Daily and Sports Activities Problem Statement

Daily sports activities (UCI) consist of 19 exercises conducted for 5 minutes by eight subjects (four females and four males between the ages of 20 and 30). The total signal duration for each subject's activity is 5 minutes.

The subjects were invited to perform the exercises in their own unique style and were not restricted in any way. As a result, the speeds and amplitudes of various activities differ amongst subjects. This is a multi-classification problem where we are classifying the person performing which of the following activities.

The 19 activities are:

“Sitting (A1),

standing (A2),

lying on back and on right side (A3 and A4),

ascending and descending stairs (A5 and A6),

standing in an elevator still (A7)

and moving around in an elevator (A8),

walking in a parking lot (A9),

walking on a treadmill with a speed of 4 km/h (in flat and 15 deg inclined positions) (A1

0 and A11),

running on a treadmill with a speed of 8 km/h (A12),

exercising on a stepper (A13),

exercising on a cross trainer (A14),

cycling on an exercise bike in horizontal and vertical positions (A15 and A16),

rowing (A17),

jumping (A18),

and playing basketball (A19)”.

Data Structure:

As per UCI “the dataset comprises motion sensor data of 19 daily and sports activities each performed by 8 subjects in their own style for 5 minutes. Five Xsens MTx units are used on the torso, arms, and legs

19 activities (a)
8 subjects (p)

60 segments (s)
5 units on torso (T), right arm (RA), left arm (LA), right leg (RL), left leg (LL)
9 sensors on each unit (x,y,z accelerometers, x,y,z gyroscopes, x,y,z magnetometers)

Folders a01, a02, ..., a19 contain data recorded from the 19 activities.

For each activity, the subfolders p1, p2, ..., p8 contain data from each of the 8 subjects.

In each subfolder, there are 60 text files s01, s02, ..., s60, one for each segment.

Columns 1-45 correspond to:
T_xacc, T_yacc, T_zacc, T_xgyro, ..., T_ymag, T_zmag,
RA_xacc, RA_yacc, RA_zacc, RA_xgyro, ..., RA_ymag, RA_zmag,
LA_xacc, LA_yacc, LA_zacc, LA_xgyro, ..., LA_ymag, LA_zmag,
RL_xacc, RL_yacc, RL_zacc, RL_xgyro, ..., RL_ymag, RL_zmag,
LL_xacc, LL_yacc, LL_zacc, LL_xgyro, ..., LL_ymag, LL_zmag.

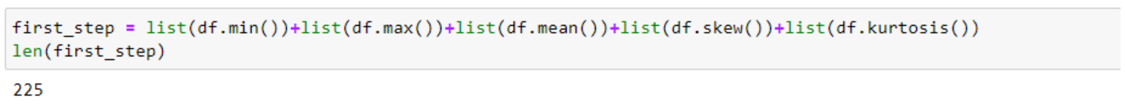
Therefore,
columns 1-9 correspond to the sensors in unit 1 (T),
columns 10-18 correspond to the sensors in unit 2 (RA),
columns 19-27 correspond to the sensors in unit 3 (LA),
columns 28-36 correspond to the sensors in unit 4 (RL),
columns 37-45 correspond to the sensors in unit 5 (LL),”

Pre-processing:

We have taken the input segment, that's a 5 second window of a patient performing an activity, which has 125 observations (5 x 25Hz) with 45 features, because of 9 axes of each sensor unit on torso, left hand, right hand, left leg, right leg. They convert the 125x45 into a handcrafted meaningful 1170x1 matrix (B. Barshan and M. C. Yüksek, 2014).

The 1170 features represent,

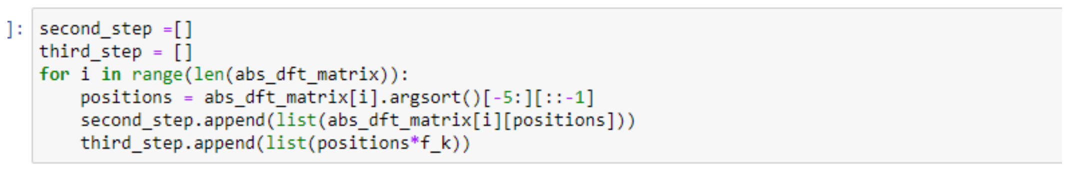
- 225 features (min, max, mean, skewness, kurtosis of all 9 axes of all 5 units, thus 5x9x5).



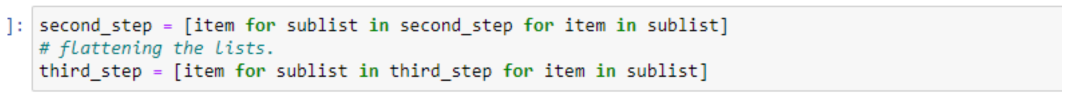
```
first_step = list(df.min())+list(df.max())+list(df.mean())+list(df.skew())+list(df.kurtosis())
len(first_step)
```

```
225
```

- 225 features which represent the maximum 5 peaks of the DFT applied on each of the axes of all the 5 units.
- 225 features which represent the corresponding frequency of the 5 peaks of the DFT over the time series.

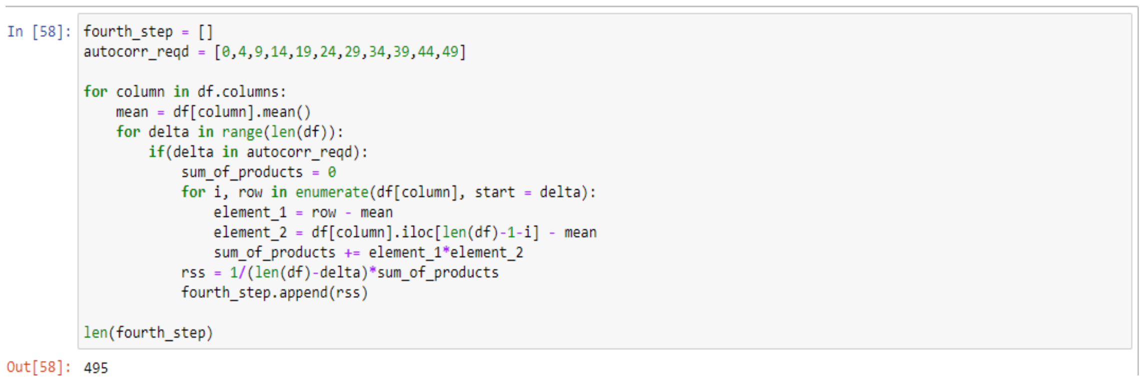


```
]: second_step =[]
   third_step = []
   for i in range(len(abs_dft_matrix)):
       positions = abs_dft_matrix[i].argsort()[-5:][::-1]
       second_step.append(list(abs_dft_matrix[i][positions]))
       third_step.append(list(positions*f_k))
```



