All training data used in the testing phase. We play with the hyperparameter **n_neighbors** during the tuning phase to achieve good accuracy.

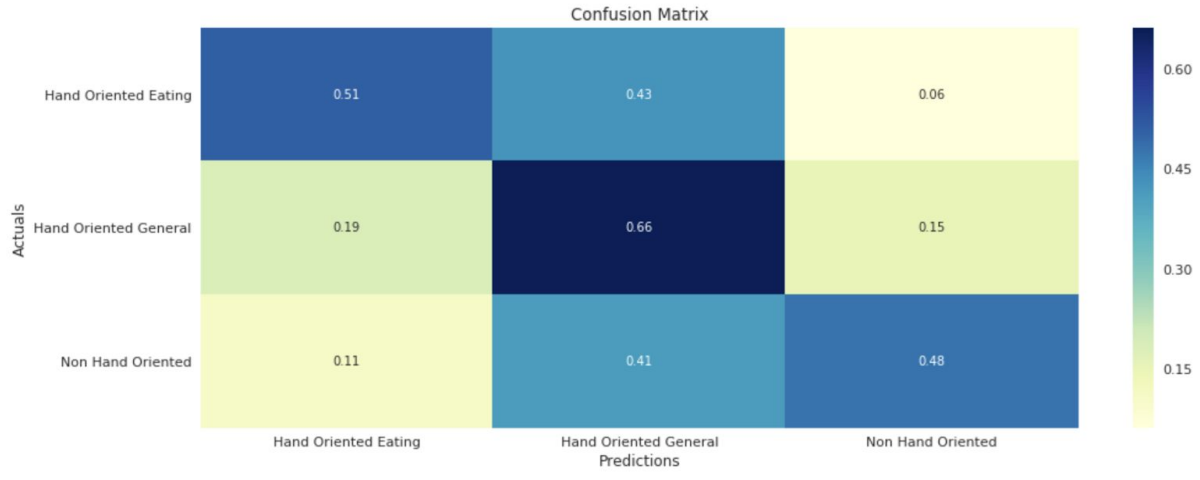
Benchmark

For this problem, the benchmark model was **Linear Learner model.** It was trained with training data and with setting the hyperparameter of **predictor_type** as 'multiclass_classifier'. Further algorithms were explored then to have a better accuracy than this model.

The sagemaker estimator was

```
multiclass_estimator = sagemaker.LinearLearner(role=sagemaker.get_execution_role(),
                                               train_instance_count=1,
                                               train_instance_type='ml.m4.xlarge',
                                               predictor_type='multiclass_classifier',
                                               output_path=output_path,
                                               num_classes=3)
```

```
Accuracy: 0.560
```



The benchmark accuracy score was **~56%**

III. Methodology

Data Preprocessing

This involved some column names cleaning in terms of removing unnecessary characters like “”, ‘’, b from the column values.

There are 18 distinct classes or number of activities in which the data points have been categorised into. For this project these classes were further grouped into the **3 Classes** as following.

0: (Non-hand-oriented activities) {walking, jogging, stairs, standing, kicking}
1: (Hand-oriented activities- General): {dribbling, playing catch, typing, writing, clapping, brushing teeth, folding clothes}
2: (Hand-oriented activities -eating): {eating pasta, eating soup, eating sandwich, eating chips, drinking}

Target Variable is **ACTIVITY**, whose value can be 0, 1 or 2. which is further mapped to the respective group.

```
def activty_mapper(activity):
    if(activity in ['A', 'B', 'C', 'D', 'E', 'M']):
        return 0
    elif(activity in ['P', 'O', 'F', 'Q', 'R','G', 'S']):
        return 1
    else:
        return 2

df['ACTIVITY'] = df['ACTIVITY'].apply(activty_mapper)
```

Data Normalisation

During the exploration phase we discovered the range of column values or features is large. The values ranged from negative numbers to large positive numbers. For good models, It was imperative to scale the features such that the values are between 0 and 1.

