Assignment 2 (EE769)

L Sai Sandeep Reddy(213070023) , Abhishek Verma(213070024)

Importing all the Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_predict
from sklearn import svm
from sklearn import linear_model
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_boston
from sklearn.inspection import permutation_importance
```

Importing the Training Data Set

```
# Importing the Train data Set Provided into Goggle colab.
from google.colab import files
data = pd.read_csv('https://www.ee.iitb.ac.in/~asethi/Dump/MouseTrain.csv')
for fn in data.keys():
  print('You uploaded "{name}" with length "{length}" bytes'.format(name=fn, length=len(da
```

```
You uploaded "CAMKII_N" with length "762" bytes
You uploaded "CREB_N" with length "762" bytes
You uploaded "ELK_N" with length "762" bytes
You uploaded "ERK_N" with length "762" bytes
```

```
You uploaded "ERK_N" with length "762" bytes
You uploaded "GSK3B_N" with length "762" bytes
You uploaded "JNK_N" with length "762" bytes
You uploaded "MEK_N" with length "762" bytes
You uploaded "TRKA_N" with length "762" bytes
You uploaded "RSK_N" with length "762" bytes
You uploaded "APP_N" with length "762" bytes
You uploaded "Bcatenin_N" with length "762" bytes
You uploaded "SOD1_N" with length "762" bytes
You uploaded "MTOR_N" with length "762" bytes
You uploaded "P38_N" with length "762" bytes
You uploaded "pMTOR_N" with length "762" bytes
You uploaded "DSCR1_N" with length "762" bytes
You uploaded "AMPKA_N" with length "762" bytes
You uploaded "NR2B_N" with length "762" bytes
You uploaded "pNUMB_N" with length "762" bytes
You uploaded "RAPTOR_N" with length "762" bytes
You uploaded "TIAM1_N" with length "762" bytes
You uploaded "pP70S6_N" with length "762" bytes
You uploaded "NUMB_N" with length "762" bytes
You uploaded "P70S6_N" with length "762" bytes
You uploaded "pGSK3B_N" with length "762" bytes
You uploaded "pPKCG_N" with length "762" bytes
You uploaded "CDK5_N" with length "762" bytes
You uploaded "S6_N" with length "762" bytes
You uploaded "ADARB1_N" with length "762" bytes
You uploaded "AcetylH3K9_N" with length "762" bytes
You uploaded "RRP1_N" with length "762" bytes
You uploaded "BAX_N" with length "762" bytes
You uploaded "ARC_N" with length "762" bytes
You uploaded "ERBB4_N" with length "762" bytes
You uploaded "nNOS_N" with length "762" bytes
You uploaded "Tau_N" with length "762" bytes
You uploaded "GFAP_N" with length "762" bytes
You uploaded "GluR3_N" with length "762" bytes
You uploaded "GluR4_N" with length "762" bytes
You uploaded "IL1B_N" with length "762" bytes
You uploaded "P3525_N" with length "762" bytes
You uploaded "pCASP9_N" with length "762" bytes
You uploaded "PSD95_N" with length "762" bytes
You uploaded "SNCA_N" with length "762" bytes
You uploaded "Ubiquitin_N" with length "762" bytes
You uploaded "pGSK3B_Tyr216_N" with length "762" bytes
You uploaded "SHH_N" with length "762" bytes
You uploaded "BAD_N" with length "762" bytes
You uploaded "BCL2_N" with length "762" bytes
You uploaded "pS6_N" with length "762" bytes
You uploaded "pCFOS_N" with length "762" bytes
You uploaded "SYP_N" with length "762" bytes
You uploaded "H3AcK18_N" with length "762" bytes
You uploaded "EGR1_N" with length "762" bytes
You uploaded "H3MeK4_N" with length "762" bytes
You uploaded "CaNA_N" with length "762" bytes
You uploaded "Genotype" with length "762" bytes
You uploaded "Treatment_Behavior" with length "762" bytes
```

```
display(data.dtypes) # Displaying data types to see objects and floats
```

```
DYRK1A_N              float64
ITSN1_N               float64
BDNF_N                float64
NR1_N                 float64
NR2A_N                float64
                       ...
EGR1_N                float64
H3MeK4_N              float64
CaNA_N                float64
Genotype               object
Treatment_Behavior     object
```

```
display(data) # Displaying data
```

	DYRK1A_N	ITSN1_N	BDNF_N	NR1_N	NR2A_N	pAKT_N	pBRAF_N	pCAMKII_N	
0	0.503644	0.747193	0.430175	2.816329	5.990152	0.218830	0.177565	2.373744	0
1	0.514617	0.689064	0.411770	2.789514	5.685038	0.211636	0.172817	2.292150	0
2	0.509183	0.730247	0.418309	2.687201	5.622059	0.209011	0.175722	2.283337	0
3	0.442107	0.617076	0.358626	2.466947	4.979503	0.222886	0.176463	2.152301	0
4	0.434940	0.617430	0.358802	2.365785	4.718679	0.213106	0.173627	2.134014	0
...	...	...	...	...	...	...	...	...	
757	0.394169	0.623696	0.316564	2.104045	3.314706	0.234529	0.200657	3.208804	0
758	0.401739	0.659571	0.321841	2.183930	3.411782	0.236546	0.190826	3.384350	0
759	0.382182	0.586545	0.318545	2.072000	3.073636	0.259636	0.202727	3.006364	0
760	0.365622	0.572209	0.297992	2.004931	2.783551	0.245333	0.190384	2.948221	0
761	0.365605	0.599499	0.301539	2.062813	2.952577	0.248568	0.198819	3.069435	0

762 rows × 79 columns

Checking the datatypes of all the features

