Home Work – 5
Machine Learning – CS633
Name: Guru Sarath Thangamani
UIN: 829009551

(a)
Motivation and Data:

Dataset - UCI Smartphone-Based Recognition of Human Activities and Postural Transitions
Number of samples – 10928
Number of features – 561
Machine Learning Task – Classification
Number of classes – 12

The potential applications of this classification problem are the major motivating factor to solve this problem.Human activity and posture recognition can be used in application like fitness monitoring. Smart phones are with everyone and carried around by the person where ever he goes. Hence, use of smart phones to monitor the activities of a person is a good idea to record almost all the physical activities of a person. We can give personalised feedback to the user regarding their daily activities, which includes lifestyle changes. This can help in improving the fitness and lifestyle of the person.
This technology can also be used to monitor old people with health issues. In case a person goes out and fall down due to some health issue, the smart phone can detect this and notify his family member. This ensure safety of old people.

Hence, this problem has massive potential to be used for different applications. This is one of the primary factor that motivates me to undertake this project.

This project seeks to be a building block of a complete software solution for a comprehensive health monitoring system using a smartphone.

(b)
Problem formulation:
This is a classification problem. Number of classes = 12.
The problem involves reading raw data from accelerometer and gyroscope, and then extracting features like mean, median, maximum frequency value, etc. during different activities. We use these features to predict the postural transitions of an individual.

Logistic Regression model –
A binary logistic regression model can be trained in a one-vs-rest fashion to achieve multiclass classification. Logistic regression is a simple linear model. Multiple logistic regressions can we trained to fit different straight lines and achieve multi-class classification.
Logistic Regression (One-vs-Rest) Performance –
Validation Accuracy: 96.7%
Test Accuracy: 96.7%

Neural Network model –
Number of layers = 3
Number of neurons in each layer = 50
Dropout probability = 0.3

Neural networks are the most versatile models, theoretically capable of learning any kind of dataset. One-hot encoding has to be done on the output values, so that a neural network can be trained.
Neural Network Performance –
Validation Accuracy: 93.6%
Test Accuracy: 93.3%
The logistic regression model gives better performance than the neural network (for this architecture) for this dataset.

(c)

Data pre-processing:

The dataset was tested for missing data, and it was found that this dataset contains no missing data.

The output values are categorical. The output values were converted to one-hot encoding.
One-hot encoding enables us to train neural networks, with each neuron associated with each output.

(d)

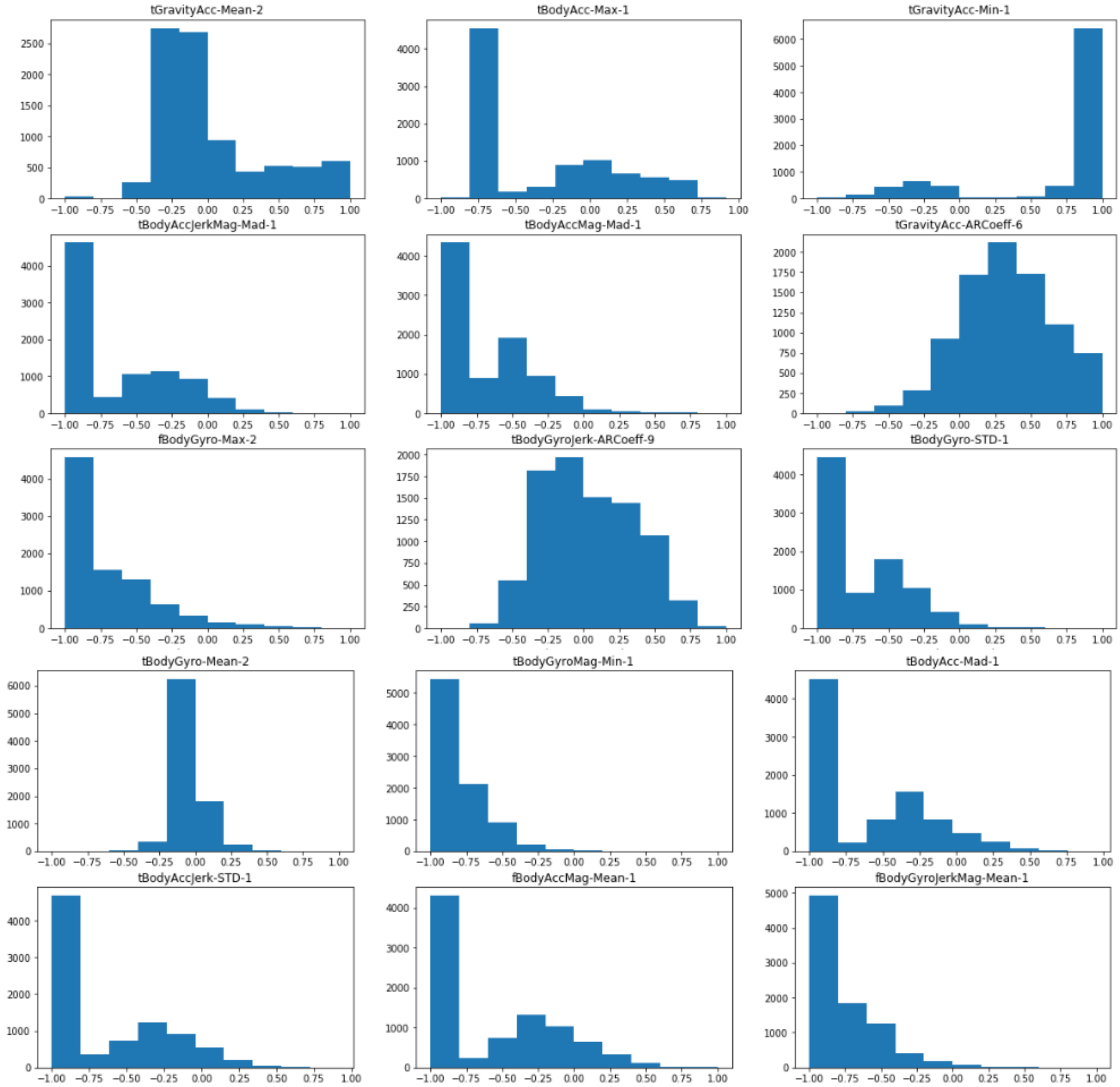
Data exploration:

Distribution of classes in train, test and validation sets-

```
For the Train dataset
Class :  0  :  WALKING  :  1367
Class :  1  :  WALKING_UPSTAIRS  :  1232
Class :  2  :  WALKING_DOWNSTAIRS  :  1147
Class :  3  :  SITTING  :  1445
Class :  4  :  STANDING  :  1581
Class :  5  :  LAYING  :  1553
Class :  6  :  STAND_TO_SIT  :  53
Class :  7  :  SIT_TO_STAND  :  25
Class :  8  :  SIT_TO_LIE  :  84
Class :  9  :  LIE_TO_SIT  :  70
Class :  10  :  STAND_TO_LIE  :  114
Class :  11  :  LIE_TO_STAND  :  71

For the Test dataset
Class :  0  :  WALKING  :  198
Class :  1  :  WALKING_UPSTAIRS  :  152
Class :  2  :  WALKING_DOWNSTAIRS  :  125
Class :  3  :  SITTING  :  185
Class :  4  :  STANDING  :  194
Class :  5  :  LAYING  :  192
Class :  6  :  STAND_TO_SIT  :  9
Class :  7  :  SIT_TO_STAND  :  3
Class :  8  :  SIT_TO_LIE  :  9
Class :  9  :  LIE_TO_SIT  :  9
Class :  10  :  STAND_TO_LIE  :  11
Class :  11  :  LIE_TO_STAND  :  6

For the Validation dataset
Class :  0  :  WALKING  :  157
Class :  1  :  WALKING_UPSTAIRS  :  160
Class :  2  :  WALKING_DOWNSTAIRS  :  135
Class :  3  :  SITTING  :  171
Class :  4  :  STANDING  :  203
Class :  5  :  LAYING  :  213
Class :  6  :  STAND_TO_SIT  :  8
```

```
Class :  7  :  SIT_TO_STAND  :  5
Class :  8  :  SIT_TO_LIE  :  14
Class :  9  :  LIE_TO_SIT  :  6
Class :  10  :  STAND_TO_LIE  :  14
Class :  11  :  LIE_TO_STAND  :  7
```

Histogram of features selected by forward feature selection algorithm (15 important features):



Fishers' criterion can be used to determine the feature that most correlated to the output. Fishers' criterion uses inter class and intra class distances to find the feature that most correlates to the output. Fishers' value was calculated for all the 561 features. The results are listed in the code output of this report.

Pearson's correlation can be used to find correlations between features. Since the data set contains 561 features, I'll be calculating Pearson's correlation between few selected features (features selected by the greedy forward feature selection algorithm) Pearson's correlation was calculated between all the selected features.

Results of correlation analysis -

Highly correlated features - tBodyAccJerkMag-Mad-1 and tBodyAccJerk-STD-1
Least correlated features - tBodyAcc-Max-1 and tBodyGyro-Mean-2

(e)

Split data into train, test, and validation:

The data was split into train, test and validation set.
Number of samples in Train set – 8742 (80% of the dataset)
Number of samples in Test set – 1093 (10% of the dataset)
Number of samples in Validation set – 1093 (10% of the dataset)

(f)

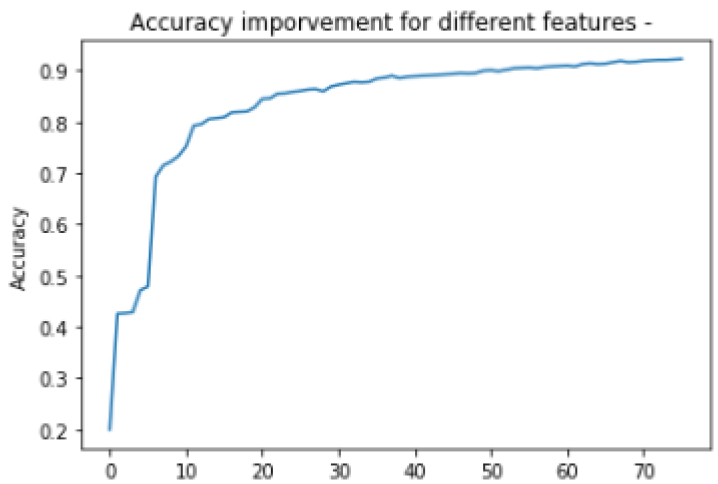
Feature selection:

Forward feature selection algorithm searched through all the features in a greedy fashion and generated a list of 15 best features that improves the performance of logistic regression model in the validation set.

List of top 15 selected features -

tGravityAcc-Mean-2
tBodyAcc-Max-1
tGravityAcc-Min-1
tBodyAccJerkMag-Mad-1
tBodyAccMag-Mad-1
tGravityAcc-ARCoeff-6
fBodyGyro-Max-2
tBodyGyroJerk-ARCoeff-9
tBodyGyro-STD-1
tBodyGyro-Mean-2
tBodyGyroMag-Min-1
tBodyAcc-Mad-1
tBodyAccJerk-STD-1
fBodyAccMag-Mean-1
fBodyGyroJerkMag-Mean-1

Test set accuracy achieved with only these 15 features = 87.2%

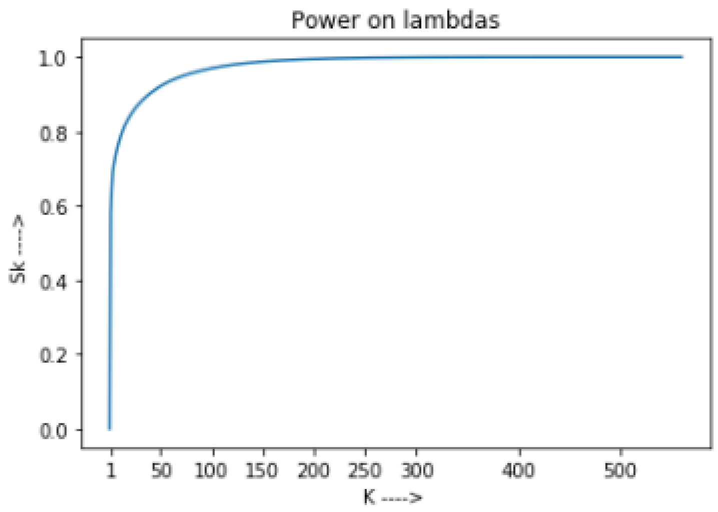


The graph above shows the improvement the accuracy as the number of features is increased.

(g)

Feature transformation:

PCA was implemented to reduce the dimensionality and select only the most important features (top 15) of data set.



Plot of sum of K eigenvalues by the sum of all the D eigenvalues reveals a lot of information regarding the features in the dataset.
Inference – Most of the variance (Close to 100%) in the data is captured using just 100 features, out of 561 total features.

Logistic regression model was trained for different PCA
Results of PCA for different K values –
K = 2
Accuracy Val = 0.5379688929551693
Accuracy Test = 0.5041171088746569
Accuracy Train = 0.5512468542667581

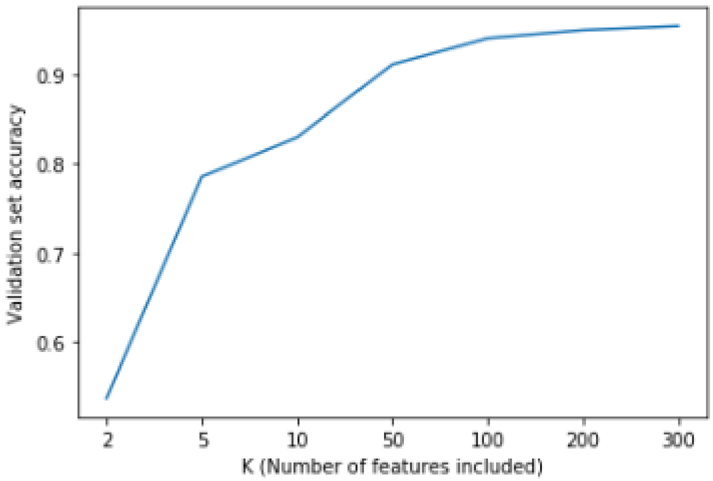
K = 5
Accuracy Val = 0.7859103385178408
Accuracy Test = 0.7301006404391582
Accuracy Train = 0.7862045298558682

K = 10
Accuracy Val = 0.8298261665141812
Accuracy Test = 0.848124428179323
Accuracy Train = 0.8523221230839625
K = 50
Accuracy Val = 0.9112534309240622
Accuracy Test = 0.9313815187557182
Accuracy Train = 0.945664607641272

K = 100
Accuracy Val = 0.9405306495882891
Accuracy Test = 0.958828911253431
Accuracy Train = 0.9735758407687028

K = 200
Accuracy Val = 0.94967978042086
Accuracy Test = 0.9624885635864593
Accuracy Train = 0.9871882864333105

K = 300
Accuracy Val = 0.9542543458371455
Accuracy Test = 0.9643183897529735
Accuracy Train = 0.9881034088309312



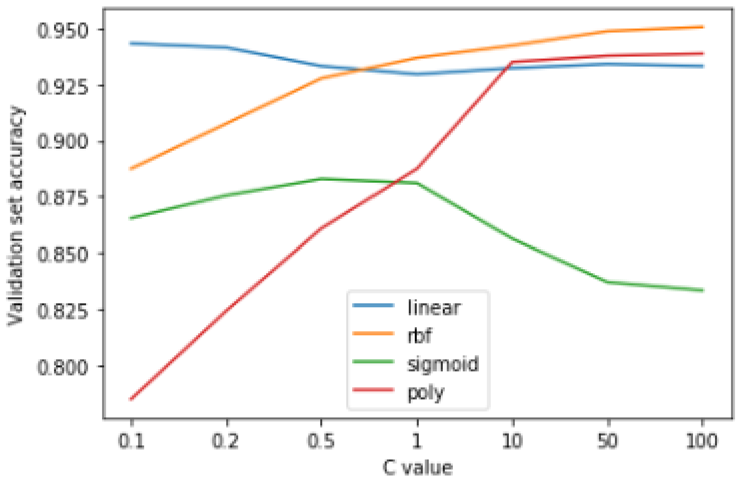
From the above data we can clearly see that the accuracy almost saturates after K=100. This clearly indicates that just having 100 features captures most of the variance in the dataset.