MinMaxScaler from scikit-learn was used to scale and translate each feature individually such that it is in the given range on the training set, e.g. between zero and one.

Feature Selection

Since in this case there are too many features - 91 in total, feature selection was required.It enables to select those features which contribute most to the prediction category.

Univariate Selection was used to select the 10 most important features. It works by selecting the best features based on univariate statistical tests. It can be seen as a preprocessing step to an estimator. SelectKBest algorithm removes all but the 10 highest scoring features. Following were the top 10 important features based on score.

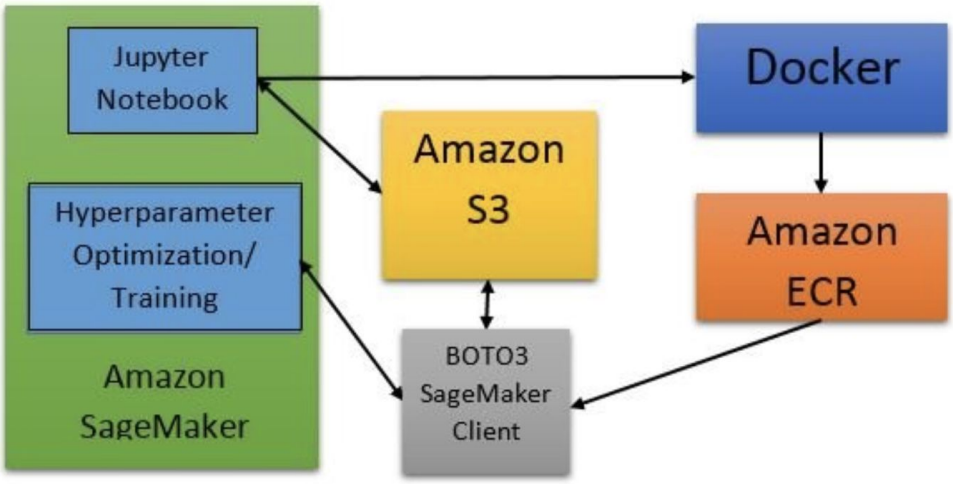
chi-squared stats score can be used to select the n_features features with the highest values for the test chi-squared statistic from feature list, which must contain only non-negative features such as booleans or frequencies (e.g., term counts in document classification), relative to the classes.
The test measures dependence between stochastic variables, so using this function “weeds out” the features that are the most likely to be independent of class and therefore irrelevant for classification.

Following was the output of feature selection.

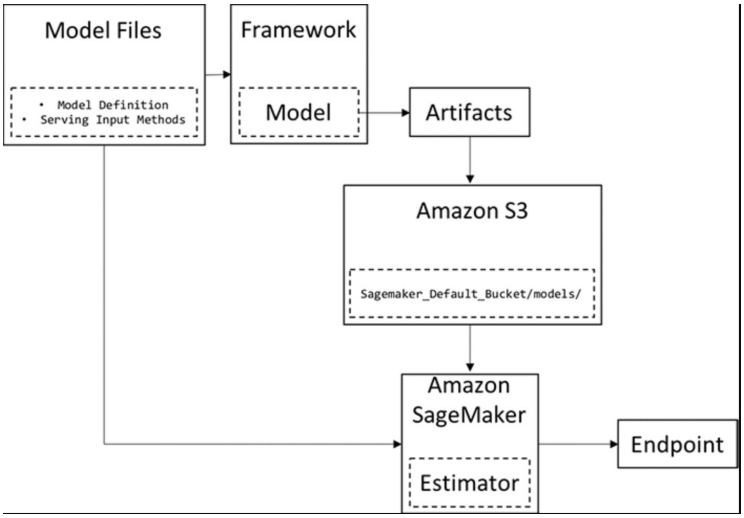
```
        Specs        Score
42       XVAR  1208.759662
43       YVAR  1139.774675
39  XSTANDDEV  1083.781560
36  XABSOLDEV  1070.791103
7           X7   970.931527
40  YSTANDDEV   906.095907
37  YABSOLDEV   902.136698
44       ZVAR   881.935651
6           X6   760.882796
24          Z4   713.046505
selected_features ['XVAR', 'YVAR', 'XSTANDDEV', 'XABSOLDEV', 'X7', 'YSTANDDEV', 'YABSOLDEV', 'ZVAR', 'X6', 'Z4']
```

Implementation

The workflow was implemented in jupyter notebook and amazon sagemaker. The workflow is as below.



