Project Report

Human Activity Recognition from Smart Phone Data

NAME: HARI KRISHNAN

COURSE: AI and ML

Question:

Perform activity recognition on the dataset using a hidden markov model. Then perform the same task using a different classification algorithm (logistic regression/decision tree) of your choice and compare the performance of the two algorithms

Prerequisites

What things you need to install the software and how to install them:

Python 3.6 This setup requires that your machine has latest version of python. The following url https://www.python.org/downloads/ can be referred to download python. Once you have python downloaded and installed, you will need to setup PATH variables (if you want to run python program directly, detail instructions are below in how to run software section). To do that check this: https://www.pythoncentral.io/add-python-to-path-python-is-not-recognized-asan-internal-or-externalcommand/ . Setting up PATH variable is optional as you can also run program without it and more instruction are given below on this topic.

Second and easier option is to download anaconda and use its anaconda prompt to run the commands. To install anaconda check this url https://www.anaconda.com/download/ You will also need to download and install below 3 packages after you install either python or anaconda from the steps above Sklearn (scikit-learn) numpy scipy if you have chosen to install python 3.6

Dataset Link: Human Activity Recognition with Smartphones

https://www.kaggle.com/uciml/human-activity-recognition-with-smartphones

Implementation

Importing the libraries and dataset

```
[47]: import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
      import seaborn as sns
      import warnings
      warnings.filterwarnings('ignore')
```

Read the data set

```
[48]: train = pd.read_csv("train.csv")
      test = pd.read_csv('test.csv')

[49]: train.head()
```

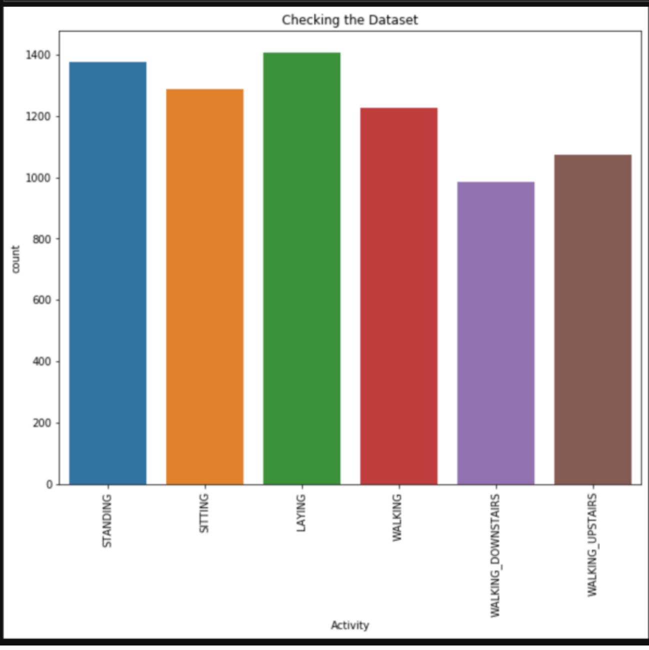
[49]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJerkMag-kurtosis()	angle(tBodyAccMean,gravity)	angle(tBodyAccJerkMean),gravityMean)	angle(tBodyGyroMean,gravityMean)	angle(tBodyGyroJerkMean,gravityMean)	angle(X,gravityM
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.710304	-0.112754	0.030400	-0.464761	-0.018446	-0.84
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	-0.861499	0.053477	-0.007435	-0.732626	0.703511	-0.84
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	-0.760104	-0.118559	0.177899	0.100699	0.808529	-0.84
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	-0.482845	-0.036788	-0.012892	0.640011	-0.485366	-0.84
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	-0.699205	0.123320	0.122542	0.693578	-0.615971	-0.84

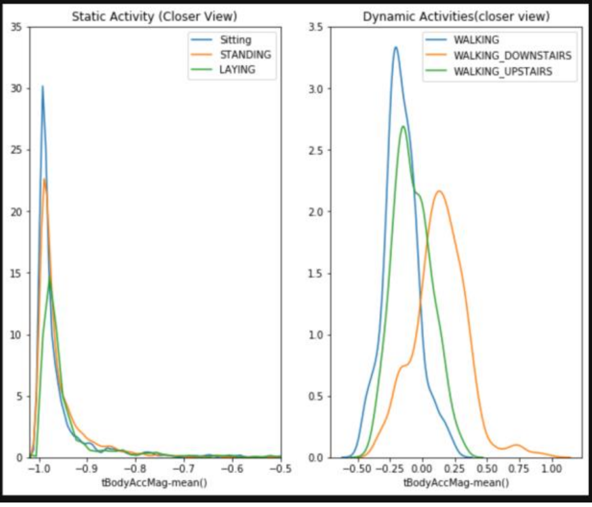
5 rows × 563 columns

Visualization of data:

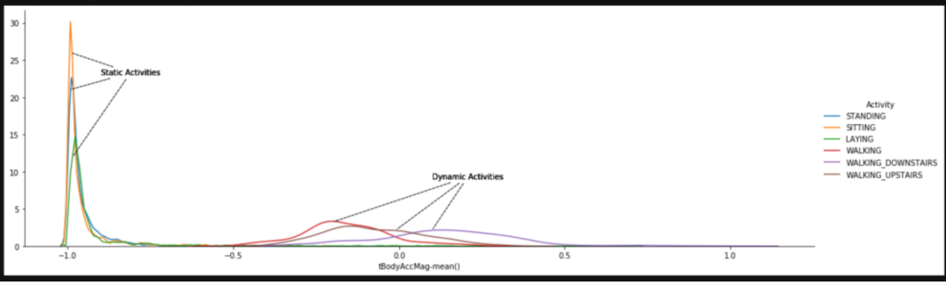
```
[50]: print('Number of duplicates in train : ',sum(train.duplicated()))
      print('Number of duplicates in test : ', sum(test.duplicated()))

      Number of duplicates in train :  0
      Number of duplicates in test :  0

[51]: print('Total number of null values in train:',train.isna().values.sum())
      print('Total number of null values in test:',test.isna().values.sum())

      Total number of null values in train: 0
      Total number of null values in test: 0

[52]: # Checking wether the classs are imbalanced or not
      plt.figure(figsize =(10,8))
      sns.countplot(train['Activity'])
      plt.title('Checking the Dataset')
      plt.xticks(rotation = 90)
      plt.show()
```



```
[54]: plt.figure(figsize=(10,8))
      plt.subplot(1,2,1)
      plt.title("Static Activity (Closer View)")
      sns.distplot(train[train['Activity']=='SITTING']['tBodyAccMag-mean()'],hist = False, label = 'Sitting')
      sns.distplot(train[train['Activity']=='STANDING']['tBodyAccMag-mean()'],hist = False, label = 'STANDING')
      sns.distplot(train[train['Activity']=='LAYING']['tBodyAccMag-mean()'],hist = False, label = 'LAYING')
      plt.axis([-1.02, -0.5, 0, 35])
      plt.subplot(1,2,2)
      plt.title("Dynamic Activities(closer view)")
      sns.distplot(train[train["Activity"]=="WALKING"]['tBodyAccMag-mean()'],hist = False, label = 'WALKING')
      sns.distplot(train[train["Activity"]=="WALKING_DOWNSTAIRS"]['tBodyAccMag-mean()'],hist = False,label = 'WALKING_DOWNSTAIRS')
      sns.distplot(train[train["Activity"]=="WALKING_UPSTAIRS"]['tBodyAccMag-mean()'],hist = False, label = 'WALKING_UPSTAIRS')

[54]: <matplotlib.axes._subplots.AxesSubplot at 0x201364a10c8>
```