Neural network model was also trained (K=100):
Test accuracy obtained: 79.8%

(h)

Support vector machine:

PCA was performed on the dataset and top 100 features were used to train SVMs.
SVMs were trained for all combinations of inbuilt kernels and broad range of regularization values (C). C controls the regularization in the SVM model. The strength of the regularization is inversely proportional to C.



()

The above curve shows that ‘rbf’ kernel with C=100 achieves the highest accuracy.
The performance of linear kernel is almost unaffected for all regularization values and performs consistently good for all C values.

Highest accuracy reached with kernel = ‘rbf’ and C=100 : 95%

We can infer that we are able to get almost the same level of performance as that of using model trained with all the features, with just 100 features and SVM.

(i)

Ensemble learning:

Using Logistic regression estimator –
Number of estimators = 2 Val Accuracy = 0.7904849039341263 Test Accuracy = 0.8005489478499542
Number of estimators = 3 Val Accuracy = 0.7941445562671546 Test Accuracy = 0.8023787740164684
Number of estimators = 4 Val Accuracy = 0.8069533394327539 Test Accuracy = 0.8115279048490394
Number of estimators = 5 Val Accuracy = 0.8133577310155535 Test Accuracy = 0.8151875571820677
Number of estimators = 6 Val Accuracy = 0.7721866422689845 Test Accuracy = 0.7968892955169259
Number of estimators = 7 Val Accuracy = 0.7703568161024703 Test Accuracy = 0.7804208600182982
Number of estimators = 8 Val Accuracy = 0.747483989021043 Test Accuracy = 0.7822506861848124
Number of estimators = 9 Val Accuracy = 0.7749313815187557 Test Accuracy = 0.8042086001829826
Number of estimators = 10 Val Accuracy = 0.787740164684355 Test Accuracy = 0.807868252516011
Number of estimators = 11 Val Accuracy = 0.7346752058554438 Test Accuracy = 0.7749313815187557
Number of estimators = 12 Val Accuracy = 0.8051235132662397 Test Accuracy = 0.8161024702653248
Number of estimators = 13 Val Accuracy = 0.8014638609332113 Test Accuracy = 0.817932296431839
Number of estimators = 14 Val Accuracy = 0.7749313815187557 Test Accuracy = 0.8005489478499542
Number of estimators = 15 Val Accuracy = 0.797804208600183 Test Accuracy = 0.817932296431839
Number of estimators = 16 Val Accuracy = 0.8051235132662397 Test Accuracy = 0.8161024702653248
Number of estimators = 17 Val Accuracy = 0.7685269899359561 Test Accuracy = 0.8005489478499542
Number of estimators = 18 Val Accuracy = 0.797804208600183 Test Accuracy = 0.8124428179322964
Number of estimators = 19 Val Accuracy = 0.8087831655992681 Test Accuracy = 0.8133577310155535
With logistic regression model, best validation accuracy was obtained with 5 estimators – 81.3%

Using decision tree estimator –
Number of estimators = 1 Val Accuracy = 0.5233302836230558 Test Accuracy = 0.5297346752058555
Number of estimators = 2 Val Accuracy = 0.5800548947849954 Test Accuracy = 0.5507776761207686
Number of estimators = 3 Val Accuracy = 0.6550777676120768 Test Accuracy = 0.645928636779506
Number of estimators = 4 Val Accuracy = 0.6907593778591034 Test Accuracy = 0.6770356816102471
Number of estimators = 5 Val Accuracy = 0.6605672461116194 Test Accuracy = 0.6678865507776761
Number of estimators = 6 Val Accuracy = 0.6303751143641354 Test Accuracy = 0.6532479414455626
Number of estimators = 7 Val Accuracy = 0.6907593778591034 Test Accuracy = 0.6733760292772186
Number of estimators = 8 Val Accuracy = 0.7026532479414456 Test Accuracy = 0.6916742909423604
Number of estimators = 9 Val Accuracy = 0.6953339432753889 Test Accuracy = 0.6999085086916743
Number of estimators = 10 Val Accuracy = 0.7108874656907593 Test Accuracy = 0.6816102470265325
Number of estimators = 11 Val Accuracy = 0.7255260750228728 Test Accuracy = 0.7118023787740164
Number of estimators = 12 Val Accuracy = 0.7090576395242452 Test Accuracy = 0.7008234217749314
Number of estimators = 13 Val Accuracy = 0.7145471180237878 Test Accuracy = 0.7099725526075022
Number of estimators = 14 Val Accuracy = 0.737419945105215 Test Accuracy = 0.7145471180237878
Number of estimators = 15 Val Accuracy = 0.7282708142726441 Test Accuracy = 0.7053979871912168
Number of estimators = 16 Val Accuracy = 0.7255260750228728 Test Accuracy = 0.6971637694419031
Number of estimators = 17 Val Accuracy = 0.7136322049405307 Test Accuracy = 0.7108874656907593
Number of estimators = 18 Val Accuracy = 0.8133577310155535 Test Accuracy = 0.7904849039341263
Number of estimators = 19 Val Accuracy = 0.8032936870997255 Test Accuracy = 0.7795059469350412
With Decision tree model, best validation accuracy was obtained with 18 estimators – 81.3%

For this dataset, the performance of ada-boost is less compared to the model trained with selected features.

Final_Project

April 30, 2020

1 Machine Learning

2 Homework 5

3 Name: Guru Sarath Thangamani

4 UIN: 829009551

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras import models
from keras import layers
from keras.utils import to_categorical
```

```
Using TensorFlow backend.
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
X = np.genfromtxt("Data_X.txt", delimiter=" ", skip_header=1)
y = np.genfromtxt("Data_y.txt", delimiter=" ", skip_header=1, dtype=int)
y = y - 1
print('X shape = ', X.shape)
print('y shape = ', y.shape)
```

```
X shape =  (10928, 561)
y shape =  (10928,)
```

[29]:

```
classes =␣
↪['WALKING','WALKING_UPSTAIRS','WALKING_DOWNSTAIRS','SITTING','STANDING','LAYING','STAND_TO_
```

5 (e) Split data into train, test, and validation.

[4]:
```
X_train, X_val_test, y_train, y_val_test = train_test_split(X, y, test_size=0.
↪2, random_state=1)
X_test, X_val, y_test, y_val = train_test_split(X_val_test, y_val_test,␣
↪test_size=0.5, random_state=3)

print('TrainX shape = ', X_train.shape)
print('TestX shape = ', X_test.shape)
print('ValX shape = ', X_val.shape)
```

```
TrainX shape =  (8742, 561)
TestX shape =  (1093, 561)
ValX shape =  (1093, 561)
```

6 (c) Data pre-processing.

[5]:
```
y_train_categorical = to_categorical(y_train)
y_test_categorical = to_categorical(y_test)
y_val_categorical = to_categorical(y_val)

print('Train y shape = ', y_train_categorical.shape)
print('Test y shape = ', y_test_categorical.shape)
print('Val y shape = ', y_val_categorical.shape)
```

```
Train y shape =  (8742, 12)
Test y shape =  (1093, 12)
Val y shape =  (1093, 12)
```

7 Check for missing data

[55]:
```
MissingMatrix = np.isnan(X_train)
MissingMatrix2 = np.isnan(y_train)
if np.sum(MissingMatrix) > 0 or np.sum(MissingMatrix2) > 0:
    print('Train set contains missing data !!')
else:
    print('No missing data in the train set :)')

MissingMatrix = np.isnan(X_test)
MissingMatrix2 = np.isnan(y_test)
if np.sum(MissingMatrix) > 0 or np.sum(MissingMatrix2) > 0:
    print('Test set contains missing data !!')
```

```
else:
    print('No missing data in the test set :)')

MissingMatrix = np.isnan(X_val)
MissingMatrix2 = np.isnan(y_val)
if np.sum(MissingMatrix) > 0 or np.sum(MissingMatrix2) > 0:
    print('Validation set contains missing data !!')
else:
    print('No missing data in the Validation set :)')
```

```
No missing data in the train set :)
No missing data in the test set :)
No missing data in the Validation set :)
```

8 (d) Data exploration.

9 Number of samples in each class output type

[143]:
```
print('For the Train dataset')
for i in range(len(classes)):
    print('Class : ', i , ' : ', classes[i] , ' : ', np.sum(y_train == i))

print('For the Test dataset')
for i in range(len(classes)):
    print('Class : ', i , ' : ', classes[i] , ' : ', np.sum(y_test == i))

print('For the Validation dataset')
for i in range(len(classes)):
    print('Class : ', i, ' : ' ,classes[i] , ' : ', np.sum(y_val == i))
```

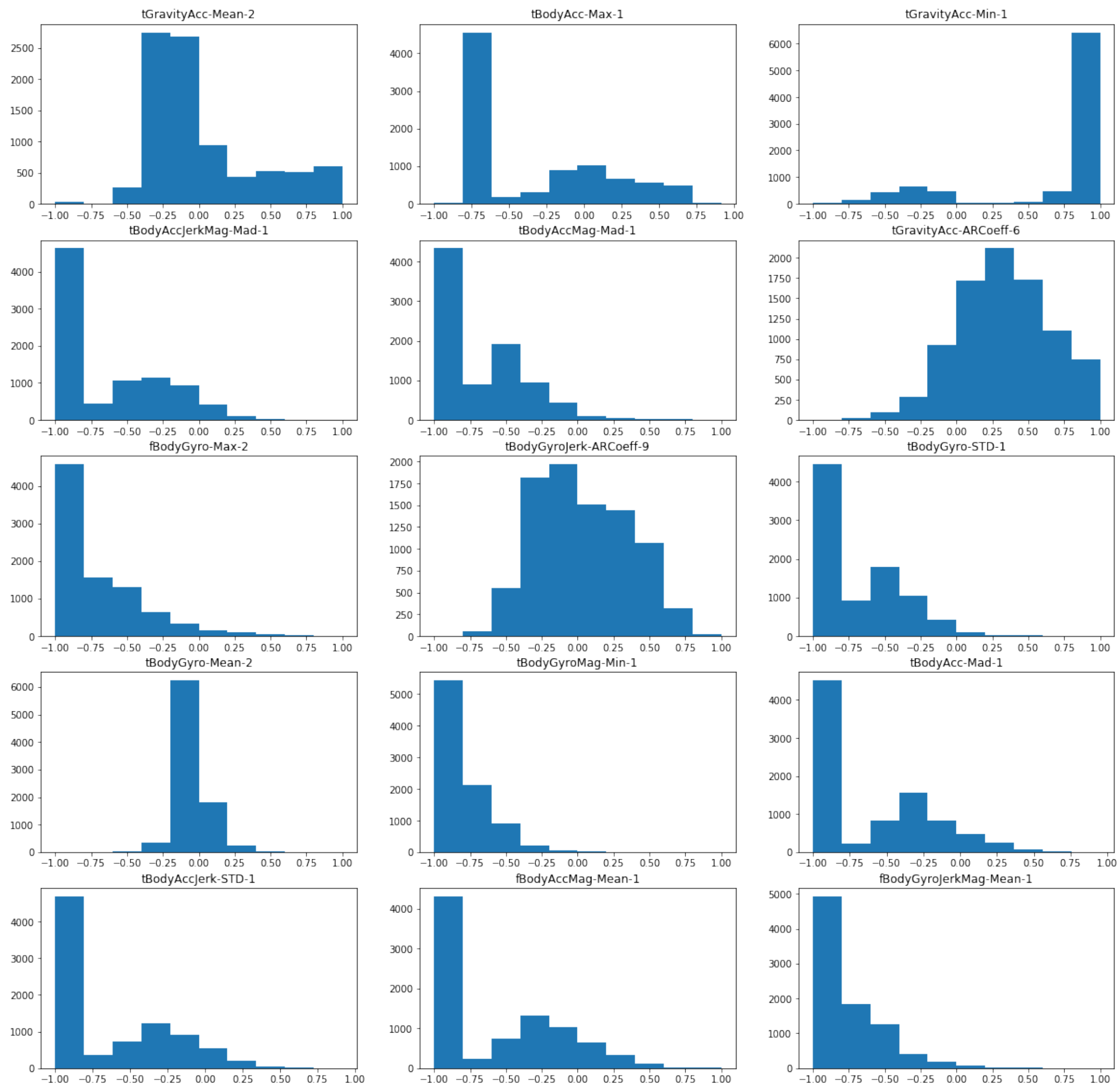
```
For the Train dataset
Class :  0  :  WALKING  :  1367
Class :  1  :  WALKING_UPSTAIRS  :  1232
Class :  2  :  WALKING_DOWNSTAIRS  :  1147
Class :  3  :  SITTING  :  1445
Class :  4  :  STANDING  :  1581
Class :  5  :  LAYING  :  1553
Class :  6  :  STAND_TO_SIT  :  53
Class :  7  :  SIT_TO_STAND  :  25
Class :  8  :  SIT_TO_LIE  :  84
Class :  9  :  LIE_TO_SIT  :  70
Class :  10  :  STAND_TO_LIE  :  114
Class :  11  :  LIE_TO_STAND  :  71
For the Test dataset
Class :  0  :  WALKING  :  198
Class :  1  :  WALKING_UPSTAIRS  :  152
```

```
Class :  2  :  WALKING_DOWNSTAIRS  :  125
Class :  3  :  SITTING  :  185
Class :  4  :  STANDING  :  194
Class :  5  :  LAYING  :  192
Class :  6  :  STAND_TO_SIT  :  9
Class :  7  :  SIT_TO_STAND  :  3
Class :  8  :  SIT_TO_LIE  :  9
Class :  9  :  LIE_TO_SIT  :  9
Class :  10  :  STAND_TO_LIE  :  11
Class :  11  :  LIE_TO_STAND  :  6
For the Validation dataset
Class :  0  :  WALKING  :  157
Class :  1  :  WALKING_UPSTAIRS  :  160
Class :  2  :  WALKING_DOWNSTAIRS  :  135
Class :  3  :  SITTING  :  171
Class :  4  :  STANDING  :  203
Class :  5  :  LAYING  :  213
Class :  6  :  STAND_TO_SIT  :  8
Class :  7  :  SIT_TO_STAND  :  5
Class :  8  :  SIT_TO_LIE  :  14
Class :  9  :  LIE_TO_SIT  :  6
Class :  10  :  STAND_TO_LIE  :  14
Class :  11  :  LIE_TO_STAND  :  7
```

10 Histogram

[149]:
```
selectedFeatures_index = [41, 9, 52, 228, 202, 70, 433, 193, 123, 121, 243, 6,
 ↪83, 502, 541]
```

[152]:
```
fig,axs = plt.subplots(5,3, figsize=(20,20))
for i,feat in enumerate(selectedFeatures_index):
    plt.subplot(5,3,i+1)
    plt.title(features[feat])
    plt.hist(X_train[:,feat])
```



11 Fishers_criterion

[41]:
```
def Fishers_criterion(X,y,classList, featureIndex):

    d = featureIndex

    mean_d = np.mean(X[:,d])
    mean_kd_List = []
    Numerator = 0
    for classX in classList:

        y_new = (y == classX)
        sample_index = 0
```

```
        mean_kd = 0
        for yCheck in y_new:
            if yCheck:
                mean_kd += X[sample_index, d]

            sample_index += 1

        mean_kd = mean_kd / np.sum(y_new)
        mean_kd_List.append(mean_kd)

        sample_index = 0
        for yCheck in y_new:
            if yCheck:
                Numerator += (X[sample_index,d] - mean_kd)**2

    denominator = 0
    index = 0
    for classX in classList:
        denominator += (mean_d - mean_kd_List[index])
        index += 1

    return Numerator / denominator
```

[43]:
```
for featureIndex in range(X.shape[1]):
    print(
            features[featureIndex],
            '    ',
            Fishers_criterion(X_train,y_train,[i for i in range(np.
 ↪max(y_train))], featureIndex)
         )
```