```
for col in data.columns.values :
  list_vals = pd.unique(data[col])
  print(col + ' is of type ' + str(data[col].dtypes) + ' has ' + str(len(list_vals)) + ' u
  if len(list_vals) <10:
    list_str = ''
    for n_value in range(0,len(list_vals)):
      list_str = list_str + str(list_vals[n_value]) + ','
    print('These are :' + list_str[0:len(list_str)-2])
```

```
ELK_N is of type float64 has 748 unique values
ERK_N is of type float64 has 762 unique values
GSK3B_N is of type float64 has 762 unique values
JNK_N is of type float64 has 762 unique values
MEK_N is of type float64 has 759 unique values
```

```
MEK_N is of type float64 has 759 unique values
TRKA_N is of type float64 has 762 unique values
RSK_N is of type float64 has 761 unique values
APP_N is of type float64 has 762 unique values
Bcatenin_N is of type float64 has 748 unique values
SOD1_N is of type float64 has 762 unique values
MTOR_N is of type float64 has 762 unique values
P38_N is of type float64 has 761 unique values
pMTOR_N is of type float64 has 762 unique values
DSCR1_N is of type float64 has 762 unique values
AMPKA_N is of type float64 has 762 unique values
NR2B_N is of type float64 has 762 unique values
pNUMB_N is of type float64 has 762 unique values
RAPTOR_N is of type float64 has 762 unique values
TIAM1_N is of type float64 has 760 unique values
pP70S6_N is of type float64 has 761 unique values
NUMB_N is of type float64 has 762 unique values
P70S6_N is of type float64 has 762 unique values
pGSK3B_N is of type float64 has 762 unique values
pPKCG_N is of type float64 has 762 unique values
CDK5_N is of type float64 has 762 unique values
S6_N is of type float64 has 762 unique values
ADARB1_N is of type float64 has 762 unique values
AcetylH3K9_N is of type float64 has 762 unique values
RRP1_N is of type float64 has 762 unique values
BAX_N is of type float64 has 762 unique values
ARC_N is of type float64 has 762 unique values
ERBB4_N is of type float64 has 762 unique values
nNOS_N is of type float64 has 762 unique values
Tau_N is of type float64 has 762 unique values
GFAP_N is of type float64 has 762 unique values
GluR3_N is of type float64 has 762 unique values
GluR4_N is of type float64 has 762 unique values
IL1B_N is of type float64 has 762 unique values
P3525_N is of type float64 has 762 unique values
pCASP9_N is of type float64 has 762 unique values
PSD95_N is of type float64 has 762 unique values
SNCA_N is of type float64 has 761 unique values
Ubiquitin_N is of type float64 has 762 unique values
pGSK3B_Tyr216_N is of type float64 has 762 unique values
SHH_N is of type float64 has 762 unique values
BAD_N is of type float64 has 582 unique values
BCL2_N is of type float64 has 568 unique values
pS6_N is of type float64 has 762 unique values
pCFOS_N is of type float64 has 703 unique values
SYP_N is of type float64 has 762 unique values
H3AcK18_N is of type float64 has 613 unique values
EGR1_N is of type float64 has 583 unique values
H3MeK4_N is of type float64 has 583 unique values
CaNA_N is of type float64 has 762 unique values
Genotype is of type object has 2 unique values
These are :Control,Ts65D
Treatment_Behavior is of type object has 4 unique values
These are :Memantine_C/S,Memantine_S/C,Saline_C/S,Saline_S/
```

Checking Correlation

```
#Correlation Matrix and heatmap is drawn.
corr = data.corr()
sns.set(rc = {'figure.figsize':(20,16)})
sns.heatmap(corr)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc5a5c528d0>
```

Identifying and Dropping the Correlated features

```
columns = np.full((corr.shape[0],), True, dtype=bool)
for i in range(corr.shape[0]):
    for j in range(i+1, corr.shape[0]):
        if corr.iloc[i,j] >= 0.85:
            if columns[j]:
                columns[j] = False
temp=data.columns[:-2]
selected_columns = temp[columns]
print(len(selected_columns))
rm=[]
q=['Genotype','Treatment_Behavior']
for i in np.array(data.columns):
  if i not in selected_columns and i not in q:
    rm.append(i)
data=data.drop(labels=rm,axis=1)
display(data)
print(rm) # These columns are Dropped.
print(data.shape)
```

```
67
```

	DYRK1A_N	BDNF_N	NR1_N	pAKT_N	pBRAF_N	pCAMKII_N	pCREB_N	pELK_N	
0	0.503644	0.430175	2.816329	0.218830	0.177565	2.373744	0.232224	1.750936	0
1	0.514617	0.411770	2.789514	0.211636	0.172817	2.292150	0.226972	1.596377	0
2	0.509183	0.418309	2.687201	0.209011	0.175722	2.283337	0.230247	1.561316	0
3	0.442107	0.358626	2.466947	0.222886	0.176463	2.152301	0.207004	1.595086	0
4	0.434940	0.358802	2.365785	0.213106	0.173627	2.134014	0.192158	1.504230	0
...	...	...	...	...	...	...	...	...	
757	0.394169	0.316564	2.104045	0.234529	0.200657	3.208804	0.221809	1.218951	0
758	0.401739	0.321841	2.183930	0.236546	0.190826	3.384350	0.231899	1.250487	0
759	0.382182	0.318545	2.072000	0.259636	0.202727	3.006364	0.223455	1.248727	0
760	0.365622	0.297992	2.004931	0.245333	0.190384	2.948221	0.217506	1.177527	0
761	0.365605	0.301539	2.062813	0.248568	0.198819	3.069435	0.226736	1.220472	0

762 rows × 69 columns