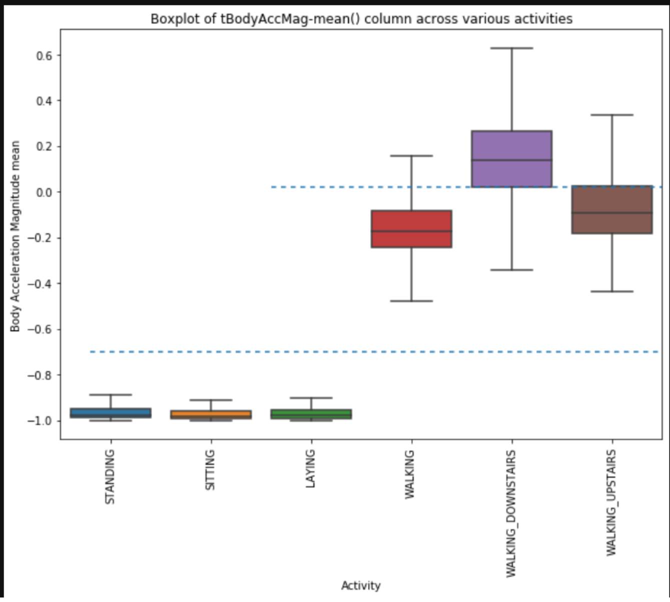



```
[53]: facetgrid = sns.FacetGrid(train, hue='Activity', height=5,aspect=3)
      facetgrid.map(sns.distplot,'tBodyAccMag-mean()', hist=False).add_legend()
      plt.annotate("Static Activities", xy=(-.996,21), xytext=(-0.9, 23),arrowprops={'arrowstyle': '-', 'ls': 'dashed'})
      plt.annotate("Static Activities", xy=(-.990,26), xytext=(-0.9, 23),arrowprops={'arrowstyle': '-', 'ls': 'dashed'})
      plt.annotate("Static Activities", xy=(-0.985,12), xytext=(-0.9, 23),arrowprops={'arrowstyle': '-', 'ls': 'dashed'})
      plt.annotate("Dynamic Activities", xy=(-0.2,3.25), xytext=(0.1, 9),arrowprops={'arrowstyle': '-', 'ls': 'dashed'})
      plt.annotate("Dynamic Activities", xy=(0.1,2.18), xytext=(0.1, 9),arrowprops={'arrowstyle': '-', 'ls': 'dashed'})
      plt.annotate("Dynamic Activities", xy=(-0.01,2.15), xytext=(0.1, 9),arrowprops={'arrowstyle': '-', 'ls': 'dashed'})

[53]: Text(0.1, 9, 'Dynamic Activities')
```



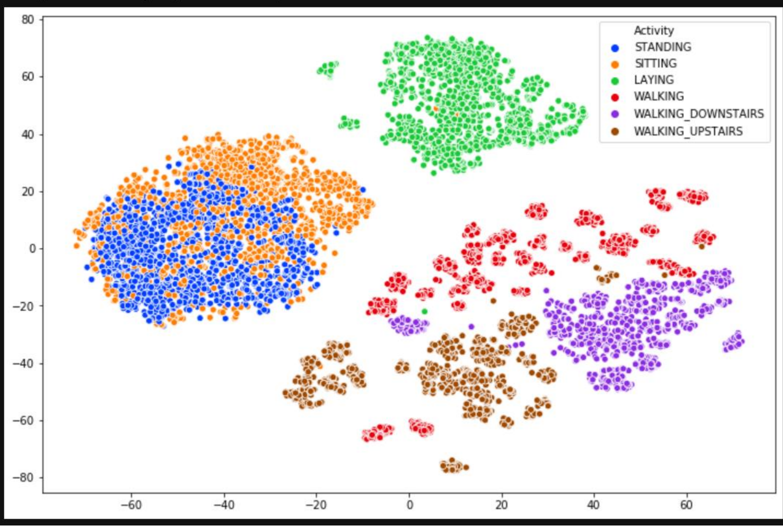
```
[55]: plt.figure(figsize=(10,7))
      sns.boxplot(x='Activity', y='tBodyAccMag-mean()',data=train, showfliers=False)
      plt.ylabel('Body Acceleration Magnitude mean')
      plt.title("Boxplot of tBodyAccMag-mean() column across various activities")
      plt.axhline(y=-0.7, xmin=0.05,dashes=(3,3))
      plt.axhline(y=0.020, xmin=0.35, dashes=(3,3))
      plt.xticks(rotation=90)
```

```
[55]: (array([0, 1, 2, 3, 4, 5]), <a list of 6 Text xticklabel objects>)
```