```
tBodyAcc-Mean-1      346.2946737127763
tBodyAcc-Mean-2      -1291.523857992131
tBodyAcc-Mean-3      -201.6430868433005
tBodyAcc-STD-1      -2027.6905052563761
tBodyAcc-STD-2      -518.2543566783183
tBodyAcc-STD-3      -307.4148314154306
tBodyAcc-Mad-1      -1881.8039471049865
tBodyAcc-Mad-2      -419.1340663207205
tBodyAcc-Mad-3      -232.24582131523792
tBodyAcc-Max-1      -4929.107662580646
tBodyAcc-Max-2      -375.21369800374674
tBodyAcc-Max-3      -352.08061423459964
tBodyAcc-Min-1      746.2626528052139
tBodyAcc-Min-2      1379.1816420338805
tBodyAcc-Min-3      527.1803835701816
tBodyAcc-SMA-1      -366.70343700650545
```

tBodyAcc-Energy-1	-615.3884824126255
tBodyAcc-Energy-2	-65.02657545768645
tBodyAcc-Energy-3	-52.5042777875331
tBodyAcc-IQR-1	-1460.3731755675592
tBodyAcc-IQR-2	-252.61864586924926
tBodyAcc-IQR-3	-153.8873772963894
tBodyAcc-ropy-1	-1705.0996781101687
tBodyAcc-ropy-1	-1932.0762961223443
tBodyAcc-ropy-1	-698.4205681984639
tBodyAcc-ARCoeff-1	671.6351304713354
tBodyAcc-ARCoeff-2	-614.2194799204639
tBodyAcc-ARCoeff-3	775.8568892291053
tBodyAcc-ARCoeff-4	874.8829528904765
tBodyAcc-ARCoeff-5	643.2014430771025
tBodyAcc-ARCoeff-6	-1954.141723315826
tBodyAcc-ARCoeff-7	2733.6868526042626
tBodyAcc-ARCoeff-8	-3588.1878323754036
tBodyAcc-ARCoeff-9	1212.3300095999673
tBodyAcc-ARCoeff-10	-1834.6552908835447
tBodyAcc-ARCoeff-11	239.97414995599735
tBodyAcc-ARCoeff-12	-250.2762198611411
tBodyAcc-Correlation-1	3204.9507751770047
tBodyAcc-Correlation-2	-2188.460548795115
tBodyAcc-Correlation-3	-421.92047771133304
tGravityAcc-Mean-1	-26957.364174532602
tGravityAcc-Mean-2	-2101.1277495002287
tGravityAcc-Mean-3	-11018.819217040469
tGravityAcc-STD-1	-58.87762550870549
tGravityAcc-STD-2	-59.560579565783506
tGravityAcc-STD-3	-45.685505536688055
tGravityAcc-Mad-1	-55.75148255817907
tGravityAcc-Mad-2	-57.37994307033903
tGravityAcc-Mad-3	-44.93708953535567
tGravityAcc-Max-1	-2784.1675740143546
tGravityAcc-Max-2	-773.8311122825874
tGravityAcc-Max-3	-1219.3344259760747
tGravityAcc-Min-1	3387.086294799823
tGravityAcc-Min-2	1778.2446511441267
tGravityAcc-Min-3	1657.2250589193134
tGravityAcc-SMA-1	-709.421814521286
tGravityAcc-Energy-1	15581.804411087905
tGravityAcc-Energy-2	-14799.579295411586
tGravityAcc-Energy-3	5152.503003278431
tGravityAcc-IQR-1	-43.999252083693534
tGravityAcc-IQR-2	-48.93409909706269
tGravityAcc-IQR-3	-41.77455475164875
tGravityAcc-ropy-1	-329.42361696178557
tGravityAcc-ropy-1	-207.5418184350434

tGravityAcc-ropy-1	-212.56221473876133
tGravityAcc-ARCoeff-1	240.45704664398727
tGravityAcc-ARCoeff-2	-256.74604267098744
tGravityAcc-ARCoeff-3	284.96565533196053
tGravityAcc-ARCoeff-4	-328.37343619600625
tGravityAcc-ARCoeff-5	135.7194937977448
tGravityAcc-ARCoeff-6	-111.0349822079693
tGravityAcc-ARCoeff-7	92.71683915736753
tGravityAcc-ARCoeff-8	-82.33794074244346
tGravityAcc-ARCoeff-9	140.57899147515525
tGravityAcc-ARCoeff-10	-129.5670390671661
tGravityAcc-ARCoeff-11	121.61812589592449
tGravityAcc-ARCoeff-12	-116.86630120149617
tGravityAcc-Correlation-1	2859.7154596537307
tGravityAcc-Correlation-2	-12347.159437661523
tGravityAcc-Correlation-3	-3691.7685128571056
tBodyAccJerk-Mean-1	261.27578691585086
tBodyAccJerk-Mean-2	-855.9431775640007
tBodyAccJerk-Mean-3	-77.31214278396084
tBodyAccJerk-STD-1	5043.411978932907
tBodyAccJerk-STD-2	-11469.518758613778
tBodyAccJerk-STD-3	-43220.83989239945
tBodyAccJerk-Mad-1	4153.598497428412
tBodyAccJerk-Mad-2	-14870.401794360288
tBodyAccJerk-Mad-3	-30584.997514143637
tBodyAccJerk-Max-1	-225107.71732624955
tBodyAccJerk-Max-2	-3832.643543055823
tBodyAccJerk-Max-3	-5717.114674679865
tBodyAccJerk-Min-1	-7109.8221305214765
tBodyAccJerk-Min-2	6432.916636563127
tBodyAccJerk-Min-3	-53368.14160211756
tBodyAccJerk-SMA-1	17556.967682542574
tBodyAccJerk-Energy-1	872.7236982226261
tBodyAccJerk-Energy-2	1310.8475982947275
tBodyAccJerk-Energy-3	472.84296900153544
tBodyAccJerk-IQR-1	3551.377881713675
tBodyAccJerk-IQR-2	-13145.709651990841
tBodyAccJerk-IQR-3	-9556.549887911944
tBodyAccJerk-ropy-1	-2996.3905658873027
tBodyAccJerk-ropy-1	-3279.3267895824406
tBodyAccJerk-ropy-1	-2536.536701206829
tBodyAccJerk-ARCoeff-1	828.0866770149248
tBodyAccJerk-ARCoeff-2	-1159.7099179251816
tBodyAccJerk-ARCoeff-3	612.3168659626423
tBodyAccJerk-ARCoeff-4	283.558182292002
tBodyAccJerk-ARCoeff-5	978.2199106103681
tBodyAccJerk-ARCoeff-6	-11887.186321414421
tBodyAccJerk-ARCoeff-7	756.9344408637791

tBodyAccJerk-ARCoeff-8	907.9580677529909
tBodyAccJerk-ARCoeff-9	1516.6844235854069
tBodyAccJerk-ARCoeff-10	1327.3841515418019
tBodyAccJerk-ARCoeff-11	77.58757423442766
tBodyAccJerk-ARCoeff-12	452.90088756532106
tBodyAccJerk-Correlation-1	-3688.766375907915
tBodyAccJerk-Correlation-2	-72480.31260477872
tBodyAccJerk-Correlation-3	-2688.0461556174378
tBodyGyro-Mean-1	122.88725534717699
tBodyGyro-Mean-2	-37.564241817811
tBodyGyro-Mean-3	150.04967951119974
tBodyGyro-STD-1	-1225.5251342448157
tBodyGyro-STD-2	-618.8721482128445
tBodyGyro-STD-3	-312.5175550690346
tBodyGyro-Mad-1	-1333.6303007585536
tBodyGyro-Mad-2	-595.0536287700587
tBodyGyro-Mad-3	-293.11325191182823
tBodyGyro-Max-1	-498.15140332555444
tBodyGyro-Max-2	-609.5532578799556
tBodyGyro-Max-3	-320.72026966235086
tBodyGyro-Min-1	973.8970611052563
tBodyGyro-Min-2	448.37420886927373
tBodyGyro-Min-3	412.47120017235864
tBodyGyro-SMA-1	-670.175943197605
tBodyGyro-Energy-1	-299.5955113969668
tBodyGyro-Energy-2	-287.87016723026125
tBodyGyro-Energy-3	-85.80524771048836
tBodyGyro-IQR-1	-1597.3796579762034
tBodyGyro-IQR-2	-612.232702636407
tBodyGyro-IQR-3	-274.2384421733757
tBodyGyro-ropy-1	-1335.085626894849
tBodyGyro-ropy-1	-411.5026043135051
tBodyGyro-ropy-1	-2030.642299397138
tBodyGyro-ARCoeff-1	1594.528653136714
tBodyGyro-ARCoeff-2	-993.2157916214188
tBodyGyro-ARCoeff-3	35.28008839467155
tBodyGyro-ARCoeff-4	-630.3975278259694
tBodyGyro-ARCoeff-5	95.59510368832512
tBodyGyro-ARCoeff-6	-121.77236903350695
tBodyGyro-ARCoeff-7	806.5541215269527
tBodyGyro-ARCoeff-8	-2542.7389624542143
tBodyGyro-ARCoeff-9	1520.7820625860886
tBodyGyro-ARCoeff-10	-1983.9684730326253
tBodyGyro-ARCoeff-11	1149.2869918811994
tBodyGyro-ARCoeff-12	-212.92897033233425
tBodyGyro-Correlation-1	-2463.374636360201
tBodyGyro-Correlation-2	1118.7521877693953
tBodyGyro-Correlation-3	567.0005037744519

tBodyGyroJerk-Mean-1	-5624.789616263992
tBodyGyroJerk-Mean-2	-406.7359023426579
tBodyGyroJerk-Mean-3	100.17179186836835
tBodyGyroJerk-STD-1	-7811.285591995061
tBodyGyroJerk-STD-2	20533.843353431632
tBodyGyroJerk-STD-3	-4877.479166109234
tBodyGyroJerk-Mad-1	-14539.740762674086
tBodyGyroJerk-Mad-2	11766.896352394857
tBodyGyroJerk-Mad-3	-5435.043056699472
tBodyGyroJerk-Max-1	-2160.2046611787796
tBodyGyroJerk-Max-2	-1464814.9020190071
tBodyGyroJerk-Max-3	-3779.2797405964066
tBodyGyroJerk-Min-1	3881.283194772033
tBodyGyroJerk-Min-2	-25350.23162461965
tBodyGyroJerk-Min-3	3234.795886592594
tBodyGyroJerk-SMA-1	-16701.861600659864
tBodyGyroJerk-Energy-1	740.0569876656381
tBodyGyroJerk-Energy-2	462.96330079115717
tBodyGyroJerk-Energy-3	642.1758770961136
tBodyGyroJerk-IQR-1	-94017.329568776
tBodyGyroJerk-IQR-2	6447.007013756065
tBodyGyroJerk-IQR-3	-6579.1775445190915
tBodyGyroJerk-ropy-1	-3169.08439954157
tBodyGyroJerk-ropy-1	-2305.89048138783
tBodyGyroJerk-ropy-1	-2192.893576628721
tBodyGyroJerk-ARCoeff-1	1562.0331472911753
tBodyGyroJerk-ARCoeff-2	-816.4439656759747
tBodyGyroJerk-ARCoeff-3	458.672647746615
tBodyGyroJerk-ARCoeff-4	270.159538878442
tBodyGyroJerk-ARCoeff-5	340.28942086139887
tBodyGyroJerk-ARCoeff-6	-552.9175448471844
tBodyGyroJerk-ARCoeff-7	1566.7729364222378
tBodyGyroJerk-ARCoeff-8	28.49133994706895
tBodyGyroJerk-ARCoeff-9	1805.1532537560545
tBodyGyroJerk-ARCoeff-10	-2892.0999340505987
tBodyGyroJerk-ARCoeff-11	384.9498796892645
tBodyGyroJerk-ARCoeff-12	163.34564370329915
tBodyGyroJerk-Correlation-1	-1542.3973031396806
tBodyGyroJerk-Correlation-2	415.8633754726586
tBodyGyroJerk-Correlation-3	491.12516715409856
tBodyAccMag-Mean-1	-405.5185798582338
tBodyAccMag-STD-1	-309.0690966066833
tBodyAccMag-Mad-1	-244.08358372113594
tBodyAccMag-Max-1	-696.9583772618921
tBodyAccMag-Min-1	-199.99396117987868
tBodyAccMag-SMA-1	-405.5185798582338
tBodyAccMag-Energy-1	-142.39168634442436
tBodyAccMag-IQR-1	-153.39879258534057