```
['ITSN1_N', 'NR2A_N', 'pERK_N', 'pMEK_N', 'pNR1_N', 'pNR2B_N', 'BRAF_N', 'ERK_N', 'B
(762, 69)
```

```
features = data.iloc[:,:-2].values  # Removinng the last 2 columns to get features data se
display(features.shape)
```

```
(762, 67)
```

Taking care of Missing Data(Multivariate feature imputation)

Source:https://scikit-learn.org/stable/modules/impute.html

```
# Doing Multivariate Feature Imputation to deal with missing data.
imputer = IterativeImputer(max_iter=10, random_state=0)
imputer.fit(features)
imputed_features= imputer.transform(features)
display(imputed_features)
```

```
array([[0.50364388, 0.4301753 , 2.81632854, ..., 0.13179003, 0.1281856 ,
        1.67565235],
       [0.51461708, 0.41177034, 2.78951404, ..., 0.13510297, 0.1311187 ,
        1.74360965],
       [0.50918309, 0.41830878, 2.68720107, ..., 0.13336183, 0.12743108,
        1.92642659],
       ...,
       [0.38218182, 0.31854545, 2.072     , ..., 0.16364894, 0.19121209,
        1.43351401],
       [0.3656217 , 0.29799225, 2.00493131, ..., 0.1406397 , 0.21389934,
        1.54327375],
       [0.36560487, 0.30153901, 2.06281317, ..., 0.16626766, 0.2061288 ,
        1.52011013]])
```

Label Encoding

Note: Here since the output(dependent) variables are categorial and mutli class we need to perform label encoding instead of one hot encoding.
source:https://stackoverflow.com/a/54617927

Source:https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

```
# Performing label encododing for both genotype class and Treatment Class.
le = LabelEncoder()
encoded_genotype = le.fit_transform(data.iloc[:,-2].values)
#print(encoded_genotype)

le1 = LabelEncoder()
encoded_Treatment = le1.fit_transform(data.iloc[:,-1].values)
```

```
#print(encoded_Treatment)
```

Feature Scaling

```
# Performing Feature Scaling to our imputed data.
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
features = sc.fit_transform(imputed_features)
print(features.shape)
```

```
(762, 67)
```

```
print(features)
```

```
[[ 0.23974473  2.37245233  1.47192076 ... -1.24854766 -1.44409891
   0.94304603]
 [ 0.27964803  1.97669047  1.39367543 ... -1.17890625 -1.39237984
   1.14476809]
 [ 0.25988768  2.11728651  1.09512379 ... -1.21550682 -1.45740344
   1.68743553]
 ...
 [-0.20194398 -0.02792479 -0.70004734 ... -0.57884072 -0.33275792
   0.22429103]
 [-0.26216375 -0.46988044 -0.89575533 ... -1.06251856  0.06728465
   0.55009802]
 [-0.26222495 -0.39361442 -0.72685472 ... -0.52379275 -0.06973267
   0.48133992]]
```

```
# Concatenating genotype class with features.
fi=np.concatenate((features,np.reshape(encoded_genotype, (-1, 1))), axis=1)
data_encode=pd.DataFrame(fi,columns=np.array(data.columns)[:-1])
display(data_encode)
```

	DYRK1A_N	BDNF_N	NR1_N	pAKT_N	pBRAF_N	pCAMKII_N	pCREB_N	pELK_
0	0.239745	2.372452	1.471921	-0.351734	-0.186528	-0.898820	0.648744	0.59521

10-Fold cross-validation

3	0.015969	0.833934	0.452418	-0.249043	-0.228099	-1.073520	-0.163587	0.2898(

```
def cross_validation(model, data, output):
  scores = ['accuracy', 'precision', 'recall', 'f1']
  results = cross_validate(estimator=model,
                           X=data,
                            y=output,
                            cv=10,
                            scoring=scores,
                            return_train_score=True)
  return results


def training(x_train, y_train,x_test,y_test, fold_no,model):
  model.fit(x_train, y_train)
  predicted = model.predict(x_test)
  return f1_score(y_test,predicted,average='weighted')


def cross_val_pre(model, data, output):
  e=[]
  skf = StratifiedKFold(n_splits=10)
  fold_no = 1
  for train_index,test_index in skf.split(data, output):
    data_train = data[train_index]
    data_test = data[test_index]
    output_train = output[train_index]
    output_test = output[test_index]
    fold_no += 1
    e.append(training(data_train,output_train,data_test,output_test, fold_no,model))
  return np.array(e)
```

Estimation of performance metric for the class genotype

```
c=0
for i in encoded_genotype:
  if i==0:
    c=c+1
print((c/len(encoded_genotype))*100)
```

```
57.08661417322835
```

Estimation of performance metric for the class Treatment_Behavior