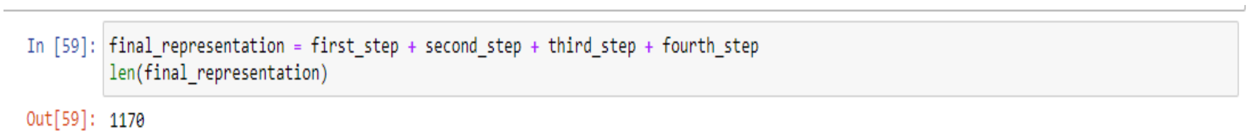
```
]: second_step = [item for sublist in second_step for item in sublist]
   # flattening the lists.
   third_step = [item for sublist in third_step for item in sublist]
```

- 495 features which represent the autocorrelation of the series, 11 handpicked values from the 125 autocorrelation values for each axes, thus 11 x 9 x 5 = 495.



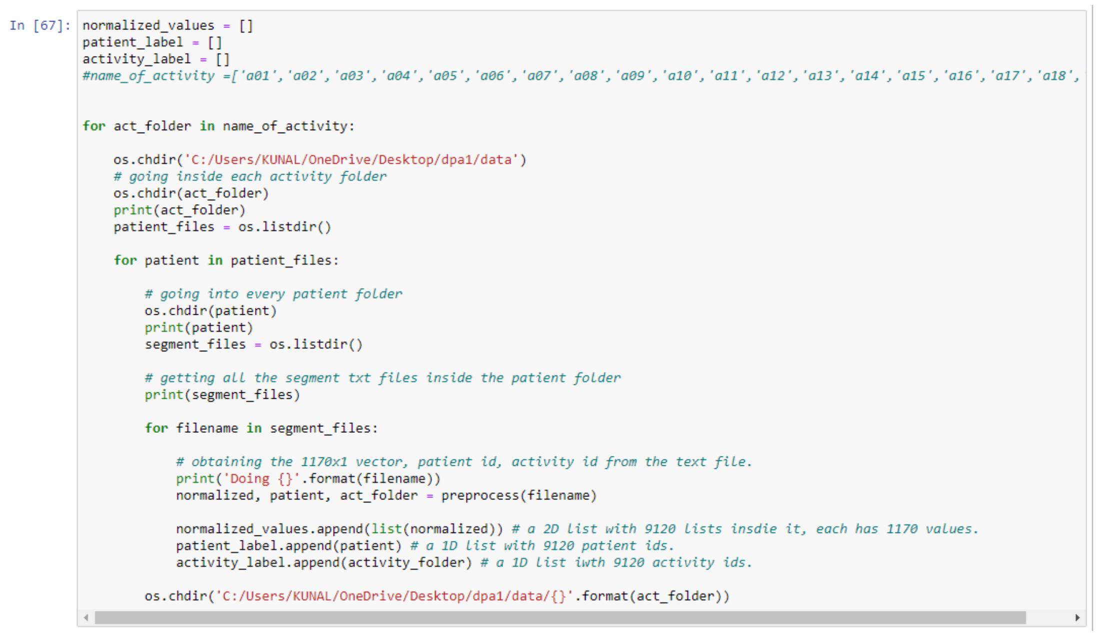
```
In [58]: fourth_step = []
         autocorr_reqd = [0,4,9,14,19,24,29,34,39,44,49]

         for column in df.columns:
             mean = df[column].mean()
             for delta in range(len(df)):
                 if(delta in autocorr_reqd):
                     sum_of_products = 0
                     for i, row in enumerate(df[column], start = delta):
                         element_1 = row - mean
                         element_2 = df[column].iloc[len(df)-1-i] - mean
                         sum_of_products += element_1*element_2
                     rss = 1/(len(df)-delta)*sum_of_products
                     fourth_step.append(rss)

         len(fourth_step)

Out[58]: 495
```

Adding them all 225 + 225 + 225 + 495 = 1170, for each segment, i.e., each text file.



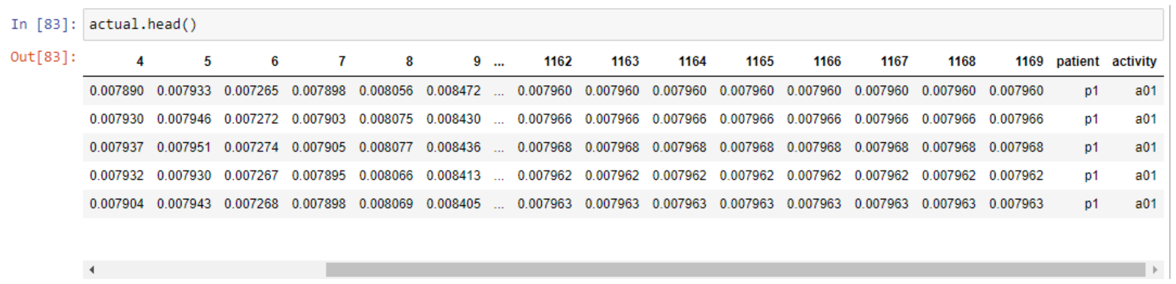
```
In [59]: final_representation = first_step + second_step + third_step + fourth_step
         len(final_representation)

Out[59]: 1170
```