```
tBodyAccMag-ropy-1      -1515.1693568743133
tBodyAccMag-ARCoeff-1      140.30945438983568
tBodyAccMag-ARCoeff-2      -109.2216392288144
tBodyAccMag-ARCoeff-3      90.96010140760174
tBodyAccMag-ARCoeff-4      -84.81639392372719
tGravityAccMag-Mean-1      -405.5185798582338
tGravityAccMag-STD-1      -309.0690966066833
tGravityAccMag-Mad-1      -244.08358372113594
tGravityAccMag-Max-1      -696.9583772618921
tGravityAccMag-Min-1      -199.99396117987868
tGravityAccMag-SMA-1      -405.5185798582338
tGravityAccMag-Energy-1      -142.39168634442436
tGravityAccMag-IQR-1      -153.39879258534057
tGravityAccMag-ropy-1      -1515.1693568743133
tGravityAccMag-ARCoeff-1      140.30945438983568
tGravityAccMag-ARCoeff-2      -109.2216392288144
tGravityAccMag-ARCoeff-3      90.96010140760174
tGravityAccMag-ARCoeff-4      -84.81639392372719
tBodyAccJerkMag-Mean-1      15689.115598708917
tBodyAccJerkMag-STD-1      59499.56792907169
tBodyAccJerkMag-Mad-1      17214.88167612318
tBodyAccJerkMag-Max-1      -15890.4193186738
tBodyAccJerkMag-Min-1      3644.2062761538104
tBodyAccJerkMag-SMA-1      15689.115598708917
tBodyAccJerkMag-Energy-1      861.0409031670557
tBodyAccJerkMag-IQR-1      8874.233255387011
tBodyAccJerkMag-ropy-1      -3455.0434410113917
tBodyAccJerkMag-ARCoeff-1      1510.4311198487976
tBodyAccJerkMag-ARCoeff-2      -2098.4758172790785
tBodyAccJerkMag-ARCoeff-3      3352.8396554317865
tBodyAccJerkMag-ARCoeff-4      -8500.365646006177
tBodyGyroMag-Mean-1      -668.991602018128
tBodyGyroMag-STD-1      -516.2188519619626
tBodyGyroMag-Mad-1      -571.3081185044618
tBodyGyroMag-Max-1      -497.70785366638825
tBodyGyroMag-Min-1      -255.18771537344617
tBodyGyroMag-SMA-1      -668.991602018128
tBodyGyroMag-Energy-1      -258.436875595323
tBodyGyroMag-IQR-1      -568.0046342403245
tBodyGyroMag-ropy-1      -3394.165660391125
tBodyGyroMag-ARCoeff-1      939.4948183449752
tBodyGyroMag-ARCoeff-2      -821.7924171551593
tBodyGyroMag-ARCoeff-3      65.20727721523008
tBodyGyroMag-ARCoeff-4      -101.6098665977811
tBodyGyroJerkMag-Mean-1      -19784.69069494845
tBodyGyroJerkMag-STD-1      -6540.149736345576
tBodyGyroJerkMag-Mad-1      -6432.316597580882
tBodyGyroJerkMag-Max-1      -5987.868397311854
```

tBodyGyroJerkMag-Min-1	4305.205078619904
tBodyGyroJerkMag-SMA-1	-19784.69069494845
tBodyGyroJerkMag-Energy-1	464.8555830102983
tBodyGyroJerkMag-IQR-1	-6335.454408777969
tBodyGyroJerkMag-ropy-1	-2786.2583047280314
tBodyGyroJerkMag-ARCoeff-1	1595.8460262113745
tBodyGyroJerkMag-ARCoeff-2	-761.6481870325316
tBodyGyroJerkMag-ARCoeff-3	5902.9310176562185
tBodyGyroJerkMag-ARCoeff-4	4204.689302844445
fBodyAcc-Mean-1	-3943.229949837471
fBodyAcc-Mean-2	-1334.5098565171156
fBodyAcc-Mean-3	-722.1638223620582
fBodyAcc-STD-1	-1811.9989871557439
fBodyAcc-STD-2	-391.41809282252626
fBodyAcc-STD-3	-234.58838100936381
fBodyAcc-Mad-1	-3549.7113393888962
fBodyAcc-Mad-2	-1134.496040797522
fBodyAcc-Mad-3	-656.7932180572552
fBodyAcc-Max-1	-1349.0960951261266
fBodyAcc-Max-2	-212.7668467014196
fBodyAcc-Max-3	-136.60130336239536
fBodyAcc-Min-1	-456.1586777416869
fBodyAcc-Min-2	-108.3728122411898
fBodyAcc-Min-3	-79.1887008555359
fBodyAcc-SMA-1	-1504.55670424624
fBodyAcc-Energy-1	-19320.8740465681
fBodyAcc-Energy-2	-113.9172117204955
fBodyAcc-Energy-3	-84.73391417872654
fBodyAcc-IQR-1	-36643.63733789534
fBodyAcc-IQR-2	-3771.020708150198
fBodyAcc-IQR-3	-4238.190062579921
fBodyAcc-ropy-1	-2708.423632048702
fBodyAcc-ropy-1	-2882.21593721532
fBodyAcc-ropy-1	-2385.843351436951
fBodyAcc-MaxInds-1	79.6262708840174
fBodyAcc-MaxInds-2	83.65356264779003
fBodyAcc-MaxInds-3	157.32696775452058
fBodyAcc-MeanFreq-1	185.40962737289533
fBodyAcc-MeanFreq-2	61.14690377674844
fBodyAcc-MeanFreq-3	200.43946803086982
fBodyAcc-Skewness-1	-1173.374768695212
fBodyAcc-Kurtosis-1	-1275.5095006241654
fBodyAcc-Skewness-1	-82.84683602592806
fBodyAcc-Kurtosis-1	-97.10273743363243
fBodyAcc-Skewness-1	-115.84452120360201
fBodyAcc-Kurtosis-1	-172.8079333273603
fBodyAcc-BandsEnergyOld-1	-3153.3212271209536
fBodyAcc-BandsEnergyOld-2	800.054742056164

fBodyAcc-BandsEnergyOld-3	825.6907167884248
fBodyAcc-BandsEnergyOld-4	817.1380579992302
fBodyAcc-BandsEnergyOld-5	754.9850547293394
fBodyAcc-BandsEnergyOld-6	919.311900581152
fBodyAcc-BandsEnergyOld-7	1191.8270261288594
fBodyAcc-BandsEnergyOld-8	-681.2197495698941
fBodyAcc-BandsEnergyOld-9	-8614.089612468691
fBodyAcc-BandsEnergyOld-10	913.0658770905179
fBodyAcc-BandsEnergyOld-11	811.4444224072992
fBodyAcc-BandsEnergyOld-12	4515.162121730244
fBodyAcc-BandsEnergyOld-13	-15345.649614557005
fBodyAcc-BandsEnergyOld-14	946.3827233314183
fBodyAcc-BandsEnergyOld-15	-58.94459857258014
fBodyAcc-BandsEnergyOld-16	1165.3822846770654
fBodyAcc-BandsEnergyOld-17	1024.831459919985
fBodyAcc-BandsEnergyOld-18	-1613.0596553308342
fBodyAcc-BandsEnergyOld-19	-1003.8009476853287
fBodyAcc-BandsEnergyOld-20	-3109.243179250258
fBodyAcc-BandsEnergyOld-21	-1189.2501909305415
fBodyAcc-BandsEnergyOld-22	-49.481805378782155
fBodyAcc-BandsEnergyOld-23	-96.43980352508159
fBodyAcc-BandsEnergyOld-24	1576.1667554589353
fBodyAcc-BandsEnergyOld-25	-1442.2581649933982
fBodyAcc-BandsEnergyOld-26	-437.68982071960477
fBodyAcc-BandsEnergyOld-27	-108.83564894224185
fBodyAcc-BandsEnergyOld-28	-1277.6547967732831
fBodyAcc-BandsEnergyOld-29	-54.50395150974449
fBodyAcc-BandsEnergyOld-30	880.19440929483
fBodyAcc-BandsEnergyOld-31	482.32992685499494
fBodyAcc-BandsEnergyOld-32	801.1770669408551
fBodyAcc-BandsEnergyOld-33	-1563.164094794871
fBodyAcc-BandsEnergyOld-34	-1008.907548968696
fBodyAcc-BandsEnergyOld-35	-199.3560058616378
fBodyAcc-BandsEnergyOld-36	-60.955151944906035
fBodyAcc-BandsEnergyOld-37	-70.42570381170209
fBodyAcc-BandsEnergyOld-38	463.8916847554655
fBodyAcc-BandsEnergyOld-39	-1201.523772271068
fBodyAcc-BandsEnergyOld-40	-118.06827222261326
fBodyAcc-BandsEnergyOld-41	-79.47413099530773
fBodyAcc-BandsEnergyOld-42	2117.9202098831133
fBodyAccJerk-Mean-1	6107.550456261997
fBodyAccJerk-Mean-2	-7968.472349090264
fBodyAccJerk-Mean-3	-15319.447524746267
fBodyAccJerk-STD-1	3921.5716195493947
fBodyAccJerk-STD-2	-20266.506158874112
fBodyAccJerk-STD-3	29155.24539527151
fBodyAccJerk-Mad-1	5760.576481071189
fBodyAccJerk-Mad-2	-12643.644163440413

fBodyAccJerk-Mad-3	-1153859.657365762
fBodyAccJerk-Max-1	2493.989862379626
fBodyAccJerk-Max-2	-57078.11441279561
fBodyAccJerk-Max-3	12090.76098007945
fBodyAccJerk-Min-1	1834.7235646804684
fBodyAccJerk-Min-2	-13699.83035000131
fBodyAccJerk-Min-3	3290.452893918737
fBodyAccJerk-SMA-1	-268096.6939351895
fBodyAccJerk-Energy-1	872.7287418531888
fBodyAccJerk-Energy-2	1303.4022500738995
fBodyAccJerk-Energy-3	470.26853824146485
fBodyAccJerk-IQR-1	8500.548463366826
fBodyAccJerk-IQR-2	-5083.76444686708
fBodyAccJerk-IQR-3	-9767.578128853671
fBodyAccJerk-ropy-1	-4492.870427578424
fBodyAccJerk-ropy-1	-4254.993429674095
fBodyAccJerk-ropy-1	-3828.6422806381015
fBodyAccJerk-MaxInds-1	2121.528347861288
fBodyAccJerk-MaxInds-2	560.9615325999408
fBodyAccJerk-MaxInds-3	138.96931134645612
fBodyAccJerk-MeanFreq-1	449.5237224376145
fBodyAccJerk-MeanFreq-2	285.32310213455503
fBodyAccJerk-MeanFreq-3	719.8547586721802
fBodyAccJerk-Skewness-1	810.7479545832421
fBodyAccJerk-Kurtosis-1	477.4741991359334
fBodyAccJerk-Skewness-1	-1661.9972849753185
fBodyAccJerk-Kurtosis-1	-344.0521677455928
fBodyAccJerk-Skewness-1	-477.7862001925093
fBodyAccJerk-Kurtosis-1	-125.63987180582075
fBodyAccJerk-BandsEnergyOld-1	834.3815968785585
fBodyAccJerk-BandsEnergyOld-2	673.6229777025198
fBodyAccJerk-BandsEnergyOld-3	702.9870271609043
fBodyAccJerk-BandsEnergyOld-4	698.8485552556622
fBodyAccJerk-BandsEnergyOld-5	453.26833629725434
fBodyAccJerk-BandsEnergyOld-6	616.9519717954096
fBodyAccJerk-BandsEnergyOld-7	384.7080102169269
fBodyAccJerk-BandsEnergyOld-8	105.06557242231372
fBodyAccJerk-BandsEnergyOld-9	791.1960559131678
fBodyAccJerk-BandsEnergyOld-10	833.4982870675484
fBodyAccJerk-BandsEnergyOld-11	554.6589718356898
fBodyAccJerk-BandsEnergyOld-12	397.7110526077702
fBodyAccJerk-BandsEnergyOld-13	888.7055983696196
fBodyAccJerk-BandsEnergyOld-14	874.3027884867209
fBodyAccJerk-BandsEnergyOld-15	7039.609912692185
fBodyAccJerk-BandsEnergyOld-16	817.0371943721794
fBodyAccJerk-BandsEnergyOld-17	980.5845137377555
fBodyAccJerk-BandsEnergyOld-18	7472.214660481061
fBodyAccJerk-BandsEnergyOld-19	895.5759105014087

fBodyAccJerk-BandsEnergyOld-20	6328.08262205447
fBodyAccJerk-BandsEnergyOld-21	6180.140248045382
fBodyAccJerk-BandsEnergyOld-22	285.61524262241835
fBodyAccJerk-BandsEnergyOld-23	1254.2561232517057
fBodyAccJerk-BandsEnergyOld-24	1284.8238265577277
fBodyAccJerk-BandsEnergyOld-25	2396.238329817791
fBodyAccJerk-BandsEnergyOld-26	5262.721167234087
fBodyAccJerk-BandsEnergyOld-27	1187.3260031242826
fBodyAccJerk-BandsEnergyOld-28	2137.4153249206433
fBodyAccJerk-BandsEnergyOld-29	2854.3964371030024
fBodyAccJerk-BandsEnergyOld-30	617.2434632915314
fBodyAccJerk-BandsEnergyOld-31	369.9987400302095
fBodyAccJerk-BandsEnergyOld-32	376.2990556559942
fBodyAccJerk-BandsEnergyOld-33	312.30340162551005
fBodyAccJerk-BandsEnergyOld-34	563.4429538672708
fBodyAccJerk-BandsEnergyOld-35	462.98582408912495
fBodyAccJerk-BandsEnergyOld-36	188.75016852168199
fBodyAccJerk-BandsEnergyOld-37	895.9498990175307
fBodyAccJerk-BandsEnergyOld-38	335.0439851186153
fBodyAccJerk-BandsEnergyOld-39	407.773958554032
fBodyAccJerk-BandsEnergyOld-40	458.9002644192326
fBodyAccJerk-BandsEnergyOld-41	609.3304133942715
fBodyAccJerk-BandsEnergyOld-42	386.0035781434473
fBodyGyro-Mean-1	-1525.2771741672395
fBodyGyro-Mean-2	-1109.4767542303227
fBodyGyro-Mean-3	-677.6088967014183
fBodyGyro-STD-1	-1174.9363595063128
fBodyGyro-STD-2	-554.6296347026474
fBodyGyro-STD-3	-248.8638823688577
fBodyGyro-Mad-1	-1112.6918881436247
fBodyGyro-Mad-2	-737.2325727941429
fBodyGyro-Mad-3	-532.7540111688913
fBodyGyro-Max-1	-1550.8213680010062
fBodyGyro-Max-2	-386.3013898470172
fBodyGyro-Max-3	-178.67322092747963
fBodyGyro-Min-1	-138.55515522814292
fBodyGyro-Min-2	-417.6826625407308
fBodyGyro-Min-3	-95.05155229084143
fBodyGyro-SMA-1	-1018.198663946078
fBodyGyro-Energy-1	-345.6307571183578
fBodyGyro-Energy-2	-346.360746742872
fBodyGyro-Energy-3	-90.2044080436343
fBodyGyro-IQR-1	-4370.7439746376
fBodyGyro-IQR-2	-11716.6657262402
fBodyGyro-IQR-3	-5064.734798356074
fBodyGyro-ropy-1	-2624.0963755868506
fBodyGyro-ropy-1	-1691.0747729907034
fBodyGyro-ropy-1	-1971.8794975745345

```
fBodyGyro-MaxInds-1      95.71274810544858
fBodyGyro-MaxInds-2      417.90815727476445
fBodyGyro-MaxInds-3      151.91375348111987
fBodyGyro-MeanFreq-1      100.61243278505535
fBodyGyro-MeanFreq-2      78.88465237992774
fBodyGyro-MeanFreq-3      192.33637968155145
fBodyGyro-Skewness-1      -2276.445474345732
fBodyGyro-Kurtosis-1      -2969.6611429061745
fBodyGyro-Skewness-1      -562.1078424406046
fBodyGyro-Kurtosis-1      -280.6629867775174
fBodyGyro-Skewness-1      -104.536271761201
fBodyGyro-Kurtosis-1      -81.03842736962854
fBodyGyro-BandsEnergyOld-1      -197.5066098668359
fBodyGyro-BandsEnergyOld-2      2654.3729078274287
fBodyGyro-BandsEnergyOld-3      508.34818530459677
fBodyGyro-BandsEnergyOld-4      19489.596135648317
fBodyGyro-BandsEnergyOld-5      -5500.630661930682
fBodyGyro-BandsEnergyOld-6      -863.3682294261224
fBodyGyro-BandsEnergyOld-7      -215.0860733212317
fBodyGyro-BandsEnergyOld-8      -132.5171712590105
fBodyGyro-BandsEnergyOld-9      -325.44150579432676
fBodyGyro-BandsEnergyOld-10      646.1995342942887
fBodyGyro-BandsEnergyOld-11      -2142.0675947118093
fBodyGyro-BandsEnergyOld-12      -162.47086721334054
fBodyGyro-BandsEnergyOld-13      -339.9552445552855
fBodyGyro-BandsEnergyOld-14      -6417.135619295834
fBodyGyro-BandsEnergyOld-15      -144.40984707183426
fBodyGyro-BandsEnergyOld-16      514.9547374673974
fBodyGyro-BandsEnergyOld-17      266.98236311808375
fBodyGyro-BandsEnergyOld-18      306.063767254326
fBodyGyro-BandsEnergyOld-19      174.89714264715104
fBodyGyro-BandsEnergyOld-20      508.8384537503488
fBodyGyro-BandsEnergyOld-21      811.3755377085738
fBodyGyro-BandsEnergyOld-22      -174.3413991516826
fBodyGyro-BandsEnergyOld-23      -224.51310082778517
fBodyGyro-BandsEnergyOld-24      333.53259836618287
fBodyGyro-BandsEnergyOld-25      234.39587660573963
fBodyGyro-BandsEnergyOld-26      2124.5305546519876
fBodyGyro-BandsEnergyOld-27      -332.97546790857757
fBodyGyro-BandsEnergyOld-28      302.74393342290625
fBodyGyro-BandsEnergyOld-29      -57.67431853840122
fBodyGyro-BandsEnergyOld-30      590.8892585002591
fBodyGyro-BandsEnergyOld-31      456.02105754202887
fBodyGyro-BandsEnergyOld-32      613.9983809120955
fBodyGyro-BandsEnergyOld-33      4859.166168872157
fBodyGyro-BandsEnergyOld-34      -687.2391114917613
fBodyGyro-BandsEnergyOld-35      -264.8478016928956
fBodyGyro-BandsEnergyOld-36      -75.09527870571156
```

fBodyGyro-BandsEnergyOld-37 -76.02631738024274
fBodyGyro-BandsEnergyOld-38 636.0284364656416
fBodyGyro-BandsEnergyOld-39 -2630.2730929731006
fBodyGyro-BandsEnergyOld-40 -153.06525688256005
fBodyGyro-BandsEnergyOld-41 -85.98317940936617
fBodyGyro-BandsEnergyOld-42 915.4533485096505
fBodyAccMag-Mean-1 -874.301891550501
fBodyAccMag-STD-1 -210.62047934958076
fBodyAccMag-Mad-1 -573.9027132070845
fBodyAccMag-Max-1 -118.60149323951408
fBodyAccMag-Min-1 -105.73287149375513
fBodyAccMag-SMA-1 -874.301891550501
fBodyAccMag-Energy-1 -126.11464172472296
fBodyAccMag-IQR-1 -2629.098417980535
fBodyAccMag-ropy-1 -1912.6690245295547
fBodyAccMag-MaxInds-1 548.6795356893816
fBodyAccMag-MeanFreq-1 224.30343386287282
fBodyAccMag-Skewness-1 -79.29232817112496
fBodyAccMag-Kurtosis-1 -78.28059382904846
fBodyAccJerkMag-Mean-1 88816.03713869864
fBodyAccJerkMag-STD-1 37317.26422772581
fBodyAccJerkMag-Mad-1 -66347.4588864669
fBodyAccJerkMag-Max-1 8308.431989480534
fBodyAccJerkMag-Min-1 17628.694457059464
fBodyAccJerkMag-SMA-1 88816.03713869864
fBodyAccJerkMag-Energy-1 1103.7896543891547
fBodyAccJerkMag-IQR-1 130428.76171980033
fBodyAccJerkMag-ropy-1 -3007.386665462356
fBodyAccJerkMag-MaxInds-1 22.964717575150626
fBodyAccJerkMag-MeanFreq-1 467.66695650321446
fBodyAccJerkMag-Skewness-1 -39663.692904834534
fBodyAccJerkMag-Kurtosis-1 167258.23063528203
fBodyGyroMag-Mean-1 -830.8375165046043
fBodyGyroMag-STD-1 -387.4326273976131
fBodyGyroMag-Mad-1 -552.2702777150347
fBodyGyroMag-Max-1 -233.71955743889018
fBodyGyroMag-Min-1 -257.1173231461457
fBodyGyroMag-SMA-1 -830.8375165046043
fBodyGyroMag-Energy-1 -156.54683657887037
fBodyGyroMag-IQR-1 -1999.7005804673906
fBodyGyroMag-ropy-1 -2510.0111467520483
fBodyGyroMag-MaxInds-1 1889.8540799532482
fBodyGyroMag-MeanFreq-1 104.53173973330856
fBodyGyroMag-Skewness-1 -108.25175419920218
fBodyGyroMag-Kurtosis-1 -231.8572443500368
fBodyGyroJerkMag-Mean-1 -8017.2462080566265
fBodyGyroJerkMag-STD-1 -4992.158278216696
fBodyGyroJerkMag-Mad-1 -5069.190285151086