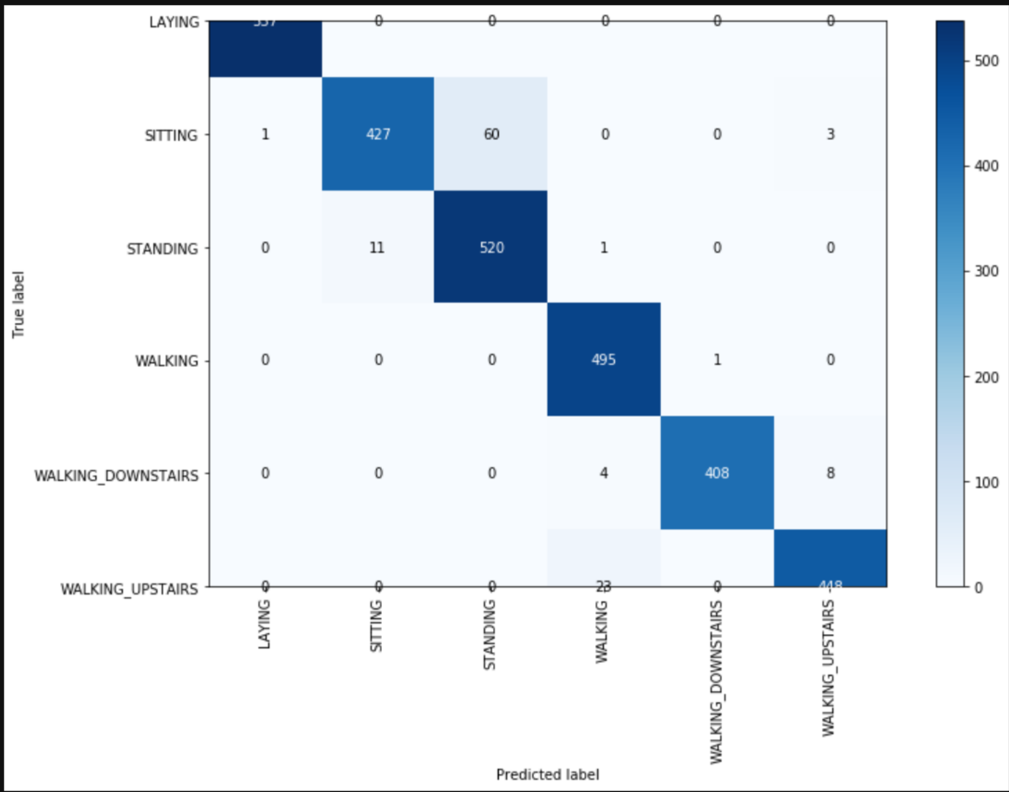



```
[61]: plt.figure(figsize=(12,8))
      sns.scatterplot(x =tsne[:, 0], y = tsne[:, 1], hue = train["Activity"],palette="bright")
```

```
[61]: <matplotlib.axes._subplots.AxesSubplot at 0x20137f18648>
```



```
[69]: cm = confusion_matrix(y_test.values,y_pred)
      plot_confusion_matrix(cm, np.unique(y_pred))  # plotting confusion matrix
```




```
[70]: #function to get best random search attributes
```

Decision Tree Classifier

```
[71]: # getting best random search attributes
      get_best_randomsearch_results(lr_classifier_rs)

      Best estimator :  LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
                         intercept_scaling=1, l1_ratio=None, max_iter=100,
                         multi_class='warn', n_jobs=None, penalty='l2',
                         random_state=None, solver='warn', tol=0.0001, verbose=0,
                         warm_start=False)
      Best set of parameters :  {'penalty': 'l2', 'C': 10}
      Best score :  0.9412404787812 84
```

```
[72]: from sklearn.tree import DecisionTreeClassifier
      parameters = {'max_depth':np.arange(2,10,2)}
      dt_classifier = DecisionTreeClassifier()
      dt_classifier_rs = RandomizedSearchCV(dt_classifier,param_distributions=parameters,random_state = 42)
      dt_classifier_rs.fit(X_train, y_train)
      y_pred = dt_classifier_rs.predict(X_test)
```

```
[73]: dt_accuracy = accuracy_score(y_true=y_test, y_pred=y_pred)
      print("Accuracy using Decision tree : ", dt_accuracy)

      Accuracy using Decision tree :  0.8096369189005769
```

HMM output:

```
[30]: # getting best random search attributes
      get_best_randomsearch_results(dt_classifier_rs)

      Best estimator :  DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=4,
                             max_features=None, max_leaf_nodes=None,
                             min_impurity_decrease=0.0, min_impurity_split=None,
                             min_samples_leaf=1, min_samples_split=2,
                             min_weight_fraction_leaf=0.0, presort=False,
                             random_state=None, splitter='best')
      Best set of parameters :  {'max_depth': 4}
      Best score :  0.8427638737758433

[31]: from hmmlearn import hmm
      model = hmm.GaussianHMM(n_components=3, covariance_type="full", n_iter=100)

[32]: model.fit(X_train)

[32]: GaussianHMM(algorithm='viterbi', covariance_type='full', covars_prior=0.01,
                  covars_weight=1, init_params='stmc', means_prior=0, means_weight=0,
                  min_covar=0.001, n_components=3, n_iter=100, params='stmc',
                  random_state=None, startprob_prior=1.0, tol=0.01,
                  transmat_prior=1.0, verbose=False)

[33]: y_pred_hmm = model.predict(X_test)

[34]: np.unique(y_pred_hmm)

[34]: array([0, 1, 2])
```