Then these values are normalized in the range [0,1], and stored along with the patient ID and activity ID for that segment / text file.



```
In [67]: normalized_values = []
         patient_label = []
         activity_label = []
         #name_of_activity =['a01','a02','a03','a04','a05','a06','a07','a08','a09','a10','a11','a12','a13','a14','a15','a16','a17','a18',


         for act_folder in name_of_activity:

             os.chdir('C:/Users/KUNAL/OneDrive/Desktop/dpa1/data')
             # going inside each activity folder
             os.chdir(act_folder)
             print(act_folder)
             patient_files = os.listdir()

             for patient in patient_files:

                 # going into every patient folder
                 os.chdir(patient)
                 print(patient)
                 segment_files = os.listdir()

                 # getting all the segment txt files inside the patient folder
                 print(segment_files)

                 for filename in segment_files:

                     # obtaining the 1170x1 vector, patient id, activity id from the text file.
                     print('Doing {}'.format(filename))
                     normalized, patient, act_folder = preprocess(filename)

                     normalized_values.append(list(normalized)) # a 2D list with 9120 lists insdie it, each has 1170 values.
                     patient_label.append(patient) # a 1D list with 9120 patient ids.
                     activity_label.append(activity_folder) # a 1D list iwth 9120 activity ids.

                 os.chdir('C:/Users/KUNAL/OneDrive/Desktop/dpa1/data/{}'.format(act_folder))
```

After doing the above technique for each text file we save the data in CSV file which has final shape of (9120,1172).

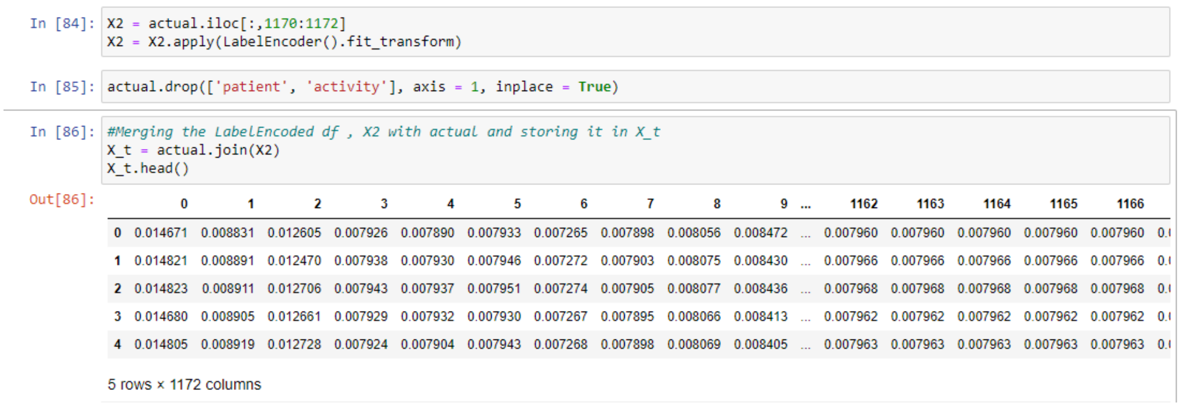


```
In [83]: actual.head()
```

Out[83]:

4	5	6	7	8	9	...	1162	1163	1164	1165	1166	1167	1168	1169	patient	activity
0.007890	0.007933	0.007265	0.007898	0.008056	0.008472	...	0.007960	0.007960	0.007960	0.007960	0.007960	0.007960	0.007960	0.007960	p1	a01
0.007930	0.007946	0.007272	0.007903	0.008075	0.008430	...	0.007966	0.007966	0.007966	0.007966	0.007966	0.007966	0.007966	0.007966	p1	a01
0.007937	0.007951	0.007274	0.007905	0.008077	0.008436	...	0.007968	0.007968	0.007968	0.007968	0.007968	0.007968	0.007968	0.007968	p1	a01
0.007932	0.007930	0.007267	0.007895	0.008066	0.008413	...	0.007962	0.007962	0.007962	0.007962	0.007962	0.007962	0.007962	0.007962	p1	a01
0.007904	0.007943	0.007268	0.007898	0.008069	0.008405	...	0.007963	0.007963	0.007963	0.007963	0.007963	0.007963	0.007963	0.007963	p1	a01

We have applied LabelEncoding, which will convert all string categories ('a1', 'a2'... and 'p1', 'p2'...) into numerical categories [0, 1, 2 ... 18] for activity, since scikit-learn understands numbers only.



```
In [84]: X2 = actual.iloc[:,1170:1172]
         X2 = X2.apply(LabelEncoder().fit_transform)

In [85]: actual.drop(['patient', 'activity'], axis = 1, inplace = True)

In [86]: #Merging the LabelEncoded df , X2 with actual and storing it in X_t
         X_t = actual.join(X2)
         X_t.head()
```

Out[86]:

	0	1	2	3	4	5	6	7	8	9	...	1162	1163	1164	1165	1166	
0	0.014671	0.008831	0.012605	0.007926	0.007890	0.007933	0.007265	0.007898	0.008056	0.008472	...	0.007960	0.007960	0.007960	0.007960	0.007960	0.(
1	0.014821	0.008891	0.012470	0.007938	0.007930	0.007946	0.007272	0.007903	0.008075	0.008430	...	0.007966	0.007966	0.007966	0.007966	0.007966	0.(
2	0.014823	0.008911	0.012706	0.007943	0.007937	0.007951	0.007274	0.007905	0.008077	0.008436	...	0.007968	0.007968	0.007968	0.007968	0.007968	0.(
3	0.014680	0.008905	0.012661	0.007929	0.007932	0.007930	0.007267	0.007895	0.008066	0.008413	...	0.007962	0.007962	0.007962	0.007962	0.007962	0.(
4	0.014805	0.008919	0.012728	0.007924	0.007904	0.007943	0.007268	0.007898	0.008069	0.008405	...	0.007963	0.007963	0.007963	0.007963	0.007963	0.(

5 rows × 1172 columns

Tasks:

1. **Define the training and testing set of your data set.**

Using Sklearn train_test_split we have defined the 80 % of our new saved dataset as training set and other 20 % testing set.