```
fBodyGyroJerkMag-Max-1      -5791.386679852823
fBodyGyroJerkMag-Min-1      -19712.7372617949
fBodyGyroJerkMag-SMA-1      -8017.2462080566265
fBodyGyroJerkMag-Energy-1      531.9859560099273
fBodyGyroJerkMag-IQR-1      -4301.684360390659
fBodyGyroJerkMag-ropy-1      -2231.230679521368
fBodyGyroJerkMag-MaxInds-1      17.161619750754426
fBodyGyroJerkMag-MeanFreq-1      85.99670325538354
fBodyGyroJerkMag-Skewness-1      -6083.2734712062465
fBodyGyroJerkMag-Kurtosis-1      -7167.424668325024
tBodyAcc-AngleWRTGravity-1      -1311.8698481587562
tBodyAccJerk-AngleWRTGravity-1      -1064.6343163304944
tBodyGyro-AngleWRTGravity-1      -25081.096544143064
tBodyGyroJerk-AngleWRTGravity-1      23254.443353950872
tXAxisAcc-AngleWRTGravity-1      12601.854756190784
tYAxisAcc-AngleWRTGravity-1      1777.836728337151
tZAxisAcc-AngleWRTGravity-1      7011.128404408679
```

12 Pearson's correlation

[43]:
```python
from scipy.stats import pearsonr
```

[44]:
```python
selectedFeatures_index = [41, 9, 52, 228, 202, 70, 433, 193, 123, 121, 243, 6,␣
↪83, 502, 541]
```

[60]:
```python
HighlyCorrelatedFeatures = None
HighCorr = float('-inf')
LeastCorrelatedFeatures = None
LeastCorr = float('inf')
for i in range(len(selectedFeatures_index)):
    for j in range(i+1,len(selectedFeatures_index)):
        corr = pearsonr(X_train[:,selectedFeatures_index[i]], X_train[:
↪,selectedFeatures_index[j]])[0]

        if HighCorr < np.abs(corr):
            HighCorr = np.abs(corr)
            HighlyCorrelatedFeatures = (features[selectedFeatures_index[i]],␣
↪features[selectedFeatures_index[j]], corr)

        if LeastCorr > np.abs(corr):
            LeastCorr = np.abs(corr)
            LeastCorrelatedFeatures = (features[selectedFeatures_index[i]],␣
↪features[selectedFeatures_index[j]], corr)

        print(features[selectedFeatures_index[i]], '    '␣
↪,features[selectedFeatures_index[j]], ' :   ', corr)
```

tGravityAcc-Mean-2 tBodyAcc-Max-1 : -0.4339063925060642
tGravityAcc-Mean-2 tGravityAcc-Min-1 : -0.7673838759819631
tGravityAcc-Mean-2 tBodyAccJerkMag-Mad-1 : -0.46242949843672077
tGravityAcc-Mean-2 tBodyAccMag-Mad-1 : -0.3146608120167613
tGravityAcc-Mean-2 tGravityAcc-ARCoeff-6 : -0.14394953210875003
tGravityAcc-Mean-2 fBodyGyro-Max-2 : -0.38334798517007795
tGravityAcc-Mean-2 tBodyGyroJerk-ARCoeff-9 : 0.4170869799924418
tGravityAcc-Mean-2 tBodyGyro-STD-1 : -0.42562833469359573
tGravityAcc-Mean-2 tBodyGyro-Mean-2 : -0.022768282353281204
tGravityAcc-Mean-2 tBodyGyroMag-Min-1 : -0.3189714567010238
tGravityAcc-Mean-2 tBodyAcc-Mad-1 : -0.4119864615889127
tGravityAcc-Mean-2 tBodyAccJerk-STD-1 : -0.47004043188283184
tGravityAcc-Mean-2 fBodyAccMag-Mean-1 : -0.4114576971618521
tGravityAcc-Mean-2 fBodyGyroJerkMag-Mean-1 : -0.41869788334286834
tBodyAcc-Max-1 tGravityAcc-Min-1 : 0.33016548378022403
tBodyAcc-Max-1 tBodyAccJerkMag-Mad-1 : 0.934334662223337
tBodyAcc-Max-1 tBodyAccMag-Mad-1 : 0.847373465986971
tBodyAcc-Max-1 tGravityAcc-ARCoeff-6 : -0.11152729714631202
tBodyAcc-Max-1 fBodyGyro-Max-2 : 0.692492076717723
tBodyAcc-Max-1 tBodyGyroJerk-ARCoeff-9 : -0.5734810560495016
tBodyAcc-Max-1 tBodyGyro-STD-1 : 0.8463957209982756
tBodyAcc-Max-1 tBodyGyro-Mean-2 : -0.00022302374161254973
tBodyAcc-Max-1 tBodyGyroMag-Min-1 : 0.6752647340098299
tBodyAcc-Max-1 tBodyAcc-Mad-1 : 0.9612319388514441
tBodyAcc-Max-1 tBodyAccJerk-STD-1 : 0.948806219726004
tBodyAcc-Max-1 fBodyAccMag-Mean-1 : 0.9463883098988615
tBodyAcc-Max-1 fBodyGyroJerkMag-Mean-1 : 0.8298290491281353
tGravityAcc-Min-1 tBodyAccJerkMag-Mad-1 : 0.3721461278590039
tGravityAcc-Min-1 tBodyAccMag-Mad-1 : 0.1983680166049218
tGravityAcc-Min-1 tGravityAcc-ARCoeff-6 : 0.058312941559624666
tGravityAcc-Min-1 fBodyGyro-Max-2 : 0.25295028301972283
tGravityAcc-Min-1 tBodyGyroJerk-ARCoeff-9 : -0.3372538725978167
tGravityAcc-Min-1 tBodyGyro-STD-1 : 0.31469308694858206
tGravityAcc-Min-1 tBodyGyro-Mean-2 : 0.08337292645285002
tGravityAcc-Min-1 tBodyGyroMag-Min-1 : 0.21249704340538816
tGravityAcc-Min-1 tBodyAcc-Mad-1 : 0.30518315779427896
tGravityAcc-Min-1 tBodyAccJerk-STD-1 : 0.3775151303991036
tGravityAcc-Min-1 fBodyAccMag-Mean-1 : 0.30121058888887575
tGravityAcc-Min-1 fBodyGyroJerkMag-Mean-1 : 0.31966253401505645
tBodyAccJerkMag-Mad-1 tBodyAccMag-Mad-1 : 0.798465783353562
tBodyAccJerkMag-Mad-1 tGravityAcc-ARCoeff-6 : -0.21951552062228694
tBodyAccJerkMag-Mad-1 fBodyGyro-Max-2 : 0.6834192571126508
tBodyAccJerkMag-Mad-1 tBodyGyroJerk-ARCoeff-9 : -0.55175860641473
tBodyAccJerkMag-Mad-1 tBodyGyro-STD-1 : 0.849216232540462
tBodyAccJerkMag-Mad-1 tBodyGyro-Mean-2 : 0.03202892622338806
tBodyAccJerkMag-Mad-1 tBodyGyroMag-Min-1 : 0.6661693976923173
tBodyAccJerkMag-Mad-1 tBodyAcc-Mad-1 : 0.9327245393008514
tBodyAccJerkMag-Mad-1 tBodyAccJerk-STD-1 : 0.9842726911010856

```
tBodyAccJerkMag-Mad-1     fBodyAccMag-Mean-1  :   0.940355954782806
tBodyAccJerkMag-Mad-1     fBodyGyroJerkMag-Mean-1  :   0.9196779723993647
tBodyAccMag-Mad-1     tGravityAcc-ARCoeff-6  :   0.09086569406882779
tBodyAccMag-Mad-1     fBodyGyro-Max-2  :   0.7104611718968642
tBodyAccMag-Mad-1     tBodyGyroJerk-ARCoeff-9  :   -0.5883207204408946
tBodyAccMag-Mad-1     tBodyGyro-STD-1  :   0.8173316426905883
tBodyAccMag-Mad-1     tBodyGyro-Mean-2  :   0.016699455756098727
tBodyAccMag-Mad-1     tBodyGyroMag-Min-1  :   0.7010301222497253
tBodyAccMag-Mad-1     tBodyAcc-Mad-1  :   0.8851860715579807
tBodyAccMag-Mad-1     tBodyAccJerk-STD-1  :   0.7914859582786847
tBodyAccMag-Mad-1     fBodyAccMag-Mean-1  :   0.9375188603773023
tBodyAccMag-Mad-1     fBodyGyroJerkMag-Mean-1  :   0.7377313834859738
tGravityAcc-ARCoeff-6     fBodyGyro-Max-2  :   0.040409840725956415
tGravityAcc-ARCoeff-6     tBodyGyroJerk-ARCoeff-9  :   -0.17699950558576885
tGravityAcc-ARCoeff-6     tBodyGyro-STD-1  :   -0.01958425603149138
tGravityAcc-ARCoeff-6     tBodyGyro-Mean-2  :   -0.007524135934941744
tGravityAcc-ARCoeff-6     tBodyGyroMag-Min-1  :   0.11315520298186373
tGravityAcc-ARCoeff-6     tBodyAcc-Mad-1  :   -0.0972447995510616
tGravityAcc-ARCoeff-6     tBodyAccJerk-STD-1  :   -0.1974903841607408
tGravityAcc-ARCoeff-6     fBodyAccMag-Mean-1  :   -0.04989556477364504
tGravityAcc-ARCoeff-6     fBodyGyroJerkMag-Mean-1  :   -0.20237226157587868
fBodyGyro-Max-2     tBodyGyroJerk-ARCoeff-9  :   -0.571484991352176
fBodyGyro-Max-2     tBodyGyro-STD-1  :   0.7263607703178482
fBodyGyro-Max-2     tBodyGyro-Mean-2  :   0.02003705779900134
fBodyGyro-Max-2     tBodyGyroMag-Min-1  :   0.6738728506079759
fBodyGyro-Max-2     tBodyAcc-Mad-1  :   0.6977618716608691
fBodyGyro-Max-2     tBodyAccJerk-STD-1  :   0.6857192222438688
fBodyGyro-Max-2     fBodyAccMag-Mean-1  :   0.7284115395841129
fBodyGyro-Max-2     fBodyGyroJerkMag-Mean-1  :   0.6929537739391008
tBodyGyroJerk-ARCoeff-9     tBodyGyro-STD-1  :   -0.6123740473187813
tBodyGyroJerk-ARCoeff-9     tBodyGyro-Mean-2  :   -0.025394345518728347
tBodyGyroJerk-ARCoeff-9     tBodyGyroMag-Min-1  :   -0.5379136662226554
tBodyGyroJerk-ARCoeff-9     tBodyAcc-Mad-1  :   -0.5774256529151484
tBodyGyroJerk-ARCoeff-9     tBodyAccJerk-STD-1  :   -0.5456528421337663
tBodyGyroJerk-ARCoeff-9     fBodyAccMag-Mean-1  :   -0.5941400383690237
tBodyGyroJerk-ARCoeff-9     fBodyGyroJerkMag-Mean-1  :   -0.5003842750556026
tBodyGyro-STD-1     tBodyGyro-Mean-2  :   0.04910958301103918
tBodyGyro-STD-1     tBodyGyroMag-Min-1  :   0.7178818730196939
tBodyGyro-STD-1     tBodyAcc-Mad-1  :   0.8613180534373523
tBodyGyro-STD-1     tBodyAccJerk-STD-1  :   0.8471310744764909
tBodyGyro-STD-1     fBodyAccMag-Mean-1  :   0.8766515783759751
tBodyGyro-STD-1     fBodyGyroJerkMag-Mean-1  :   0.7856355033306411
tBodyGyro-Mean-2     tBodyGyroMag-Min-1  :   0.11529655228948611
tBodyGyro-Mean-2     tBodyAcc-Mad-1  :   0.0146560540492784
tBodyGyro-Mean-2     tBodyAccJerk-STD-1  :   0.022857628356471246
tBodyGyro-Mean-2     fBodyAccMag-Mean-1  :   0.021315113858215568
tBodyGyro-Mean-2     fBodyGyroJerkMag-Mean-1  :   0.03224238110956783
tBodyGyroMag-Min-1     tBodyAcc-Mad-1  :   0.7050423465716047
```

```
tBodyGyroMag-Min-1     tBodyAccJerk-STD-1   :   0.6704952516197735
tBodyGyroMag-Min-1     fBodyAccMag-Mean-1   :   0.724513618483428
tBodyGyroMag-Min-1     fBodyGyroJerkMag-Mean-1   :   0.6236099865867623
tBodyAcc-Mad-1     tBodyAccJerk-STD-1   :   0.9485984947024525
tBodyAcc-Mad-1     fBodyAccMag-Mean-1   :   0.9575961857034766
tBodyAcc-Mad-1     fBodyGyroJerkMag-Mean-1   :   0.826473148320449
tBodyAccJerk-STD-1     fBodyAccMag-Mean-1   :   0.9344479751255199
tBodyAccJerk-STD-1     fBodyGyroJerkMag-Mean-1   :   0.8891614311511125
fBodyAccMag-Mean-1     fBodyGyroJerkMag-Mean-1   :   0.8700527161614238
```

[61]:
```
print('Highly Correlated features - ', HighlyCorrelatedFeatures)
print('Least Correlated features - ', LeastCorrelatedFeatures)
```

```
Highly Correlated features -  ('tBodyAccJerkMag-Mad-1', 'tBodyAccJerk-STD-1',
0.9842726911010856)
Least Correlated features -  ('tBodyAcc-Max-1', 'tBodyGyro-Mean-2',
-0.00022302374161254973)
```

13 Training Models

14 LogisticRegression

[17]:
```
from sklearn.linear_model import LogisticRegression
```

[52]:
```
clf = LogisticRegression()
clf.fit(X_train, y_train+1)

predictionsVal = clf.predict(X_val)
predictionsTest = clf.predict(X_test)

print('Accuracy Val = ', np.sum(predictionsVal == (y_val+1)) / y_val.shape[0] )
print('Accuracy Test = ', np.sum(predictionsTest == (y_test+1)) / y_test.
 ↪shape[0] )
```

```
Accuracy Val =  0.9679780420860018
Accuracy Test =  0.9679780420860018
```

15 Neural Network

[73]:
```
def Generate_Model_NN1():
    m = models.Sequential()
    m.add(layers.Dense( 50 , input_shape = (561,), activation='relu'))
    m.add(layers.Dropout(0.3))
    m.add(layers.Dense( 50 , activation='relu'))
    m.add(layers.Dropout(0.3))
    m.add(layers.Dense( 50 , activation='relu'))
```