```
a=[0,0,0,0]
for i in encoded_Treatment:
  if i==0:
    a[0]=a[0]+1
  if i==1:
    a[1]=a[1]+1
  if i==2:
    a[2]=a[2]+1
  else:
    a[3]=a[3]+1
print((a[3]/sum(a))*100)
```

```
48.01980198019802
```

Observations:

1. **For class genotype**

- We can say that "Accuracy" is a good measure because the target variable class (Genotype) in the data are nearly balanced. In this case it is 57% and 43%

2. **For class Treatment_Behavior**

- From the above code we can see that the we have a high class imbalance so we choose "f1 score" as the performance metric.
  source:https://towardsdatascience.com/comprehensive-guide-on-multiclass-classification-metrics-af94cfb83fbd

---

Elastic Net Logistic Regression Using SGD Classifier

It implements logistic regression with elastic net penalty.

For "Genotype" class

```
logistic_regression= linear_model.SGDClassifier(penalty='elasticnet' , alpha = 0.1 )
ans=cross_validation(logistic_regression, features, encoded_genotype)
print("The Training-Accuracy scores are ",ans['train_accuracy'])
print("The Mean Traning Accuracy is ",ans['train_accuracy'].mean()*100)
print("The Validation-Accuracy scores are ",ans["test_accuracy"])
print("The Mean Validation Accuracy is ",ans["test_accuracy"].mean()*100)
```

```
The Training-Accuracy scores are  [0.94014599 0.95328467 0.97230321 0.94314869 0.9606
 0.94460641 0.95043732 0.94752187 0.94460641]
The Mean Traning Accuracy is  95.08591006788534
The Validation-Accuracy scores are  [1.         0.8961039  0.73684211 0.81578947 0.67
 0.77631579 0.80263158 0.96052632 0.81578947]
The Mean Validation Accuracy is  83.6978810663021
```

For "Treatment_Behavior" class

```
logistic_regression= linear_model.SGDClassifier(penalty='elasticnet', alpha = 0.0001)
ans=cross_val_pre(logistic_regression, features, encoded_Treatment)
print("The validation f1 scores are ",ans)
print("The Mean validation f1 score is ",ans.mean()*100)
```

```
The validation f1 scores are  [0.67176765 0.80589178 0.94807312 0.82627473 0.9737813
 0.69455546 0.72350146 0.80364372 0.66778789]
The Mean validation f1 score is  80.10773587952708
```

SVM

SVM with "rbf" kernel

For "genotype" class

```
sv1=svm.SVC(kernel='rbf', C = 10)
ans=cross_validation(sv1, features, encoded_genotype)
print("The Traning-Accuracy scores are ",ans['train_accuracy'])
print("The Mean Traning Accuracy is ",ans['train_accuracy'].mean()*100)
print("The Validation-Accuracy scores are ",ans["test_accuracy"])
print("The Mean Validation Accuracy is ",ans["test_accuracy"].mean()*100)
```

```
The Traning-Accuracy scores are  [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

```
The Mean Traning Accuracy is  100.0
The Validation-Accuracy scores are  [0.98701299 0.90909091 0.73684211 0.76315789 0.8:
 0.65789474 0.81578947 1.         0.65789474]
The Mean Validation Accuracy is  80.01367053998634
```

For "Treatment_Behavior" class

```
sv2=svm.SVC(kernel='rbf', C =10)
ans=cross_val_pre(sv2, features, encoded_Treatment)
print("The validation f1 scores are ",ans)
print("The Mean validation f1 score is ",ans.mean()*100)
```

```
The validation f1 scores are  [0.72277056 0.81753247 0.94618421 0.77149208 0.9065169!
 0.89261031 0.8611336  0.960401   0.60946637]
The Mean validation f1 score is  84.48099418274987
```

SVM with linear kernel

For "genotype" class

```
sv3=svm.SVC(kernel='linear' , C =0.01 )
ans=cross_validation(sv3, features, encoded_genotype)
print("The Traning-Accuracy scores are ",ans['train_accuracy'])
print("The Mean Traning Accuracy is ",ans['train_accuracy'].mean()*100)
print("The Validation-Accuracy scores are ",ans["test_accuracy"])
print("The Mean Validation Accuracy is ",ans["test_accuracy"].mean()*100)
```

```
The Traning-Accuracy scores are  [0.94890511 0.95474453 0.97959184 0.95043732 0.9650:
 0.94460641 0.95189504 0.94897959 0.95043732]
The Mean Traning Accuracy is  95.45049051946117
The Validation-Accuracy scores are  [1.         0.85714286 0.71052632 0.84210526 0.6!
 0.76315789 0.80263158 0.97368421 0.77631579]
The Mean Validation Accuracy is  81.72932330827068
```

For "Treatment_Behavior" class

```
sv4=svm.SVC(kernel = 'rbf' , gamma = 'scale',  C = 10)
ans=cross_val_pre(sv4, features, encoded_Treatment)
print("The validation f1 scores are ",ans)
print("The Mean validation f1 score is ",ans.mean()*100)
```

```
The validation f1 scores are  [0.72277056 0.81753247 0.94618421 0.77149208 0.9065169!
 0.89261031 0.8611336  0.960401   0.60946637]
The Mean validation f1 score is  84.48099418274987
```