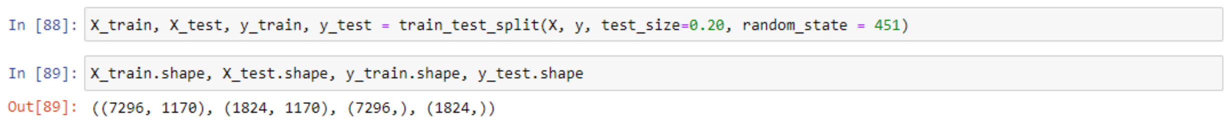
After applying the split, the shape of each are as follows:

X_train: (7296, 1170)

X_test: (1824, 1170

y_train: (7296,)

y_test: (1824,)



```
In [88]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state = 451)

In [89]: X_train.shape, X_test.shape, y_train.shape, y_test.shape
Out[89]: ((7296, 1170), (1824, 1170), (7296,), (1824,))
```

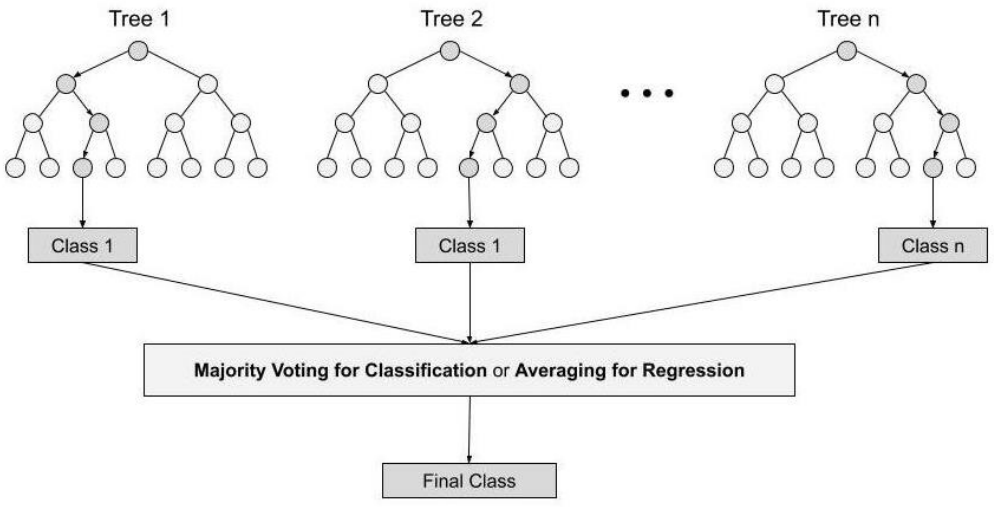
2. **Implement neural network and one other classification algorithm of your choice and compare the performance for the dataset you choose.**

We have used **Random Forest classifier** (scikit-learn, n.d.) for our multi classification problem. It is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

The reason we have used this algorithm are:

1. Overfitting is avoided since random forests are generated from subsets of data and the final output is based on average or majority rating.
2. Random forest collects data at random, creates a decision tree, and takes the average result. It makes no use of any formulas.
3. It is comparatively faster.
4. the feature space is reduced as each tree does not consider all the features present in dataset.

5. Because each tree is built independently from distinct data and attributes, the algorithm makes full use of the CPU to build random forests.


Tree 1
Tree 2
Tree n
Class 1
Class 1
Class n
Majority Voting for Classification or Averaging for Regression
Final Class

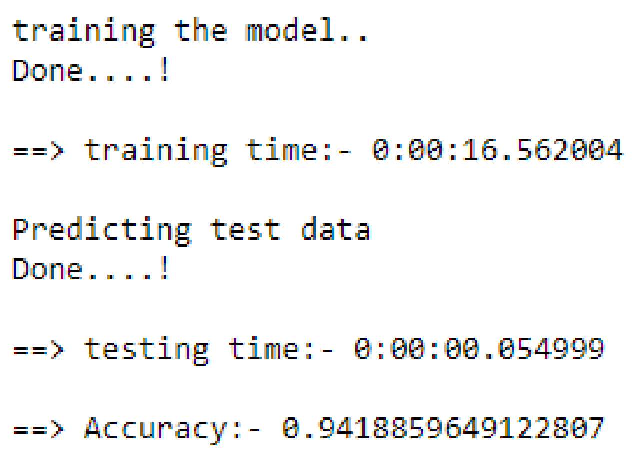
Implementation

Training the model:

```
rfc = RandomForestClassifier()
rf_model=perform_model(rfc, X_train, y_train, X_test, y_test,class_labels=labels)
rf_model
```

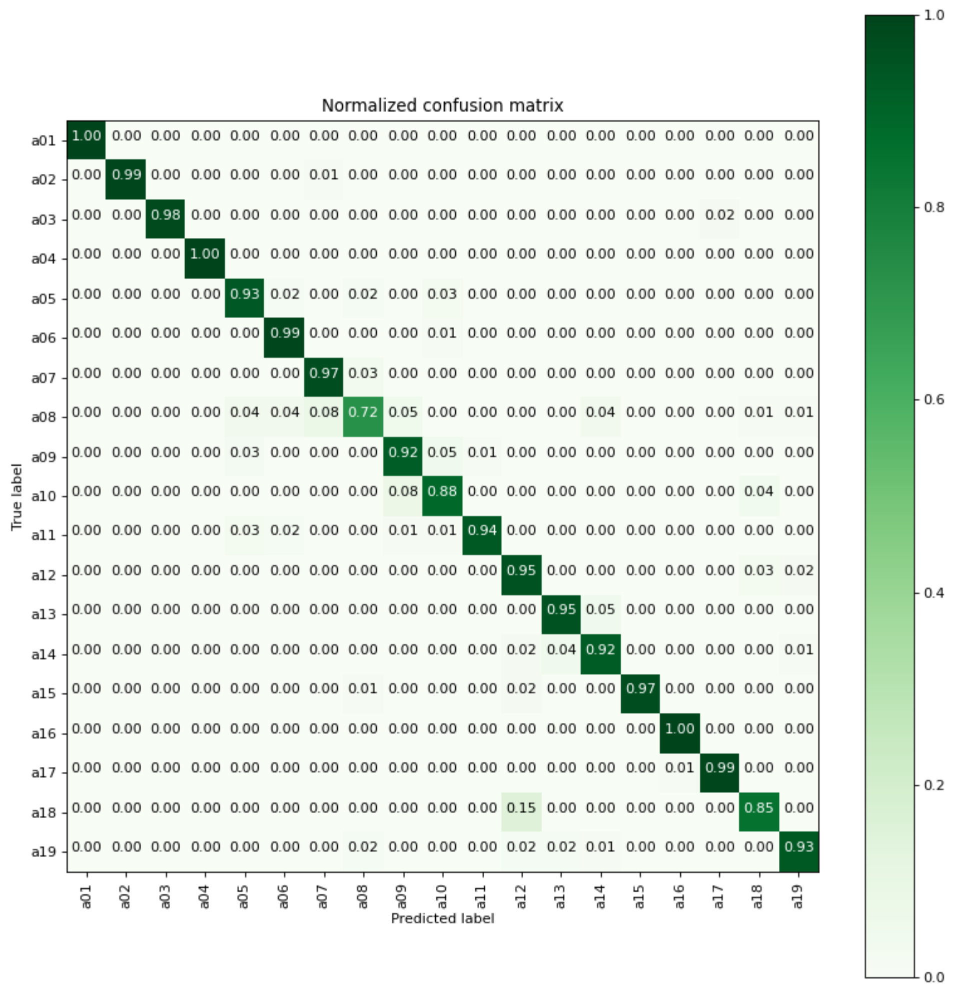
Result:

Time taken to train the model was 0:00:16 with accuracy of 94.18%.



```
training the model..
Done....!

==> training time:- 0:00:16.562004

Predicting test data
Done....!

==> testing time:- 0:00:00.054999

==> Accuracy:- 0.9418859649122807
```

Confusion matrix



	a01	a02	a03	a04	a05	a06	a07	a08	a09	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19
a01	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
a02	0.00	0.99	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
a03	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00
a04	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
a05	0.00	0.00	0.00	0.00	0.93	0.02	0.00	0.02	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
a06	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
a07	0.00	0.00	0.00	0.00	0.00	0.00	0.97	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
a08	0.00	0.00	0.00	0.00	0.04	0.04	0.08	0.72	0.05	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.01	0.01
a09	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.92	0.05	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
a10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.88	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.00
a11	0.00	0.00	0.00	0.00	0.03	0.02	0.00	0.00	0.01	0.01	0.94	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
a12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.95	0.00	0.00	0.00	0.00	0.00	0.03	0.02
a13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.95	0.05	0.00	0.00	0.00	0.00	0.00
a14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.04	0.92	0.00	0.00	0.00	0.00	0.01
a15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.02	0.00	0.00	0.97	0.00	0.00	0.00	0.00
a16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
a17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.99	0.00	0.00
a18	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.15	0.00	0.00	0.00	0.00	0.00	0.85	0.00
a19	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.02	0.02	0.01	0.00	0.00	0.00	0.00	0.93

Neural network:

We have used PyTorch (Pytorch team, n.d.) model while training our data on neural network.

Model parameters: Below are the parameters we have used to train the model.

EPOCHS = 500

BATCH_SIZE = 64

LEARNING_RATE = 1e-2

Optimizer: Adam Optimizer

Implementation:

1. Uploading the data and converting the NumPy data to torch as model only accept torch data type.

```
]: X_train = X_train.values
   X_test = X_test.values
   y_train = y_train.values
   y_test = y_test.values
```

```
]: X_train = torch.from_numpy(X_train)
   X_test = torch.from_numpy(X_test)
   y_train = torch.from_numpy(y_train)
   y_test = torch.from_numpy(y_test)
```

2. Defining test and train data.

```
y_train = y_train.long()
y_test = y_test.long()
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
(torch.Size([7296, 1170]),
 torch.Size([1824, 1170]),
 torch.Size([7296]),
 torch.Size([1824]))
```

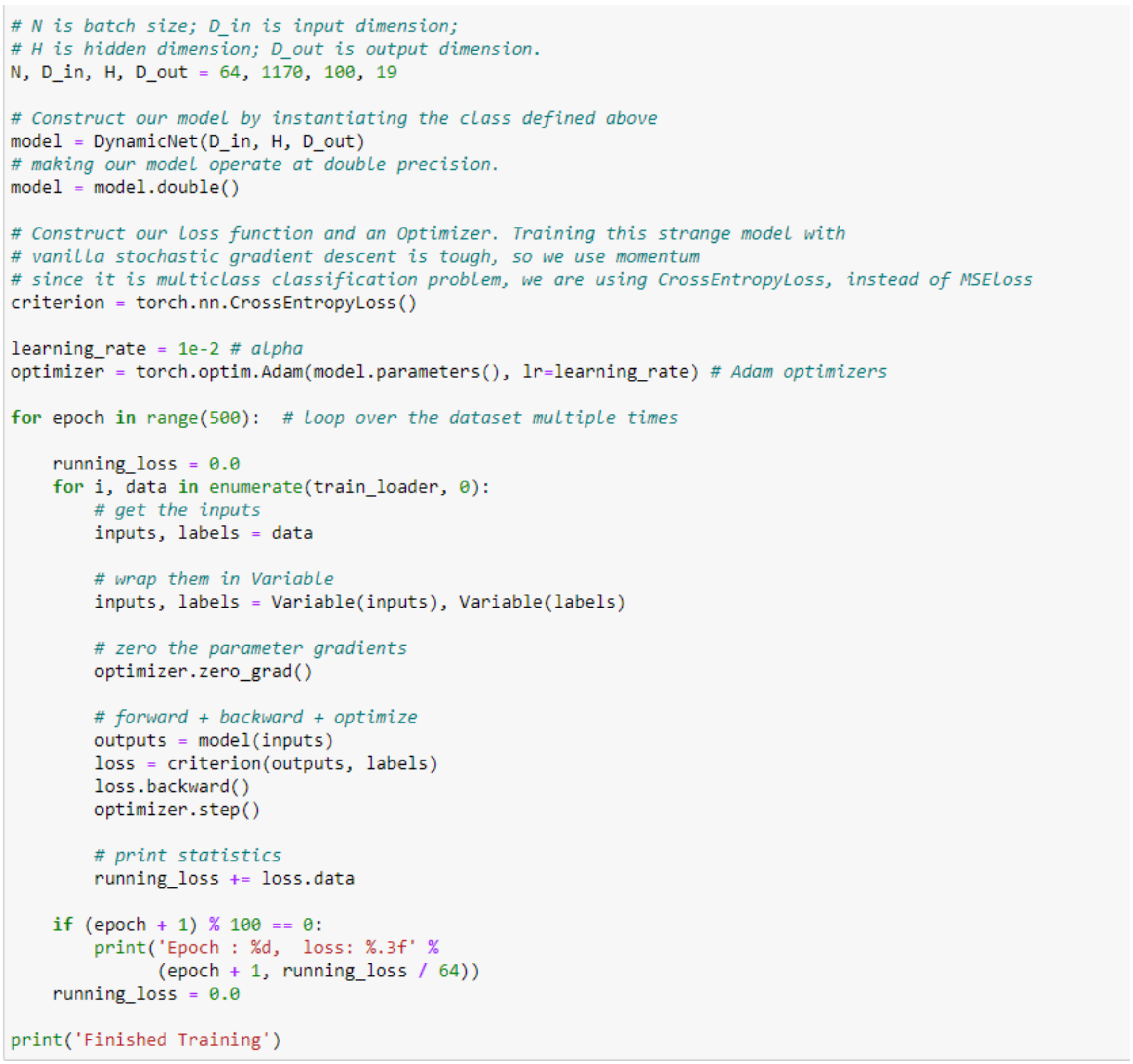
3. In the constructor we construct three nn.Linear instances that we will use in the forward pass.

```
def __init__(self, D_in, H, D_out):
    """
   .
    """
    super(DynamicNet, self).__init__()
    self.input_linear = torch.nn.Linear(D_in, H)
    self.middle_linear = torch.nn.Linear(H, H)
    self.output_linear = torch.nn.Linear(H, D_out)
```