```
    m.add(layers.Dropout(0.3))
    m.add(layers.Dense( 12 , activation='sigmoid'))

    m.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
    return m
```

[85]:
```
model_1 = Generate_Model_NN1()
model_1.summary()
history = model_1.fit(X_train, y_train_categorical,
                      batch_size=100,
                      epochs=30,
                      validation_data=(X_val, y_val_categorical))
```

```
Model: "sequential_13"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_59 (Dense)             (None, 50)                28100

_________________________________________________________________
dropout_47 (Dropout)         (None, 50)                0

_________________________________________________________________
dense_60 (Dense)             (None, 50)                2550

_________________________________________________________________
dropout_48 (Dropout)         (None, 50)                0

_________________________________________________________________
dense_61 (Dense)             (None, 50)                2550

_________________________________________________________________
dropout_49 (Dropout)         (None, 50)                0

_________________________________________________________________
dense_62 (Dense)             (None, 12)                612
=================================================================
Total params: 33,812
Trainable params: 33,812
Non-trainable params: 0

_________________________________________________________________
Train on 8742 samples, validate on 1093 samples
Epoch 1/30
8742/8742 [==============================] - 0s 55us/step - loss: 1.6532 -
accuracy: 0.2880 - val_loss: 1.2092 - val_accuracy: 0.3458
Epoch 2/30
8742/8742 [==============================] - 0s 37us/step - loss: 1.2236 -
accuracy: 0.3334 - val_loss: 1.1773 - val_accuracy: 0.5425
Epoch 3/30
8742/8742 [==============================] - 0s 41us/step - loss: 1.1832 -
accuracy: 0.3423 - val_loss: 1.1513 - val_accuracy: 0.3477
Epoch 4/30
```

```
8742/8742 [==============================] - 0s 48us/step - loss: 1.1596 -
accuracy: 0.3593 - val_loss: 1.1216 - val_accuracy: 0.4163
Epoch 5/30
8742/8742 [==============================] - 0s 47us/step - loss: 1.1207 -
accuracy: 0.4085 - val_loss: 1.0215 - val_accuracy: 0.5242
Epoch 6/30
8742/8742 [==============================] - 0s 54us/step - loss: 1.0212 -
accuracy: 0.4484 - val_loss: 0.9136 - val_accuracy: 0.4309
Epoch 7/30
8742/8742 [==============================] - 0s 45us/step - loss: 0.9398 -
accuracy: 0.4986 - val_loss: 0.8666 - val_accuracy: 0.6185
Epoch 8/30
8742/8742 [==============================] - 0s 42us/step - loss: 0.8186 -
accuracy: 0.5668 - val_loss: 0.7191 - val_accuracy: 0.6359
Epoch 9/30
8742/8742 [==============================] - 0s 41us/step - loss: 0.7590 -
accuracy: 0.6004 - val_loss: 0.6914 - val_accuracy: 0.5773
Epoch 10/30
8742/8742 [==============================] - 0s 40us/step - loss: 0.7252 -
accuracy: 0.6080 - val_loss: 0.6760 - val_accuracy: 0.6542
Epoch 11/30
8742/8742 [==============================] - 0s 39us/step - loss: 0.7122 -
accuracy: 0.6201 - val_loss: 0.6824 - val_accuracy: 0.6267
Epoch 12/30
8742/8742 [==============================] - 0s 38us/step - loss: 0.6959 -
accuracy: 0.6227 - val_loss: 0.6652 - val_accuracy: 0.6304
Epoch 13/30
8742/8742 [==============================] - 0s 39us/step - loss: 0.6538 -
accuracy: 0.6627 - val_loss: 0.3798 - val_accuracy: 0.8289
Epoch 14/30
8742/8742 [==============================] - 0s 34us/step - loss: 0.4442 -
accuracy: 0.7813 - val_loss: 0.3210 - val_accuracy: 0.8161
Epoch 15/30
8742/8742 [==============================] - 0s 33us/step - loss: 0.3960 -
accuracy: 0.8024 - val_loss: 0.3133 - val_accuracy: 0.8161
Epoch 16/30
8742/8742 [==============================] - 0s 33us/step - loss: 0.3542 -
accuracy: 0.8253 - val_loss: 0.2910 - val_accuracy: 0.8545
Epoch 17/30
8742/8742 [==============================] - 0s 33us/step - loss: 0.3470 -
accuracy: 0.8463 - val_loss: 0.2765 - val_accuracy: 0.8774
Epoch 18/30
8742/8742 [==============================] - 0s 33us/step - loss: 0.2984 -
accuracy: 0.8735 - val_loss: 0.2342 - val_accuracy: 0.8975
Epoch 19/30
8742/8742 [==============================] - 0s 33us/step - loss: 0.2813 -
accuracy: 0.8816 - val_loss: 0.2347 - val_accuracy: 0.8957
Epoch 20/30
```

```
8742/8742 [==============================] - 0s 33us/step - loss: 0.2642 -
accuracy: 0.8934 - val_loss: 0.1733 - val_accuracy: 0.9222
Epoch 21/30
8742/8742 [==============================] - 0s 33us/step - loss: 0.2415 -
accuracy: 0.9038 - val_loss: 0.1840 - val_accuracy: 0.9222
Epoch 22/30
8742/8742 [==============================] - 0s 33us/step - loss: 0.2299 -
accuracy: 0.9068 - val_loss: 0.1711 - val_accuracy: 0.9259
Epoch 23/30
8742/8742 [==============================] - 0s 38us/step - loss: 0.2210 -
accuracy: 0.9156 - val_loss: 0.1627 - val_accuracy: 0.9286
Epoch 24/30
8742/8742 [==============================] - 0s 31us/step - loss: 0.2088 -
accuracy: 0.9198 - val_loss: 0.1563 - val_accuracy: 0.9305
Epoch 25/30
8742/8742 [==============================] - 0s 26us/step - loss: 0.1947 -
accuracy: 0.9238 - val_loss: 0.1578 - val_accuracy: 0.9323
Epoch 26/30
8742/8742 [==============================] - 0s 24us/step - loss: 0.1899 -
accuracy: 0.9282 - val_loss: 0.1562 - val_accuracy: 0.9387
Epoch 27/30
8742/8742 [==============================] - 0s 24us/step - loss: 0.1840 -
accuracy: 0.9251 - val_loss: 0.1492 - val_accuracy: 0.9369
Epoch 28/30
8742/8742 [==============================] - 0s 25us/step - loss: 0.1726 -
accuracy: 0.9329 - val_loss: 0.1372 - val_accuracy: 0.9424
Epoch 29/30
8742/8742 [==============================] - 0s 27us/step - loss: 0.1885 -
accuracy: 0.9256 - val_loss: 0.1474 - val_accuracy: 0.9396
Epoch 30/30
8742/8742 [==============================] - 0s 25us/step - loss: 0.1790 -
accuracy: 0.9295 - val_loss: 0.1391 - val_accuracy: 0.9369
```

[88]:

```
loss, test_acc = model_1.evaluate(X_test, y_test_categorical)
print('test_acc: ', test_acc)

loss, val_acc = model_1.evaluate(X_val, y_val_categorical)
print('val_acc: ', val_acc)

loss, train_acc = model_1.evaluate(X_train, y_train_categorical)
print('train_acc: ', train_acc)
```

```
1093/1093 [==============================] - 0s 30us/step
test_acc:  0.9332113265991211
1093/1093 [==============================] - 0s 31us/step
val_acc:  0.9368709921836853
8742/8742 [==============================] - 0s 22us/step
train_acc:  0.9604209661483765
```

[90]:
```
#model_1.save('model_1_Test_93_3.h5')
```

16 (f) Feature selection.

[18]:
```
def createDataSetWithSelectedFeatures(FullDataSet, FeaturesList):

    numFeatures = FullDataSet.shape[1]
    NewReducedDataSet = np.array( [ FullDataSet[:,FeaturesList[0]] ] ).T

    for i in range(len(FeaturesList)):

        if i == 0:
            continue

        if FeaturesList[i] < 0 or FeaturesList[i] > numFeatures-1:
            continue

        featureToSelect = FeaturesList[i]
        NewReducedDataSet = np.hstack( (NewReducedDataSet , np.array( [␣
↪FullDataSet[:,FeaturesList[i]] ] ).T) )

    return NewReducedDataSet
```

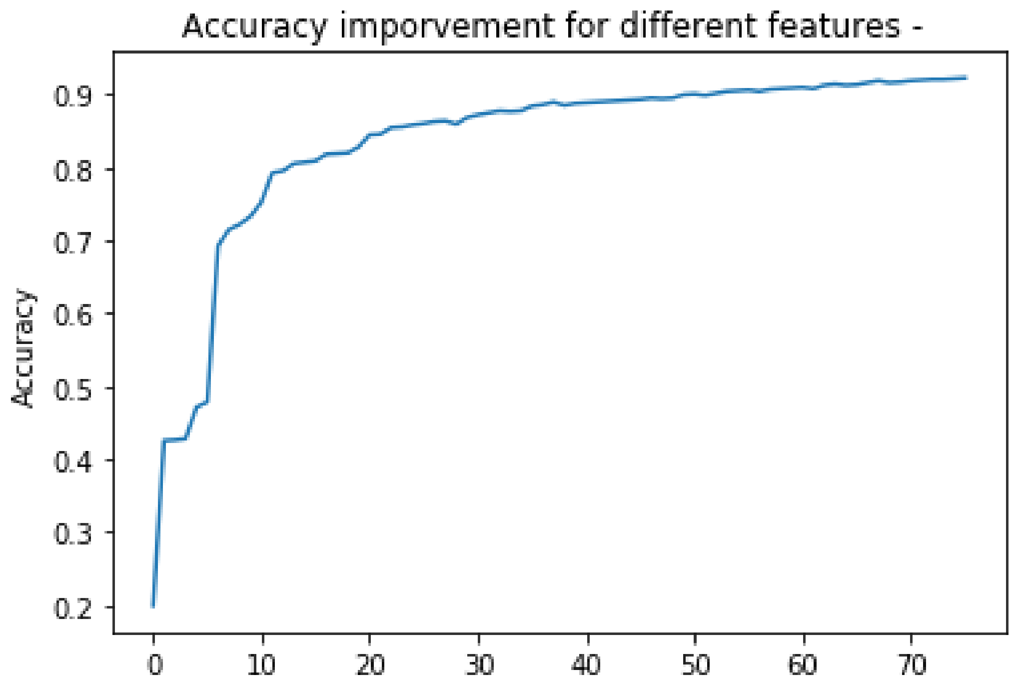
[21]:
```
NumFeaturesToSelect = 15
TotalNumFeatures = X_train.shape[1]
#TotalNumFeatures = 50
IncludedFeatures_X = np.zeros(shape=(X_train.shape[0], NumFeaturesToSelect))
IncludedFeatures_list = [-1 for i in range(NumFeaturesToSelect)]


PreviousBestValAcc = 0
EndSelection = False
Accuracies = []
for i in range(NumFeaturesToSelect):

    print('\nSelecting feature ', i+1)

    bestFeatureIndex = 0
    bestAccuracy = 0
    for feature_index in range(TotalNumFeatures):

        if feature_index in IncludedFeatures_list:
            continue

        featureX = np.array( [X_train[:,feature_index]] )
        IncludedFeatures_X[:,i] = featureX
```

```
        IncludedFeatures_list[i] = feature_index

        clfX = LogisticRegression()
        clfX.fit(IncludedFeatures_X[:,:i+1], (y_train+1))

        reducedValidationDataset = createDataSetWithSelectedFeatures(X_val,␣
↪IncludedFeatures_list)
        predictionsVal = clfX.predict(reducedValidationDataset)
        val_acc = np.sum(predictionsVal == (y_val+1)) / y_val.shape[0]

        #print(feature_index, ' Accuracy Val = ', val_acc )

        if val_acc > bestAccuracy:
            bestAccuracy = val_acc
            Accuracies.append(bestAccuracy)
            bestFeatureIndex = feature_index

    print('Best Validation accuracy:', bestAccuracy)
    if bestAccuracy > PreviousBestValAcc:
        PreviousBestValAcc = bestAccuracy
    else:
        break

    IncludedFeatures_list[i] = bestFeatureIndex
    IncludedFeatures_X[:,i] = np.array( [X_train[:,bestFeatureIndex]] )
    print('Selected Features - ', [x for x in IncludedFeatures_list if x != -1]␣
↪)


plt.figure()
plt.title('Accuracy imporvement for different features - ')
plt.ylabel('Accuracy')
plt.plot(Accuracies)
print(IncludedFeatures_list)
```

```
Selecting feature  1
Best Validation accuracy: 0.47118023787740165
Selected Features -  [41]

Selecting feature  2
Best Validation accuracy: 0.7145471180237878
Selected Features -  [41, 9]

Selecting feature  3
Best Validation accuracy: 0.7950594693504117
```

```
Selected Features -  [41, 9, 52]

Selecting feature  4
Best Validation accuracy: 0.8453796889295517
Selected Features -  [41, 9, 52, 228]

Selecting feature  5
Best Validation accuracy: 0.8636779505946935
Selected Features -  [41, 9, 52, 228, 202]

Selecting feature  6
Best Validation accuracy: 0.8774016468435498
Selected Features -  [41, 9, 52, 228, 202, 70]

Selecting feature  7
Best Validation accuracy: 0.889295516925892
Selected Features -  [41, 9, 52, 228, 202, 70, 433]

Selecting feature  8
Best Validation accuracy: 0.8947849954254345
Selected Features -  [41, 9, 52, 228, 202, 70, 433, 193]

Selecting feature  9
Best Validation accuracy: 0.9002744739249772
Selected Features -  [41, 9, 52, 228, 202, 70, 433, 193, 123]

Selecting feature  10
Best Validation accuracy: 0.9057639524245197
Selected Features -  [41, 9, 52, 228, 202, 70, 433, 193, 123, 121]

Selecting feature  11
Best Validation accuracy: 0.909423604757548
Selected Features -  [41, 9, 52, 228, 202, 70, 433, 193, 123, 121, 243]

Selecting feature  12
Best Validation accuracy: 0.9139981701738334
Selected Features -  [41, 9, 52, 228, 202, 70, 433, 193, 123, 121, 243, 6]

Selecting feature  13
Best Validation accuracy: 0.918572735590119
Selected Features -  [41, 9, 52, 228, 202, 70, 433, 193, 123, 121, 243, 6, 83]

Selecting feature  14
Best Validation accuracy: 0.9204025617566332
Selected Features -  [41, 9, 52, 228, 202, 70, 433, 193, 123, 121, 243, 6, 83,
502]

Selecting feature  15
```

```
Best Validation accuracy: 0.9222323879231473
Selected Features -  [41, 9, 52, 228, 202, 70, 433, 193, 123, 121, 243, 6, 83,
502, 541]
[41, 9, 52, 228, 202, 70, 433, 193, 123, 121, 243, 6, 83, 502, 541]
```



[30]:
```python
print('List of selected features - \n')
for i in IncludedFeatures_list:
    print(features[i])
```

```
List of selected features -

tGravityAcc-Mean-2
tBodyAcc-Max-1
tGravityAcc-Min-1
tBodyAccJerkMag-Mad-1
tBodyAccMag-Mad-1
tGravityAcc-ARCoeff-6
fBodyGyro-Max-2
tBodyGyroJerk-ARCoeff-9
tBodyGyro-STD-1
tBodyGyro-Mean-2
tBodyGyroMag-Min-1
tBodyAcc-Mad-1
tBodyAccJerk-STD-1
fBodyAccMag-Mean-1
```

fBodyGyroJerkMag-Mean-1

[23]:
```
X_train_reduced_featuresSelected = createDataSetWithSelectedFeatures(X_train,␣
 ↪IncludedFeatures_list)
X_test_reduced_featuresSelected = createDataSetWithSelectedFeatures(X_test,␣
 ↪IncludedFeatures_list)
```

[24]:
```
clfX = LogisticRegression()
clfX.fit(X_train_reduced_featuresSelected, (y_train+1))
```

[24]:
```
LogisticRegression(C=1.0, 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)
```

[26]:
```
predictionsTest = clfX.predict(X_test_reduced_featuresSelected)
test_acc = np.sum(predictionsTest == (y_test+1)) / y_test.shape[0]
print('Accuracy on selected features Test dataset - ', test_acc)
```