RandomForestClassifier

For "GenoType" Class

```
rfc1 = RandomForestClassifier(max_depth=5 ,n_estimators = 100 )
ans=cross_validation(rfc1, features, encoded_genotype)
print("The Traning-Accuracy scores are ",ans['train_accuracy'])
print("The Mean Traning Accuracy is ",ans['train_accuracy'].mean()*100)
print("The Validation-Accuracy scores are ",ans["test_accuracy"])
print("The Mean Validation Accuracy is ",ans["test_accuracy"].mean()*100)
```

```
The Traning-Accuracy scores are  [0.99416058 0.99562044 0.99708455 0.98979592 0.9985
 0.99271137 0.98979592 0.98979592 0.97521866]
The Mean Traning Accuracy is  99.12521546679152
The Validation-Accuracy scores are  [0.97402597 0.7012987  0.67105263 0.80263158 0.8
 0.52631579 0.85526316 0.85526316 0.53947368]
The Mean Validation Accuracy is  72.93745727956255
```

For "Treatment_Behaviour" Class

```
rfc2 = RandomForestClassifier(max_depth=10, n_estimators= 10)
ans=cross_val_pre(rfc2, features, encoded_Treatment)
print("The validation f1 scores are ",ans)
print("The Mean validation f1 score is ",ans.mean()*100)
```

```
The validation f1 scores are  [0.66440667 0.58265562 0.80653179 0.72405496 0.9052897
 0.69684527 0.47201669 0.70562057 0.60122763]
The Mean validation f1 score is  69.31204410355629
```

Neural Network

For Genotype Class

```
from sklearn import neural_network
nn1 = neural_network.MLPClassifier(hidden_layer_sizes= (100,), activation = "relu" , alpha
ans = cross_validation(nn1, features,encoded_genotype)
print("The Traning-Accuracy scores are ",ans['train_accuracy'])
```

```
print("The Mean Traning Accuracy is ",ans['train_accuracy'].mean()*100)
print("The Validation-Accuracy scores are ",ans["test_accuracy"])
print("The Mean Validation Accuracy is ",ans["test_accuracy"].mean()*100)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
The Traning-Accuracy scores are  [0.99416058 0.99562044 0.99854227 0.99562682 0.9956
 0.99562682 0.9941691  0.99562682 0.99708455]
The Mean Traning Accuracy is  99.5771105105233
The Validation-Accuracy scores are  [1.         0.8961039  0.72368421 0.84210526 0.8
 0.60526316 0.80263158 1.         0.69736842]
The Mean Validation Accuracy is  80.53998632946
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
```

For "Treatment_Behaviour" Class

```
from sklearn import neural_network
nn2 = neural_network.MLPClassifier(hidden_layer_sizes=(100,), activation = "relu",alpha =
ans1 = cross_val_pre(nn2, features , encoded_Treatment)
print("The validation f1 scores are ",ans1)
print("The Mean validation f1 score is ",ans1.mean()*100)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
The validation f1 scores are  [0.72239401 0.83157443 0.94618421 0.84945609 0.9208351
 0.85052697 0.87757365 0.93331976 0.65843237]
The Mean validation f1 score is  85.36568556993365
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
```

Feature Importance

source:https://inria.github.io/scikit-learn-mooc/python_scripts/dev_features_importance.html

```
def plot_feature_importances(perm_importance_result, feat_name):
    fig, ax = plt.subplots(figsize=(22,16))
    indices = perm_importance_result['importances_mean'].argsort()
    plt.barh(range(len(indices)),
             perm_importance_result['importances_mean'][indices])

    ax.set_yticks(range(len(indices)))
    _ = ax.set_yticklabels(feat_name[indices])
def get_feature_importance(model,X_train,y_train):
  perm_importance_result_train = permutation_importance(model, X_train, y_train, n_repeats
  plot_feature_importances(perm_importance_result_train, X_train.columns)
```

Feature importance for Random forest Classifier

```
sort = rfc2.feature_importances_.argsort()
fig , ax = plt.subplots(figsize=(22,16))
plt.barh(data.columns[sort], rfc2.feature_importances_[sort])
plt.xlabel("Feature Importance for random forest")
```

```
Text(0.5, 0, 'Feature Importance for random forest')
```

Feature importance for Elastic net logistic regression


```
logistic_regression.fit(data_encode.iloc[:,:-1],data_encode.iloc[:,-1:])
get_feature_importance(logistic_regression,data_encode.iloc[:,:-1],data_encode.iloc[:,-1:]
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversic
  y = column_or_1d(y, warn=True)
```

Feature importance for SVM

```
sv3.fit(data_encode.iloc[:,:-1],data_encode.iloc[:,-1:])
get_feature_importance(sv3,data_encode.iloc[:,:-1],data_encode.iloc[:,-1:])
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversic
  y = column_or_1d(y, warn=True)
```

Feature importance for Neural Network

```
nn2.fit(data_encode.iloc[:,:-1],data_encode.iloc[:,-1:])
get_feature_importance(nn2,data_encode.iloc[:,:-1],data_encode.iloc[:,-1:])
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
```