4. For the forward pass of the model, we randomly choose either 0, 1, 2, or 3 and reuse the middle_linear Module that many times to compute hidden layer representations.
   Since each forward pass builds a dynamic computation graph, we have used normal Python control-flow operators like loops or conditional statements when defining the forward pass of the model. Here we also see that it is perfectly safe to reuse the same Module man times when defining a computational graph.

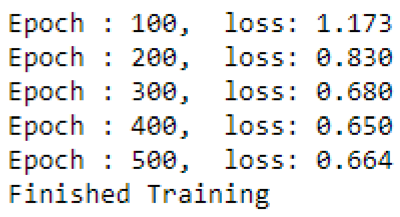
```
def forward(self, x):
    """

    """
    h_relu = self.input_linear(x).clamp(min=0)
    for _ in range(np.random.randint(0, 3)):
        h_relu = self.middle_linear(h_relu).clamp(min=0)
    y_pred = self.output_linear(h_relu)
    return y_pred
```

5. Training the model and defining the parameters.



```python
# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1170, 100, 19

# Construct our model by instantiating the class defined above
model = DynamicNet(D_in, H, D_out)
# making our model operate at double precision.
model = model.double()

# Construct our loss function and an Optimizer. Training this strange model with
# vanilla stochastic gradient descent is tough, so we use momentum
# since it is multiclass classification problem, we are using CrossEntropyLoss, instead of MSEloss
criterion = torch.nn.CrossEntropyLoss()

learning_rate = 1e-2 # alpha
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate) # Adam optimizers

for epoch in range(500):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        # get the inputs
        inputs, labels = data

        # wrap them in Variable
        inputs, labels = Variable(inputs), Variable(labels)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.data

    if (epoch + 1) % 100 == 0:
        print('Epoch : %d,  loss: %.3f' %
              (epoch + 1, running_loss / 64))
    running_loss = 0.0

print('Finished Training')
```

Results:



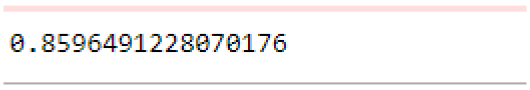
```
Epoch : 100,  loss: 1.173
Epoch : 200,  loss: 0.830
Epoch : 300,  loss: 0.680
Epoch : 400,  loss: 0.650
Epoch : 500,  loss: 0.664
Finished Training
```

After passing the data through the model we got accuracy of 85.97

```
X_test_var = Variable(X_test, volatile=True)
scores = model(X_test_var)
_, preds = torch.max(scores, dim=1)
```

accuracy_score(preds.data.numpy(), y_test.numpy())

Score:



```
0.8596491228070176
```

Comparing the models:

Algorithms	**Accuracy**	**Training time**
Classification (Random Forest classifier)	94.18 %.	0:00:16
Neural Network	85.97 %	0:00:50

3. Apply Principal Component Analysis to the dataset and explain its outcome. How does the number of principal components affect the percentage of variance covered for this dataset?

Principal Component Analysis, or PCA (scikit-learn team, n.d.), is a popular approach for minimizing the dimensionality of a big data collection. Limiting the number of variables or features reduces accuracy but streamlines, analyses, and depicts a huge data collection. PCA implementation initiates by ensuring that the data is uniform. (Where the mean is 0 and the variance is 1), and calculating the dimension's covariance matrix. Using either correlation matrix or even single value decomposition covariance matrix, retrieve Eigenvectors and Eigenvalues. Identify the top k Eigenvectors that equate to the k greatest eigenvalues (k will be the number of dimensions of the rebuilt feature subspace, k≤d, d is the number of initial dimensions) by ranking eigenvalues in decreasing order. Generate the projection matrix W from the k Eigenvectors specified. To derive the new k-dimensional feature subspace Y, reform the underlying data set X through W. Employ the dimensional projection matrix W to translate dataset onto the current subspace using the equation Y=XW in the final iteration.