```
Accuracy on selected features Test dataset -  0.8728270814272644
```

17 (g) Feature transformation (PCA).

[77]:
```
from numpy import linalg as LA
```

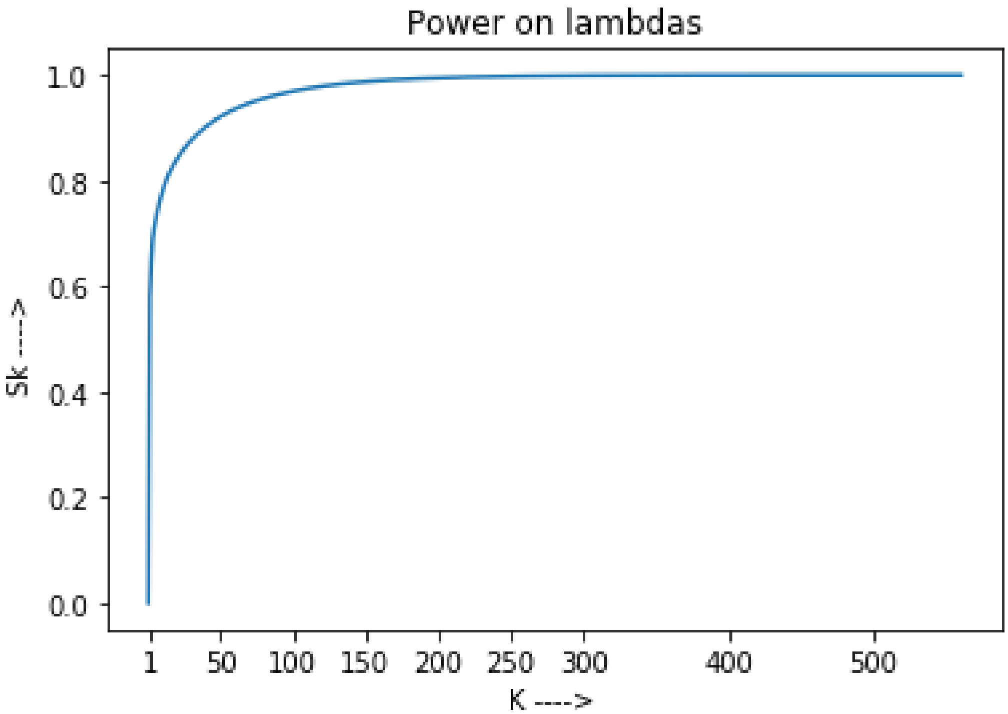
[127]:
```
def meanNormalize_features(Mat):

    NormalizedMat = np.array( [Mat[:,0] - np.mean(Mat[:,0])] ).T

    colsMax = Mat.shape[1]

    for i in range(colsMax):

        if i == 0:
            continue

        NormalizedMat = np.hstack( (NormalizedMat, np.array( [Mat[:,i] - np.
 ↪mean(Mat[:,i])] ).T ) )

    return NormalizedMat

def PCA_ML633(MatIn, K, plot='on'):

    Mat = meanNormalize_features(MatIn)
```

```
    CovMat = np.matmul(Mat.T , Mat)
    D = CovMat.shape[0]
    CovMat = CovMat / (D-1)

    Lambdas, vectors = LA.eigh(CovMat)

    if plot == 'on':

        lambda_sum = np.sum(Lambdas)
        plot_y = []
        for i in range(D):
            Vec_index = D - i
            First_i_lambda_sum = np.sum(Lambdas[D-i:D])
            plot_y.append(First_i_lambda_sum / lambda_sum)

        plt.title('Power on lambdas')
        plt.xlabel('K ----> ')
        plt.ylabel('Sk ----> ')
        plt.xticks([1,50,100,150,200,250,300,400,500])
        plt.plot(plot_y)

    if K > D:
        print('Error K > D')
        return

    U = np.array( [vectors[:,D-1]] ).T
    for i in range(K):

        if i == 0:
            continue

        Vec_index = D - i - 1
        U = np.hstack( (U, np.array( [vectors[:,Vec_index]] ).T) )

    return U , np.matmul(Mat, U)
```

[128]:
```
U , X_PCA = PCA_ML633(X , 100)
X_PCA_Train = X_PCA[:8742,:]
X_PCA_val = X_PCA[8742:9835,:]
X_PCA_test = X_PCA[9835:,:]

print('Shape of Train set ', X_PCA_Train.shape)
print('Shape of Test set ', X_PCA_test.shape)
print('Shape of Val set ', X_PCA_val.shape)
```

```
Shape of Train set  (8742, 100)
Shape of Test set  (1093, 100)
```

```
Shape of Val set  (1093, 100)
```



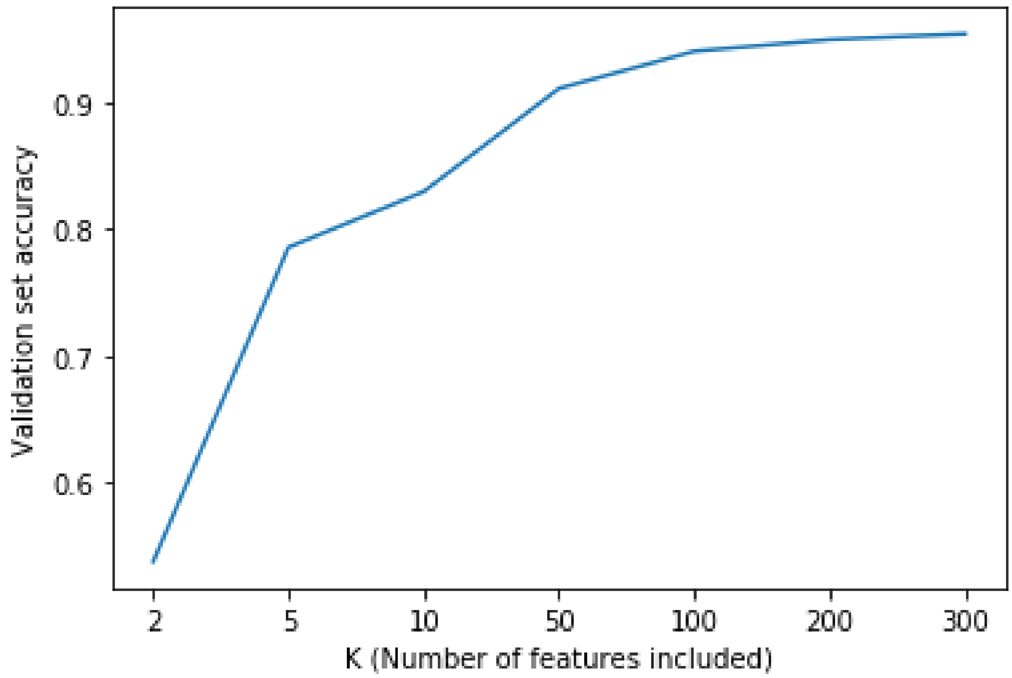
[96]:
```python
plotAcc = []
NumFeaturesInPCA = [2, 5, 10, 50, 100, 200, 300]
for NumFeaturesInPCA in NumFeaturesInPCA:
    print('K = ', NumFeaturesInPCA)
    U , X_PCA = PCA_ML633(X , NumFeaturesInPCA, plot='off')

    X_PCA_Train = X_PCA[:8742,:]
    y_PCA_Train = y[:8742]
    X_PCA_val = X_PCA[8742:9835,:]
    y_PCA_val = y[8742:9835]
    X_PCA_test = X_PCA[9835:,:]
    y_PCA_test = y[9835:]

    clf = LogisticRegression()
    clf.fit(X_PCA_Train, y_PCA_Train+1)

    predictionsTrain = clf.predict(X_PCA_Train)
    predictionsVal = clf.predict(X_PCA_val)
    predictionsTest = clf.predict(X_PCA_test)

    accVal = np.sum(predictionsVal == (y_PCA_val+1)) / y_PCA_val.shape[0]
    print('Accuracy Val = ', accVal )
```

```
    print('Accuracy Test = ', np.sum(predictionsTest == (y_PCA_test+1)) /
 ↪y_PCA_test.shape[0] )
    print('Accuracy Train = ', np.sum(predictionsTrain == (y_PCA_Train+1)) /
 ↪y_PCA_Train.shape[0] )
    plotAcc.append(accVal)
```

```
K =  2
Accuracy Val =  0.5379688929551693
Accuracy Test =  0.5041171088746569
Accuracy Train =  0.5512468542667581
K =  5
Accuracy Val =  0.7859103385178408
Accuracy Test =  0.7301006404391582
Accuracy Train =  0.7862045298558682
K =  10
Accuracy Val =  0.8298261665141812
Accuracy Test =  0.848124428179323
Accuracy Train =  0.8523221230839625
K =  50
Accuracy Val =  0.9112534309240622
Accuracy Test =  0.9313815187557182
Accuracy Train =  0.945664607641272
K =  100
Accuracy Val =  0.9405306495882891
Accuracy Test =  0.958828911253431
Accuracy Train =  0.9735758407687028
K =  200
Accuracy Val =  0.94967978042086
Accuracy Test =  0.9624885635864593
Accuracy Train =  0.9871882864333105
K =  300
Accuracy Val =  0.9542543458371455
Accuracy Test =  0.9643183897529735
Accuracy Train =  0.9881034088309312
```

[124]:
```
plt.figure()
plt.ylabel('Validation set accuracy')
plt.xlabel('K (Number of features included)')
locs, labels = plt.xticks()
NumFeaturesInPCA = [2, 5, 10, 50, 100, 200, 300]
plt.xticks(np.arange(len(NumFeaturesInPCA)), NumFeaturesInPCA)
plt.plot(plotAcc)
```

[124]:
```
[<matplotlib.lines.Line2D at 0x1ef8d73f0c8>]
```



[117]:
```
U , X_PCA = PCA_ML633(X , 100, plot='off')
X_PCA_Train = X_PCA[:8742,:]
X_PCA_val = X_PCA[8742:9835,:]
X_PCA_test = X_PCA[9835:,:]

y_cat = to_categorical(y)
y_cat_PCA_Train = y_cat[:8742,:]
y_cat_PCA_val = y_cat[8742:9835,:]
y_cat_PCA_test = y_cat[9835:,:]
```

[118]:
```
def Generate_Model_NN_PCA_DataSet():
    m = models.Sequential()
    m.add(layers.Dense( 30 , input_shape = (100,), activation='relu'))
    m.add(layers.Dropout(0.3))
    m.add(layers.Dense( 30 , activation='relu'))
    m.add(layers.Dropout(0.3))
    m.add(layers.Dense( 30 , activation='relu'))
    m.add(layers.Dropout(0.3))
    m.add(layers.Dense( 12 , activation='sigmoid'))

    m.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
    return m
```

[119]:
```
model_1_PCA_Trained = Generate_Model_NN_PCA_DataSet()
model_1_PCA_Trained.summary()
history = model_1_PCA_Trained.fit(X_PCA_Train, y_cat_PCA_Train,
                    batch_size=100,
                    epochs=20,
                    validation_data=(X_PCA_val, y_cat_PCA_val))
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 30)                3030
_________________________________________________________________
dropout_4 (Dropout)          (None, 30)                0
_________________________________________________________________
dense_6 (Dense)              (None, 30)                930
_________________________________________________________________
dropout_5 (Dropout)          (None, 30)                0
_________________________________________________________________
dense_7 (Dense)              (None, 30)                930
_________________________________________________________________
dropout_6 (Dropout)          (None, 30)                0
_________________________________________________________________
dense_8 (Dense)              (None, 12)                372
=================================================================
Total params: 5,262
Trainable params: 5,262
Non-trainable params: 0
_________________________________________________________________
Train on 8742 samples, validate on 1093 samples
Epoch 1/20
8742/8742 [==============================] - 0s 39us/step - loss: 1.9908 -
accuracy: 0.2741 - val_loss: 1.3790 - val_accuracy: 0.4437
Epoch 2/20
8742/8742 [==============================] - 0s 18us/step - loss: 1.2402 -
accuracy: 0.4307 - val_loss: 0.8685 - val_accuracy: 0.5087
Epoch 3/20
8742/8742 [==============================] - 0s 22us/step - loss: 0.9587 -
accuracy: 0.5189 - val_loss: 0.8247 - val_accuracy: 0.5023
Epoch 4/20
8742/8742 [==============================] - 0s 24us/step - loss: 0.8762 -
accuracy: 0.5320 - val_loss: 0.7723 - val_accuracy: 0.5361
Epoch 5/20
8742/8742 [==============================] - 0s 22us/step - loss: 0.7827 -
accuracy: 0.5854 - val_loss: 0.6618 - val_accuracy: 0.6340
Epoch 6/20
8742/8742 [==============================] - 0s 22us/step - loss: 0.6736 -
```

```
accuracy: 0.6576 - val_loss: 0.5769 - val_accuracy: 0.7612
Epoch 7/20
8742/8742 [==============================] - 0s 21us/step - loss: 0.5986 -
accuracy: 0.6939 - val_loss: 0.5017 - val_accuracy: 0.7338
Epoch 8/20
8742/8742 [==============================] - 0s 22us/step - loss: 0.5367 -
accuracy: 0.7210 - val_loss: 0.4532 - val_accuracy: 0.7539
Epoch 9/20
8742/8742 [==============================] - 0s 21us/step - loss: 0.4904 -
accuracy: 0.7407 - val_loss: 0.4313 - val_accuracy: 0.7658
Epoch 10/20
8742/8742 [==============================] - 0s 21us/step - loss: 0.4533 -
accuracy: 0.7552 - val_loss: 0.4120 - val_accuracy: 0.7649
Epoch 11/20
8742/8742 [==============================] - 0s 21us/step - loss: 0.4280 -
accuracy: 0.7679 - val_loss: 0.4087 - val_accuracy: 0.7521
Epoch 12/20
8742/8742 [==============================] - 0s 18us/step - loss: 0.4127 -
accuracy: 0.7674 - val_loss: 0.4046 - val_accuracy: 0.7575
Epoch 13/20
8742/8742 [==============================] - 0s 20us/step - loss: 0.3986 -
accuracy: 0.7726 - val_loss: 0.3975 - val_accuracy: 0.7548
Epoch 14/20
8742/8742 [==============================] - 0s 19us/step - loss: 0.3798 -
accuracy: 0.7773 - val_loss: 0.3939 - val_accuracy: 0.8152
Epoch 15/20
8742/8742 [==============================] - 0s 18us/step - loss: 0.3771 -
accuracy: 0.7837 - val_loss: 0.3973 - val_accuracy: 0.7850
Epoch 16/20
8742/8742 [==============================] - 0s 17us/step - loss: 0.3631 -
accuracy: 0.7844 - val_loss: 0.3981 - val_accuracy: 0.8070
Epoch 17/20
8742/8742 [==============================] - 0s 18us/step - loss: 0.3662 -
accuracy: 0.7812 - val_loss: 0.3995 - val_accuracy: 0.7585
Epoch 18/20
8742/8742 [==============================] - 0s 17us/step - loss: 0.3604 -
accuracy: 0.7869 - val_loss: 0.3926 - val_accuracy: 0.7713
Epoch 19/20
8742/8742 [==============================] - 0s 17us/step - loss: 0.3496 -
accuracy: 0.7915 - val_loss: 0.3930 - val_accuracy: 0.7722
Epoch 20/20
8742/8742 [==============================] - 0s 17us/step - loss: 0.3417 -
accuracy: 0.7939 - val_loss: 0.3896 - val_accuracy: 0.7768
```

[121]:
```
loss, test_acc = model_1_PCA_Trained.evaluate(X_PCA_test, y_cat_PCA_test)
print('test_acc: ', test_acc)
```

```
loss, val_acc = model_1_PCA_Trained.evaluate(X_PCA_val, y_cat_PCA_val)
print('val_acc: ', val_acc)

loss, train_acc = model_1_PCA_Trained.evaluate(X_PCA_Train, y_cat_PCA_Train)
print('train_acc: ', train_acc)
```