Importing the Test Data

```
# Importing the Test data Set Provided into Goggle colab.
from google.colab import files
data1 = pd.read_csv('https://www.ee.iitb.ac.in/~asethi/Dump/MouseTest.csv')
for fn in data1.keys():
  print('You uploaded "{name}" with length "{length}" bytes'.format(name=fn, length=len(da
```

```
You uploaded "DYRK1A_N" with length "225" bytes
You uploaded "ITSN1_N" with length "225" bytes
You uploaded "BDNF_N" with length "225" bytes
You uploaded "NR1_N" with length "225" bytes
You uploaded "NR2A_N" with length "225" bytes
You uploaded "pAKT_N" with length "225" bytes
You uploaded "pBRAF_N" with length "225" bytes
You uploaded "pCAMKII_N" with length "225" bytes
You uploaded "pCREB_N" with length "225" bytes
You uploaded "pELK_N" with length "225" bytes
You uploaded "pERK_N" with length "225" bytes
You uploaded "pJNK_N" with length "225" bytes
You uploaded "PKCA_N" with length "225" bytes
You uploaded "pMEK_N" with length "225" bytes
You uploaded "pNR1_N" with length "225" bytes
You uploaded "pNR2A_N" with length "225" bytes
You uploaded "pNR2B_N" with length "225" bytes
You uploaded "pPKCAB_N" with length "225" bytes
You uploaded "pRSK_N" with length "225" bytes
You uploaded "AKT_N" with length "225" bytes
You uploaded "BRAF_N" with length "225" bytes
You uploaded "CAMKII_N" with length "225" bytes
You uploaded "CREB_N" with length "225" bytes
You uploaded "ELK_N" with length "225" bytes
```

```
You uploaded "ERK_N" with length "225" bytes
You uploaded "GSK3B_N" with length "225" bytes
You uploaded "JNK_N" with length "225" bytes
You uploaded "MEK_N" with length "225" bytes
You uploaded "TRKA_N" with length "225" bytes
You uploaded "RSK_N" with length "225" bytes
You uploaded "APP_N" with length "225" bytes
You uploaded "Bcatenin_N" with length "225" bytes
You uploaded "SOD1_N" with length "225" bytes
You uploaded "MTOR_N" with length "225" bytes
You uploaded "P38_N" with length "225" bytes
You uploaded "pMTOR_N" with length "225" bytes
You uploaded "DSCR1_N" with length "225" bytes
You uploaded "AMPKA_N" with length "225" bytes
You uploaded "NR2B_N" with length "225" bytes
You uploaded "pNUMB_N" with length "225" bytes
You uploaded "RAPTOR_N" with length "225" bytes
You uploaded "TIAM1_N" with length "225" bytes
You uploaded "pP70S6_N" with length "225" bytes
You uploaded "NUMB_N" with length "225" bytes
You uploaded "P70S6_N" with length "225" bytes
You uploaded "pGSK3B_N" with length "225" bytes
You uploaded "pPKCG_N" with length "225" bytes
You uploaded "CDK5_N" with length "225" bytes
You uploaded "S6_N" with length "225" bytes
You uploaded "ADARB1_N" with length "225" bytes
You uploaded "AcetylH3K9_N" with length "225" bytes
You uploaded "RRP1_N" with length "225" bytes
You uploaded "BAX_N" with length "225" bytes
You uploaded "ARC_N" with length "225" bytes
You uploaded "ERBB4_N" with length "225" bytes
You uploaded "nNOS_N" with length "225" bytes
You uploaded "Tau_N" with length "225" bytes
You uploaded "GFAP_N" with length "225" bytes
```

Removing the correlated features from the test Data

```
#display(data1)
data1= data1.drop(labels=rm , axis=1 ) # rm = correlated Features.
print(data1.shape)
# test_data = test_data.drop(labels= rm , axis =1 )
test_data = data1.iloc[:,:-2].values
print(test_data.shape)
```

```
(225, 69)
(225, 67)
```

Taking care of Missing Data(Multivariate feature imputation)

```
imputer = IterativeImputer(max_iter=10, random_state=0)
imputer.fit(test_data)
imputed_features_test_data= imputer.transform(test_data)
print(imputed_features_test_data)
```

```
[[0.62758167 0.44695832 2.93071724 ... 0.15810257 0.13332723 1.64998272]
 [0.65125332 0.46496392 2.99268895 ... 0.16340471 0.1261546  1.63898847]
 [0.64434594 0.47005565 3.07384702 ... 0.1602933  0.11253562 1.56492482]
 ...
 [0.21613419 0.24169329 1.33083067 ... 0.25700365 0.30481121 1.14220463]
 [0.21236948 0.23903615 1.34682731 ... 0.2558521  0.30848367 1.11271806]
 [0.25728988 0.28968935 1.58928912 ... 0.27804804 0.32608696 1.17178474]]
```

Feature Scaling

```
# Doing same Feature Scaling that is done on train data.
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
test_features = sc.fit_transform(imputed_features_test_data)
```

```
print(test_features)
print(test_features.shape)
```

```
[[ 1.10080345  2.58325898  1.97470136 ... -0.6721695  -1.18262182
   1.20881331]
 [ 1.22539199  2.91562998  2.14458551 ... -0.56441819 -1.32608984
   1.16819426]
 [ 1.18903708  3.00961973  2.36706557 ... -0.62764902 -1.59849872
   0.89456087]
 ...
 [-1.06472568 -1.20579214 -2.41109589 ...  1.3377223   2.24742698
  -0.66720873]
 [-1.08454007 -1.25484123 -2.36724402 ...  1.31432004  2.32088425
  -0.77614895]
 [-0.84811515 -0.319818   -1.70257919 ...  1.7653914   2.67298781
  -0.55792293]]
(225, 67)
```