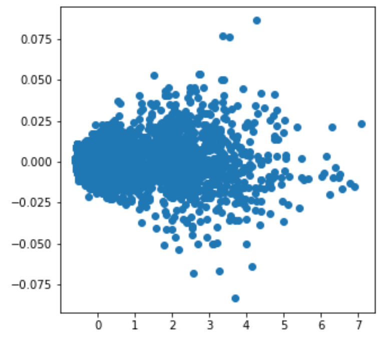
Implementation:

The below code will convert our dataset of shape (9120x1170) to (9120x40).

```
from sklearn.decomposition import PCA
X = actual
pca = PCA(n_components=40)
pca.fit(X)
X_dash = pca.transform(X)
```

Let’s visualise the PCA transformation

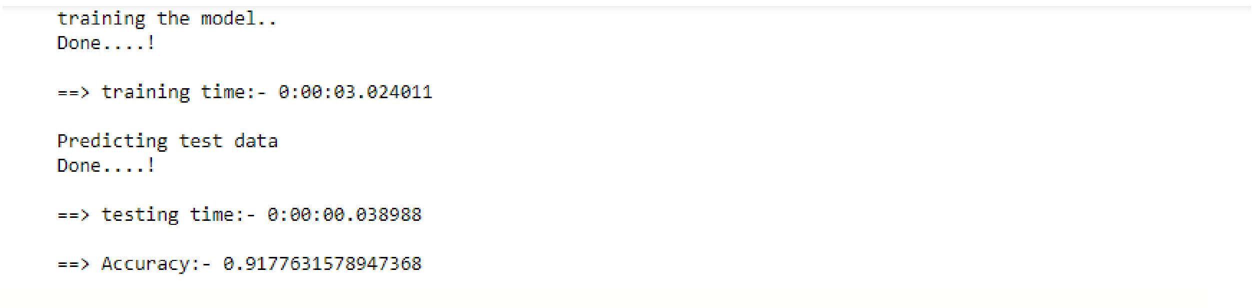
```
In [118]: plt.figure(figsize = (5,5))
          plt.scatter(X_dash[:,0],X_dash[:,39])
          plt.show()
```



The reduces matrix looks like:

```
In [136]: X_dash # the reduced matrix.

Out[136]: array([[-5.56408088e-01,  2.82491226e-05,  4.18084225e-04, ...,
                   2.79633577e-03,  4.96661456e-04, -1.57624137e-03],
                 [-5.55707576e-01,  1.79318883e-05,  4.19020170e-04, ...,
                   3.58153377e-03,  8.50752823e-04, -2.05115087e-03],
                 [-5.56088048e-01,  2.84927560e-05,  4.17309669e-04, ...,
                   2.88050192e-03,  5.59577374e-04, -1.54900339e-03],
                 ...,
                 [ 5.44728749e-01, -8.36528255e-02, -7.06172376e-02, ...,
                  -2.73027039e-03, -2.90169891e-03,  3.68517592e-02],
                 [ 9.44722781e-01,  4.37066939e-04, -1.28225518e-02, ...,
                  -1.00761373e-02, -1.54419632e-02, -4.93149280e-03],
                 [ 1.98182134e+00,  3.82713822e-03,  3.52894440e-02, ...,
                  -2.83749418e-02, -1.54279305e-02, -1.62476592e-02]])
```

Using the PCA dataset and applying the Random Forest classifier.



```
training the model..
Done....!

==> training time:- 0:00:03.024011

Predicting test data
Done....!

==> testing time:- 0:00:00.038988

==> Accuracy:- 0.9177631578947368
```

From the above result we can infer that that accuracy is reduced with PCA to 91.78% also its much faster as the training time is reduced to 0:00:03.

4. **Apply any feature selection method of your choice and compare the performance using any one algorithm used in question 2 before and after applying the feature selection algorithm.**

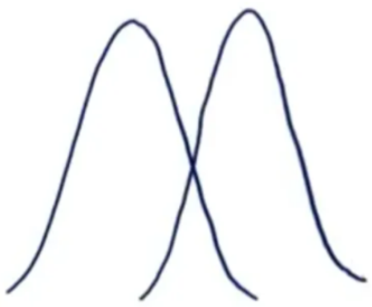
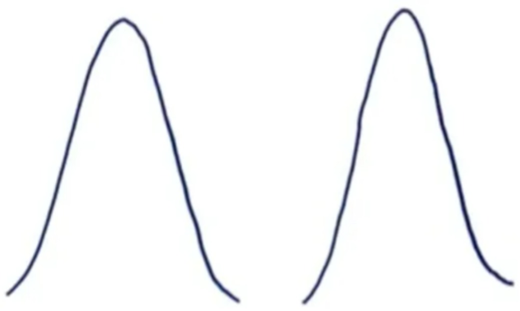
To increase the score of the model we need the dataset that has high variance, so it will be good if we can select the features in the dataset which has variance. We can do this by ANOVA (Analysis of Variance) (sampath kumar gajawada, n.d.) on the basis of f1 score.

Analysis of Variance is a statistical method, used to check the means of two or more groups that are significantly different from each other. It assumes Hypothesis as

H0: Means of all groups are equal.

H1: At least one mean of the groups is different.

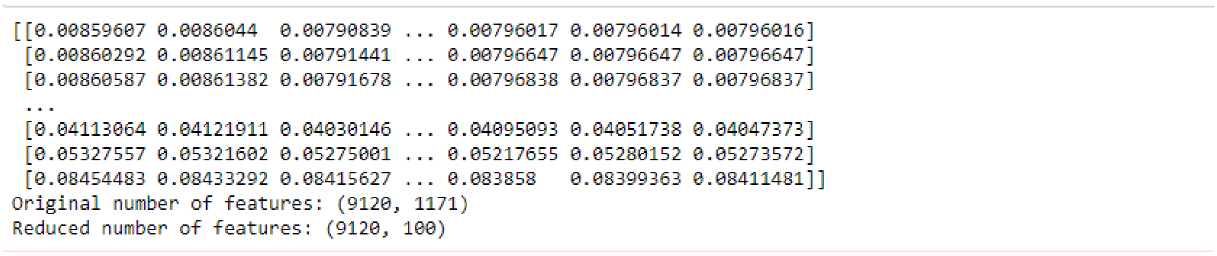
How comparison of means transformed to the comparison of variance?

a. Two distributions Overlap each Other • Individual means don't differ by great margin. • Difference between individual means and grand mean would be less.	b .Two distributions differ from each other • Individual means and grand mean differ by larger distance. • Difference between individual means and grand mean would be large.
	

According to the above figure, if the distributions overlap or are close, the grand mean will be similar to the individual means, however if the distributions are far apart, the grand mean and individual means will diverge by a greater distance. It refers to differences between groups since the values in each group differ. So, in ANOVA, we will examine between-group and within-group variability. ANOVA employs F-test to determine whether or not there is a significant difference between the groups. If there is no significant difference between groups and all variances are identical, the F-ratio of ANOVA will be close to one.