```
1093/1093 [==============================] - 0s 22us/step
test_acc:  0.7987191081047058
1093/1093 [==============================] - 0s 26us/step
val_acc:  0.7767612338066101
8742/8742 [==============================] - 0s 20us/step
train_acc:  0.812743067741394
```

18 (h) Support vector machine.

[56]:
```
from sklearn import svm
```

[132]:
```
U , X_PCA = PCA_ML633(X , 100, plot='off')

X_PCA_Train = X_PCA[:8742,:]
y_PCA_Train = y[:8742]
X_PCA_val = X_PCA[8742:9835,:]
y_PCA_val = y[8742:9835]
X_PCA_test = X_PCA[9835:,:]
y_PCA_test = y[9835:]
```

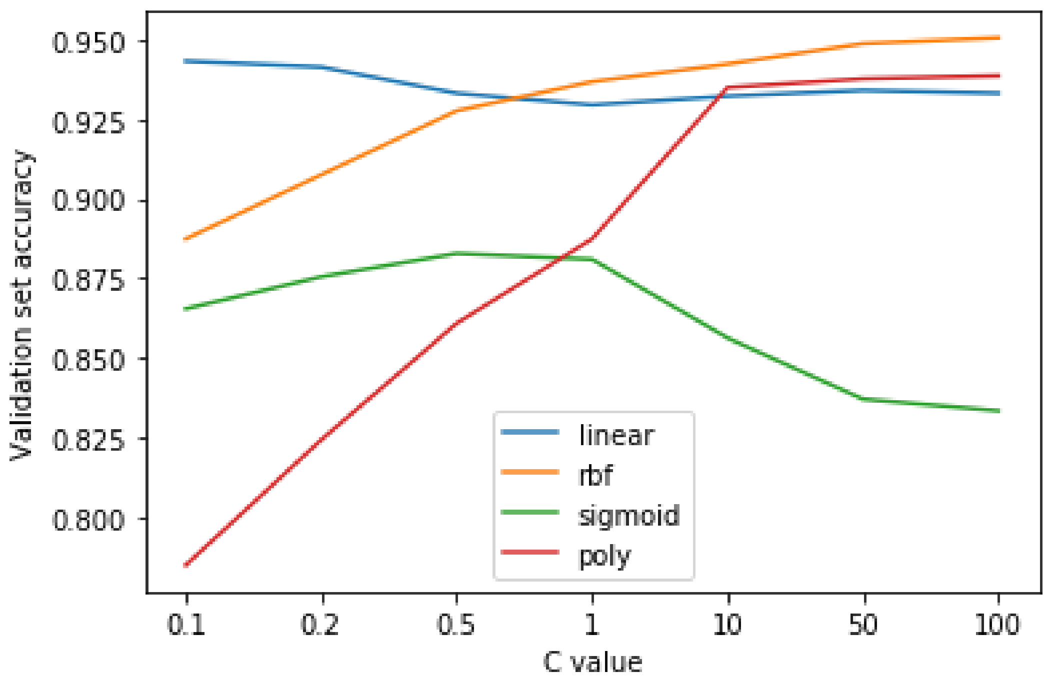
[137]:
```
accuracyDict = dict()
accuracyDict['linear'] = []
accuracyDict['rbf'] = []
accuracyDict['sigmoid'] = []
accuracyDict['poly'] = []
Reg_values = [0.1, 0.2, 0.5, 1, 10, 50, 100]
for kernelX in ['linear', 'rbf', 'sigmoid', 'poly']:
    for reg in Reg_values:
        print('Kernel: ', kernelX, '  C=', reg)
        SVM_clf = svm.SVC(C=reg, kernel=kernelX)
        SVM_clf.fit(X_PCA_Train, y_PCA_Train+1)
        predictions_val = SVM_clf.predict(X_PCA_val)
        predictions_test = SVM_clf.predict(X_PCA_test)
        acc1 = np.sum(np.equal(predictions_val, y_PCA_val+1)) / y_PCA_val.
 ↪shape[0]
        acc2 = np.sum(np.equal(predictions_test, y_PCA_test+1)) / y_PCA_test.
 ↪shape[0]
        print('Val Accuracy = ', acc1, 'Test Accuracy = ', acc2)
        accuracyDict[kernelX].append(acc1)
```

```
Kernel:  linear   C= 0.1
```

```
Val Accuracy =  0.9432753888380604 Test Accuracy =  0.9432753888380604
Kernel:  linear   C= 0.2
Val Accuracy =  0.9414455626715462 Test Accuracy =  0.9451052150045746
Kernel:  linear   C= 0.5
Val Accuracy =  0.9332113449222323 Test Accuracy =  0.9487648673376029
Kernel:  linear   C= 1
Val Accuracy =  0.929551692589204 Test Accuracy =  0.9505946935041171
Kernel:  linear   C= 10
Val Accuracy =  0.9322964318389753 Test Accuracy =  0.9560841720036597
Kernel:  linear   C= 50
Val Accuracy =  0.9341262580054894 Test Accuracy =  0.9560841720036597
Kernel:  linear   C= 100
Val Accuracy =  0.9332113449222323 Test Accuracy =  0.9560841720036597
Kernel:  rbf   C= 0.1
Val Accuracy =  0.8874656907593779 Test Accuracy =  0.8819762122598354
Kernel:  rbf   C= 0.2
Val Accuracy =  0.9075937785910339 Test Accuracy =  0.9204025617566332
Kernel:  rbf   C= 0.5
Val Accuracy =  0.9277218664226898 Test Accuracy =  0.9277218664226898
Kernel:  rbf   C= 1
Val Accuracy =  0.9368709972552608 Test Accuracy =  0.929551692589204
Kernel:  rbf   C= 10
Val Accuracy =  0.9423604757548033 Test Accuracy =  0.9432753888380604
Kernel:  rbf   C= 50
Val Accuracy =  0.9487648673376029 Test Accuracy =  0.9423604757548033
Kernel:  rbf   C= 100
Val Accuracy =  0.9505946935041171 Test Accuracy =  0.9432753888380604
Kernel:  sigmoid   C= 0.1
Val Accuracy =  0.8655077767612077 Test Accuracy =  0.8636779505946935
Kernel:  sigmoid   C= 0.2
Val Accuracy =  0.8755718206770357 Test Accuracy =  0.8774016468435498
Kernel:  sigmoid   C= 0.5
Val Accuracy =  0.8828911253430924 Test Accuracy =  0.889295516925892
Kernel:  sigmoid   C= 1
Val Accuracy =  0.8810612991765783 Test Accuracy =  0.888380603842635
Kernel:  sigmoid   C= 10
Val Accuracy =  0.8563586459286368 Test Accuracy =  0.8508691674290942
Kernel:  sigmoid   C= 50
Val Accuracy =  0.8371454711802379 Test Accuracy =  0.8536139066788655
Kernel:  sigmoid   C= 100
Val Accuracy =  0.8334858188472095 Test Accuracy =  0.8517840805123513
Kernel:  poly   C= 0.1
Val Accuracy =  0.7849954254345837 Test Accuracy =  0.7035681610247027
Kernel:  poly   C= 0.2
Val Accuracy =  0.8243366880146387 Test Accuracy =  0.777676120768527
Kernel:  poly   C= 0.5
Val Accuracy =  0.8609332113449222 Test Accuracy =  0.8298261665141812
Kernel:  poly   C= 1
```

```
Val Accuracy =  0.8874656907593779 Test Accuracy =  0.8655077767612077
Kernel:  poly  C= 10
Val Accuracy =  0.9350411710887465 Test Accuracy =  0.9121683440073193
Kernel:  poly  C= 50
Val Accuracy =  0.9377859103385179 Test Accuracy =  0.929551692589204
Kernel:  poly  C= 100
Val Accuracy =  0.938700823421775 Test Accuracy =  0.9313815187557182
```

[139]:
```python
for kernelX in accuracyDict:
    plt.plot(accuracyDict[kernelX], label=kernelX)

plt.legend()
plt.xlabel('C value')
plt.ylabel('Validation set accuracy')
plt.xticks(np.arange(len(Reg_values)), Reg_values)
plt.show()
```



19 (i) Ensemble learning

[62]:
```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
```

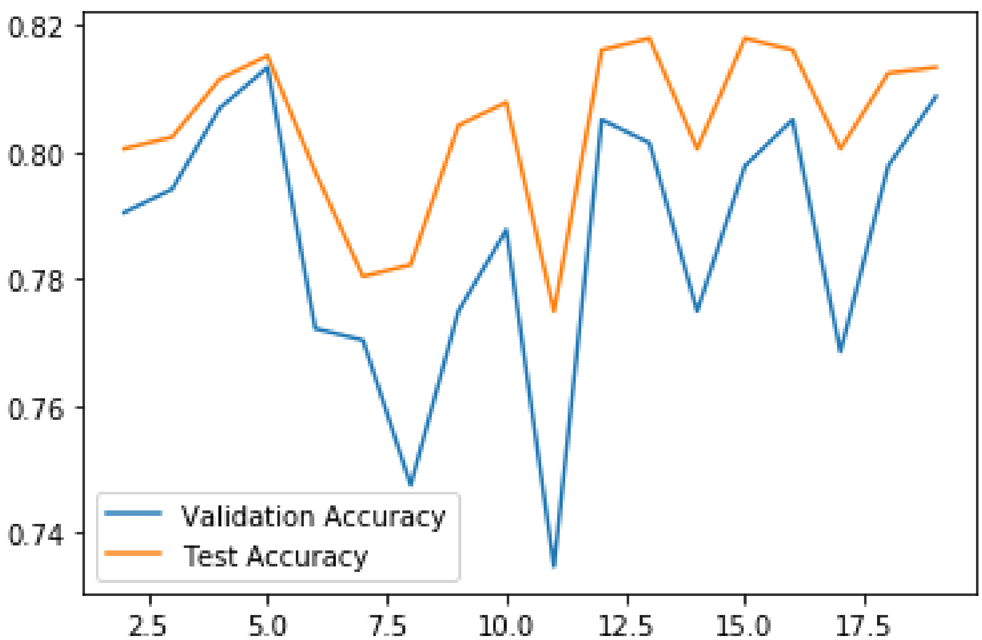
[63]:
```
Val_accuracy = []
Test_accuracy = []
num_estims = []

for n_est in range(2,20):
    estimator = LogisticRegression(multi_class='ovr')
    clf = AdaBoostClassifier(estimator, n_estimators=n_est, algorithm='SAMME')
    clf.fit(X_train, y_train+1)
    predictions_val = clf.predict(X_val)
    predictions_test = clf.predict(X_test)
    acc1 = np.sum(np.equal(predictions_val, y_val+1)) / y_val.shape[0]
    acc2 = np.sum(np.equal(predictions_test, y_test+1)) / y_test.shape[0]
    Val_accuracy.append(acc1)
    Test_accuracy.append(acc2)
    num_estims.append(n_est)
    print('Number of estimators = ', n_est, 'Val Accuracy = ', acc1, 'Test␣
↪Accuracy = ', acc2)
```

```
Number of estimators =  2 Val Accuracy =  0.7904849039341263 Test Accuracy =
0.8005489478499542
Number of estimators =  3 Val Accuracy =  0.7941445562671546 Test Accuracy =
0.8023787740164684
Number of estimators =  4 Val Accuracy =  0.8069533394327539 Test Accuracy =
0.8115279048490394
Number of estimators =  5 Val Accuracy =  0.8133577310155535 Test Accuracy =
0.8151875571820677
Number of estimators =  6 Val Accuracy =  0.7721866422689845 Test Accuracy =
0.7968892955169259
Number of estimators =  7 Val Accuracy =  0.7703568161024703 Test Accuracy =
0.7804208600182982
Number of estimators =  8 Val Accuracy =  0.747483989021043 Test Accuracy =
0.7822506861848124
Number of estimators =  9 Val Accuracy =  0.7749313815187557 Test Accuracy =
0.8042086001829826
Number of estimators =  10 Val Accuracy =  0.787740164684355 Test Accuracy =
0.807868252516011
Number of estimators =  11 Val Accuracy =  0.7346752058554438 Test Accuracy =
0.7749313815187557
Number of estimators =  12 Val Accuracy =  0.8051235132662397 Test Accuracy =
0.8161024702653248
Number of estimators =  13 Val Accuracy =  0.8014638609332113 Test Accuracy =
0.817932296431839
Number of estimators =  14 Val Accuracy =  0.7749313815187557 Test Accuracy =
0.8005489478499542
Number of estimators =  15 Val Accuracy =  0.797804208600183 Test Accuracy =
0.817932296431839
Number of estimators =  16 Val Accuracy =  0.8051235132662397 Test Accuracy =
```

```
0.8161024702653248
Number of estimators =  17 Val Accuracy =  0.7685269899359561 Test Accuracy =
0.8005489478499542
Number of estimators =  18 Val Accuracy =  0.797804208600183 Test Accuracy =
0.8124428179322964
Number of estimators =  19 Val Accuracy =  0.8087831655992681 Test Accuracy =
0.8133577310155535
```

[71]:
```python
plt.figure()
plt.plot(num_estims, Val_accuracy, label='Validation Accuracy')
plt.plot(num_estims, Test_accuracy, label='Test Accuracy')
plt.legend()
```

[71]: <matplotlib.legend.Legend at 0x1effe3c2b48>



[72]:
```python
Val_accuracy = []
Test_accuracy = []
num_estims = []

for n_est in range(1,20):
    estimator = DecisionTreeClassifier(max_depth=2)
    clf = AdaBoostClassifier(estimator, n_estimators=n_est, algorithm='SAMME')
    clf.fit(X_train, y_train+1)
    predictions_val = clf.predict(X_val)
    predictions_test = clf.predict(X_test)
    acc1 = np.sum(np.equal(predictions_val, y_val+1)) / y_val.shape[0]
```

```
    acc2 = np.sum(np.equal(predictions_test, y_test+1)) / y_test.shape[0]
    Val_accuracy.append(acc1)
    Test_accuracy.append(acc2)
    num_estims.append(n_est)
    print('Number of estimators = ', n_est, 'Val Accuracy = ', acc1, 'Test␣
↪Accuracy = ', acc2)
```

```
Number of estimators =  1 Val Accuracy =  0.5233302836230558 Test Accuracy =
0.5297346752058555
Number of estimators =  2 Val Accuracy =  0.5800548947849954 Test Accuracy =
0.5507776761207686
Number of estimators =  3 Val Accuracy =  0.6550777676120768 Test Accuracy =
0.645928636779506
Number of estimators =  4 Val Accuracy =  0.6907593778591034 Test Accuracy =
0.6770356816102471
Number of estimators =  5 Val Accuracy =  0.6605672461116194 Test Accuracy =
0.6678865507776761
Number of estimators =  6 Val Accuracy =  0.6303751143641354 Test Accuracy =
0.6532479414455626
Number of estimators =  7 Val Accuracy =  0.6907593778591034 Test Accuracy =
0.6733760292772186
Number of estimators =  8 Val Accuracy =  0.7026532479414456 Test Accuracy =
0.6916742909423604
Number of estimators =  9 Val Accuracy =  0.6953339432753889 Test Accuracy =
0.6999085086916743
Number of estimators =  10 Val Accuracy =  0.7108874656907593 Test Accuracy =
0.6816102470265325
Number of estimators =  11 Val Accuracy =  0.7255260750228728 Test Accuracy =
0.7118023787740164
Number of estimators =  12 Val Accuracy =  0.7090576395242452 Test Accuracy =
0.7008234217749314
Number of estimators =  13 Val Accuracy =  0.7145471180237878 Test Accuracy =
0.7099725526075022
Number of estimators =  14 Val Accuracy =  0.737419945105215 Test Accuracy =
0.7145471180237878
Number of estimators =  15 Val Accuracy =  0.7282708142726441 Test Accuracy =
0.7053979871912168
Number of estimators =  16 Val Accuracy =  0.7255260750228728 Test Accuracy =
0.6971637694419031
Number of estimators =  17 Val Accuracy =  0.7136322049405307 Test Accuracy =
0.7108874656907593
Number of estimators =  18 Val Accuracy =  0.8133577310155535 Test Accuracy =
0.7904849039341263
Number of estimators =  19 Val Accuracy =  0.8032936870997255 Test Accuracy =
0.7795059469350412
```