The implementation can be broken down into following steps:



Base Model Implementation

First we implement the **base model** that is implement the multiclass using **Amazon SageMaker linear learner**

- **Split the training, validation and test data**
  Train_test_split utility from sckit-learn library was used to split the data into 80% training data, 10%validation data and 10% test data.

```
xTrain, xTest, yTrain, yTest = train_test_split(features, labels, test_size = 0.2, random_state = 0)
xVal, xTest, yVal, yTest = train_test_split(xTest, yTest, test_size = 0.5, random_state = 0)
```

- **Creating the sagemaker estimator for linear learner**

```
multiclass_estimator = sagemaker.LinearLearner(role=sagemaker.get_execution_role(),
                                               train_instance_count=1,
                                               train_instance_type='ml.m4.xlarge',
                                               predictor_type='multiclass_classifier',
                                               output_path=output_path,
                                               num_classes=3)
```

  We provide the execution role , instance types using which the model will be trained, specify predictor_type as multiclass_classifier to tell the algorithm it is the case of multiclass classification, and num_classes as 3 which is the number of classes/labels.

- **Fit the model(Train) and deploy**
  We create the record sets for training, validation and test and train the model and finally deploy it using the sagemaker

```
In [ ]: # create RecordSet
        train_records = multiclass_estimator.record_set(xTrain, yTrain, channel='train')
        val_records = multiclass_estimator.record_set(xVal, yVal, channel='validation')
        test_records = multiclass_estimator.record_set(xTest, yTest, channel='test')

In [ ]: multiclass_estimator.fit([train_records, val_records, test_records])

In [ ]: multiclass_predictor = multiclass_estimator.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge')
```

- **Check for accuracy**
  The accuracy of the model was **56%.**

SKLearn Model Implementation (Linear SVC AND KNeighborsClassifier)

Next we implement the improvement over the current base model using the scikit-learn library and linear svc and KNeighborsClassifier provided by the library. The process is

- Saving the training and testing data in local library using the **create_csv function.**
- Upload the data to S3 bucket as sagemaker estimator reads data from the s3 bucket.
- Creating a sagemaker estimator specifying the entry point of the training script which would be different in either of those algorithms.
- The training script (**train.py**) has the ***model_fn*** for loading the trained model and ***__name__ == '__main__'*** block where we use the hyperparameters for model training.

```
from sagemaker.sklearn.estimator import SKLearn

output_path = 's3://{}/{}'.format(bucket, prefix)

sklearn_estimator = SKLearn(entry_point='train.py',
                            role=role,
                            train_instance_type='ml.m4.4xlarge',
                            train_instance_count=1,
                            output_path=output_path,
                            framework_version='0.20.0',
                            sagemaker_session=sagemaker_session,
                            hyperparameters = { 'kernel': 'linear' })
```

Sagemaker Estimator

- The training data is used to train the model and deployed to the sagemaker endpoint which is further used to make prediction on test data.

Refinement

The base model linear learner was based on vs rest strategy and predicted the class with an accuracy of 56%. We then moved on to use Linear SVC to check if the earlier model could be refined or improved but didn't get an improvement. The accuracy was 46%.

Linear SVC score was not good compared to base model. I then evaluated **KNeighborsClassifier.**

In case of **KNeighborsClassifier** instead of using the default hyperparameters, we tweak the hyperparameters like **n_neighbors** to improve the model performance.