Applying the transformation and reducing the features to 100.

```
fvalue_Best = SelectKBest(f_classif, k=100)
X_kbest = fvalue_Best.fit_transform(X, Y)
print(X_kbest)

print('Original number of features:', X.shape)
print('Reduced number of features:', X_kbest.shape)
```

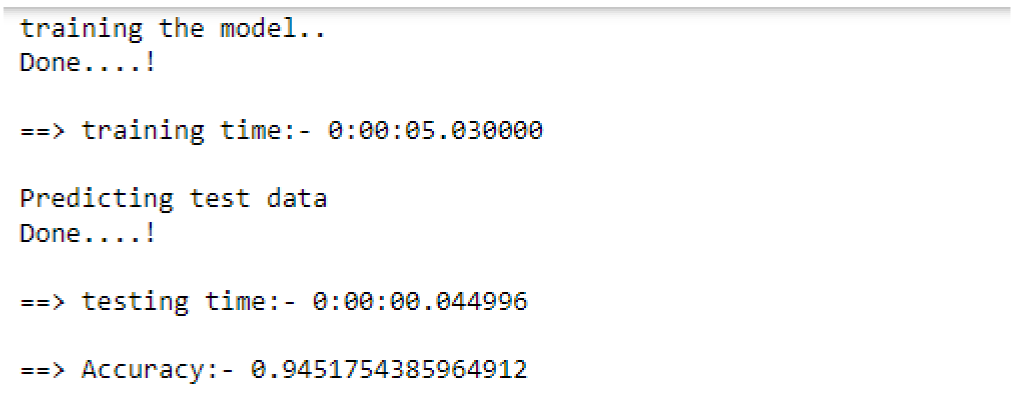


```
[[0.00859607 0.0086044  0.00790839 ... 0.00796017 0.00796014 0.00796016]
 [0.00860292 0.00861145 0.00791441 ... 0.00796647 0.00796647 0.00796647]
 [0.00860587 0.00861382 0.00791678 ... 0.00796838 0.00796837 0.00796837]
 ...
 [0.04113064 0.04121911 0.04030146 ... 0.04095093 0.04051738 0.04047373]
 [0.05327557 0.05321602 0.05275001 ... 0.05217655 0.05280152 0.05273572]
 [0.08454483 0.08433292 0.08415627 ... 0.083858   0.08399363 0.08411481]]
Original number of features: (9120, 1171)
Reduced number of features: (9120, 100)
```

Applying Random Forest Classifier after feature selection**.**

```
rfc = RandomForestClassifier()
rf_model=perform_model(rfc, X_train, y_train, X_test, y_test,class_labels=labels)
rf_model
```

Results:



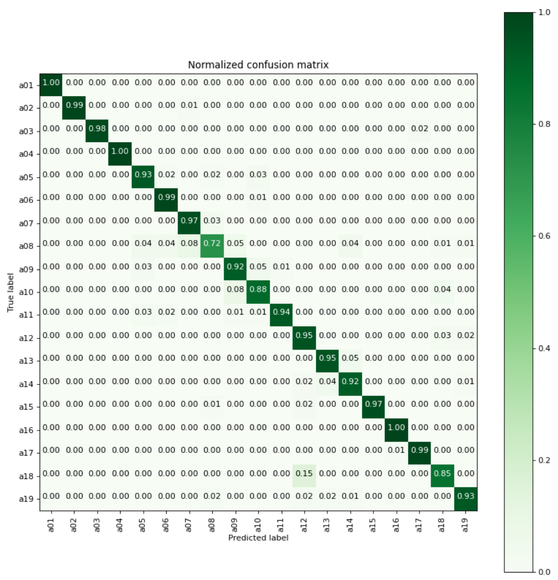
```
training the model..
Done....!

==> training time:- 0:00:05.030000

Predicting test data
Done....!

==> testing time:- 0:00:00.044996

==> Accuracy:- 0.9451754385964912
```

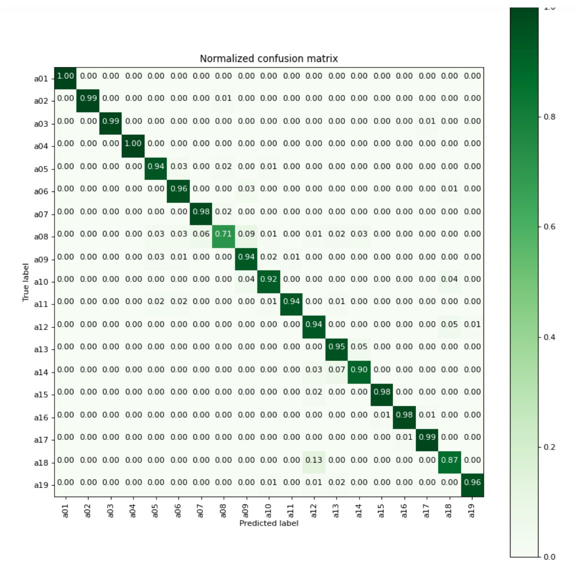
Comparison of Random Forest Classifier before and after feature selection method.

Algorithms	**Accuracy**	**Training time**
Before feature selection method	94.18%.	0:00:16
After feature selection method	94.51 %	0:00:05

From the above table we can infer that there is a slight increase in Accuracy but the good change in training time as after feature select the classifier was much fast.

Confusion Matrix comparison:





Before feature selection method After feature selection method

5. **Discuss the challenges and implications regarding the time required to build the required models. Compare the times with and without feature selection method.**

Below table refers to the time of each model we have trained above:

Algorithms	**Accuracy**	**Training time**
Random Forest Classifier Before feature selection method	94.18%.	0:00:16
Random Forest Classifier After feature selection method	94.51 %	0:00:05
Neural Network	85.97 %	0:00:50
Random Forest Classifier With PCA	91.78%	0:00:03

The main challenge was to pre-process the data for the algorithm to read it and it was tackled as per the above-mentioned pre-processing technique. The project represents the evident fact that data pre-processing facilitates the accuracy of the occupancy data set.

Employed algorithms of Random Forest Classifier After feature selection method has the maximum accuracy and also the less training time. The same algorithm after PCA has the least training time but the accuracy was less.

Neural network took the most training time with least accuracy, may be after doing hyperparameter tuning accuracy can increase.