Predicting output(encoded_genotype) by Elastic Log. Regression

```
logistic_regression.fit(features,encoded_genotype)
encoded_genotype_output_EN =logistic_regression.predict(test_features)
genotype_output_EN =le.inverse_transform(encoded_genotype_output_EN)
print(genotype_output_EN)
res=genotype_output_EN==np.reshape(data1.iloc[:,-2].values,-1)
c=0
for i in res:
```

```
  if i:
    c=c+1
print("")
print("Accuracy is ",c/len(res)*100)
```

```
['Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Control' 'Control' 'Control' 'Ts65Dn'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Ts65Dn' 'Ts65Dn' 'Control' 'Control' 'Ts65Dn' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Ts65Dn' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Control' 'Ts65Dn' 'Ts65Dn' 'Control' 'Ts65Dn' 'Control'
 'Control' 'Ts65Dn' 'Ts65Dn' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Ts65Dn' 'Control' 'Control' 'Control'
 'Control' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Control' 'Control' 'Ts65Dn' 'Control' 'Ts65Dn'
 'Control' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Control' 'Ts65Dn' 'Ts65Dn' 'Control' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Control' 'Control' 'Ts65Dn']

Accuracy is  76.44444444444444
```

Predicting output(encoded_genotype) for rbf SVM kernel

```
sv1.fit(features,encoded_genotype)
encoded_genotype_output =sv1.predict(test_features)
genotype_output =le.inverse_transform(encoded_genotype_output)
print(genotype_output)
res=genotype_output==np.reshape(data1.iloc[:,-2].values,-1)
c=0
for i in res:
  if i:
    c=c+1
print("")
print("Accuracy is ",c/len(res)*100)
```

```
['Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Ts65Dn' 'Control' 'Control' 'Control' 'Ts65Dn' 'Control' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Control' 'Control' 'Ts65Dn' 'Control' 'Control'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Control' 'Control' 'Control' 'Control' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn']

Accuracy is  79.55555555555556
```

```
# test_given_genotype= data1.iloc[:,-2].values
# encoded_given_test_genotype =le.fit_transform(test_given_genotype)
# plt.plot(encoded_given_test_genotype ,encoded_genotype_output)
# print(len(genotype_output))
# print(len(data1.iloc[:,-2].values))
# plt.show()
```

Predicting output(encoded_treatment) for linear SVM kernal

```
sv4.fit(features,encoded_Treatment)
encoded_Treatment_output = sv4.predict(test_features)
Treatment_output = le1.inverse_transform(encoded_Treatment_output)
print(Treatment_output)
res=Treatment_output==np.reshape(data1.iloc[:,-1:].values,-1)
c=0
for i in res:
```

```
    if i:
      c=c+1
  print("")
  print("Accuracy is ",c/len(res)*100)
```

```
['Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Saline_S/C' 'Saline_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_C/S' 'Saline_C/S' 'Saline_C/S'
 'Memantine_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Memantine_C/S'
 'Saline_C/S' 'Memantine_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_S/C' 'Saline_S/C'
 'Memantine_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Memantine_C/S'
 'Saline_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Saline_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Saline_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Memantine_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C']

Accuracy is  87.55555555555556
```

Predicting output(encoded_treatment) for Neural Network

```
nn2.fit(features, encoded_Treatment)
encoded_Treatment_output_nn = nn2.predict(test_features)
Treatment_output_nn = le1.inverse_transform(encoded_Treatment_output_nn)
print(Treatment_output_nn)
res=Treatment_output_nn==np.reshape(data1.iloc[:,-1:].values,-1)
c=0
for i in res:
  if i:
    c=c+1
print("")
print("Accuracy is ",c/len(res)*100)
```

```
['Memantine_C/S' 'Memantine_C/S' 'Saline_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Saline_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Saline_C/S' 'Memantine_S/C' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Memantine_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_S/C' 'Saline_S/C' 'Memantine_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Memantine_S/C' 'Saline_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Saline_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Saline_S/C' 'Memantine_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Memantine_S/C' 'Saline_S/C' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
```

```
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Memantine_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C']

Accuracy is  86.22222222222223
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_percept
  ConvergenceWarning,
```

Predicting output(encoded_genotype) for Neural Network

```
nn1.fit(features, encoded_genotype)
encoded_genotype_output_nn = nn1.predict(test_features)
genotype_output_nn = le.inverse_transform(encoded_genotype_output_nn)
print(genotype_output_nn)
res=genotype_output_nn==np.reshape(data1.iloc[:,-2].values,-1)
c=0
for i in res:
  if i:
    c=c+1
print("")
print("Accuracy is ",c/len(res)*100)
```

```
['Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Ts65Dn' 'Control' 'Control'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Control' 'Ts65Dn' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Control' 'Ts65Dn' 'Ts65Dn' 'Control' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Control' 'Ts65Dn' 'Ts65Dn' 'Control' 'Control'
```

```
 'Control' 'Control' 'Control' 'Control' 'Control' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Control'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Control' 'Ts65Dn' 'Ts65Dn']

Accuracy is  78.22222222222223
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron
  ConvergenceWarning,
```