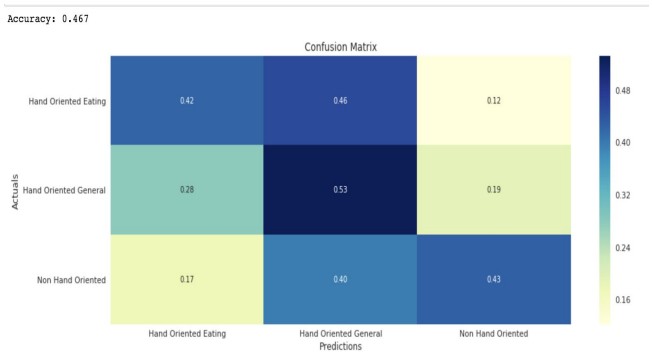
There was an improvement which could be seen as **n_neighbors** was changed from 3 to 5 to 7

IV. Results

The trained models were tested on the test data stored in the test directory. Accuracy score for each algorithm was found and confusion matrix was made depicting the results.

Algorithm	**Accuracy Score**
Linear Learner	56%
LinearSVC	46%
KNeighborsClassifier (n_neighbours:3)	59.8%
KNeighborsClassifier (n_neighbours:5)	61%
KNeighborsClassifier (n_neighbours:13)	62.2%

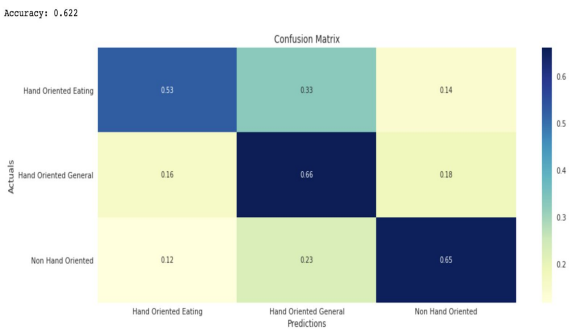
Confusion Matrix and Algorithm Accuracy Score



Linear SVC

KNeighborsClassifier (n_neighbours:3)

KNeighborsClassifier(n_neighbours:5)



KNeighborsClassifier (n_neighbours:13)

Notes:

- The final model made using the KNeighborsClassifier with n_neighbours =13 performs better than our benchmark model in terms of accuracy. We will treat this as our final model.
- This model has a 10% improvement over the base model in terms of accuracy.

V. Conclusion

Reflection

The idea of tracking user activity based on the sensor data is an interesting domain as we have our cell phones and watches with us every time giving us access to an abundance of data. It is just about making the right inferences out of it.

The process started with collecting and refinement of the sensor data. The data was preprocess to make it in a normalized form and also feature selection was done to retain most relevant features.

The model was trained using the sagemaker estimator and deployed to the endpoint. The endpoint was further used to make predictions on the test data on which we finally validated our model in terms of accuracy.

Amazon Sagemaker is a handy tool especially for someone who is just starting off in ML. Using sagemaker we get access to powerful resources, dashboards, insights etc. which makes the process of training, deployment and prediction a breeze.

There were some difficult aspects of the projects in terms of identifying the correct features. Also there was a problem of overlapping classes in the 2D space while using tsne , so the prediction might not be highly accurate.

The final model KNeighborsClassifier(n_neighbours=13) has accuracy as 62.2% which is good enough to be used in device as we collect more data and make further refinements. For future enhancement this model can be considered as the base model.

Improvements

- I see further improvements in the current setup by using neural networks for modelling.
- Sagemakere Hypertuning jobs can be used to further fine tune the model.
- The transformed data was used for this project, we can use the time series data or raw data to build an even better model.

VI. References

- https://archive.ics.uci.edu/ml/index.php
- http://www.cis.fordham.edu/wisdm/includes/files/sensorKDD-2010.pdf
- https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-p
- Ython-f24e7da3f36e
- Smartphone and Smartwatch-Based Biometrics Using Activities of Daily Living. IEEE
- Access, 7:133190-133202, Sept. 2019.