Predicting output(encoded_genotype) for Random Forest

```
rfc1.fit(features, encoded_genotype)
encoded_genotype_output_rfc1 = rfc1.predict(test_features)
genotype_output_rfc1 = le.inverse_transform(encoded_genotype_output_rfc1)
print(genotype_output_rfc1)
res=genotype_output_rfc1==np.reshape(data1.iloc[:,-2].values,-1)
c=0
for i in res:
  if i:
    c=c+1
print("")
print("Accuracy is ",c/len(res)*100)
```

```
['Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Control' 'Ts65Dn' 'Ts65Dn'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Ts65Dn' 'Ts65Dn' 'Control' 'Ts65Dn'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Control' 'Ts65Dn' 'Ts65Dn' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Control' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Control' 'Control' 'Control' 'Control' 'Control' 'Control'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
```

```
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn'
 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn' 'Ts65Dn']

Accuracy is  72.0
```

Predicting output(encoded_Treatment) for Random Forest

```
rfc2.fit(features, encoded_Treatment)
encoded_Treatment_output_rfc2 = rfc2.predict(test_features)
Treatment_output_rfc2 = le1.inverse_transform(encoded_Treatment_output_rfc2)
print(Treatment_output_rfc2)
res=Treatment_output_rfc2==np.reshape(data1.iloc[:,-1:].values,-1)
c=0
for i in res:
  if i:
    c=c+1
print("")
print("Accuracy is ",c/len(res)*100)
```

```
['Memantine_C/S' 'Saline_C/S' 'Saline_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Saline_C/S' 'Memantine_C/S' 'Saline_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Saline_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Saline_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_C/S' 'Memantine_S/C' 'Memantine_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Memantine_C/S' 'Memantine_C/S' 'Saline_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Saline_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Saline_C/S' 'Saline_C/S' 'Memantine_C/S'
 'Saline_C/S' 'Saline_C/S' 'Memantine_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Memantine_C/S' 'Saline_C/S'
 'Saline_C/S' 'Memantine_C/S' 'Saline_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Saline_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Saline_S/C'
 'Saline_S/C' 'Memantine_S/C' 'Saline_S/C' 'Saline_S/C' 'Memantine_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Memantine_S/C'
 'Saline_S/C' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Saline_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Saline_C/S' 'Memantine_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
```

```
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S'
 'Memantine_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Memantine_S/C' 'Memantine_S/C' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Saline_C/S' 'Memantine_C/S'
 'Saline_C/S' 'Memantine_C/S' 'Memantine_C/S' 'Saline_C/S' 'Saline_C/S'
 'Saline_C/S' 'Saline_C/S' 'Saline_S/C' 'Memantine_S/C' 'Memantine_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C' 'Saline_S/C'
 'Saline_S/C' 'Saline_S/C']

Accuracy is  68.0
```

Observations after prediction

1. Predicting the genotype class(binary) output by Elastic net Log. regression gives accuracy of 76 %
2. Predicting the Treatment class output by Elastic net Log. regression gives accuracy of 78 %
3. Predicting the genotype class(binary) output by rbf SVM kernel gives accuracy of 79.5 %.
4. Predicting the Treatment class output by linear SVM kernel gives accuracy of 87.5 %.
5. Predicting the genotype class(binary) output by Neural Network gives accuracy of 78.24%.
6. Predicting the Treatment class output by Neural Network gives accuracy of 87.5%.
7. Predicting the genotype class(binary) output by Random Forest gives accuracy of 73.7%
8. Predicting the Treatment class output by Random forest gives accuracy of 77.5%

Conclusion-

Neural Network with single ReLU hidden layer fits perfectly in this data set and gives better results among other models.

IIT Bombay

213070024

Home Logout IITB Website

Roll no	213070024	**Name**	Abhishek Verma
Department	Electrical Engineering	**Program**	M.Tech (Control and Computing)

Performance Summary | Graduation Requirements | Personal Information | Forms/Requests | Payment

Academic Performance Summary

Year	Sem	SPI	CPI	Sem Credits Used for SPI	Completed Semester Credits	Cumulative Credits Used for CPI	Completed Cumulative Credits
2021	Spring	8.36	8.19	28.0	28.0	52.0	52.0
2021	Autumn	8.0	8.0	24.0	24.0	24.0	24.0

Semester-wise Details

*This registration is subject to approval(s) from faculty advisor/Course Instructor/Academic office.

Year/Semester: 2022-23/Autumn

Course Code	Course Name	Credits	Tag	Grade	Credit/Audit
SC 649	Embedded Control & Robotics	6.0	Department elective	Not allotted	C

Year/Semester: 2022-23/Project

Course Code	Course Name	Credits	Tag	Grade	Credit/Audit
EE 797	I Stage Project	42.0	Core course	Not allotted	C

Year/Semester: 2021-22/Spring

Course Code	Course Name	Credits	Tag	Grade	Credit/Audit
EE 613	Nonlinear Dynamical Systems	6.0	Core course	AB	C
EE 622	Optimal Control Systems	6.0	Core course	BB	C
EE 636	Matrix Computations	6.0	Core course	BB	C
EE 694	Seminar	4.0	Core course	AB	C
EE 769	Introduction to Machine Learning	6.0	Department elective	BB	C

GC 101	Gender in the workplace	0.0	Core course	PP	N

Year/Semester: 2021-22/Autumn

Course Code	Course Name	Credits	Tag	Grade	Credit/Audit
EE 601	Statistical Signal Analysis	6.0	Core course	BB	C
EE 615	Control and Computational Laboratory	6.0	Core course	AB	C
EE 635	Applied Linear Algebra	6.0	Core course	BC	C
EE 640	Multivariable Control Systems	6.0	Core course	BB	C
EE 899	Communication Skills	6.0	Core course	PP	N