```
import re
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.decomposition import PCA
from sklearn.decomposition import SparsePCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import BernoulliNB,GaussianNB
from imblearn.over_sampling import SMOTE
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import random
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import TruncatedSVD
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix,f1_score,make_scorer
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest
from sklearn.impute import SimpleImputer
from sklearn.model_selection import ShuffleSplit,cross_val_score


#Read Train data
with open("/content/drive/MyDrive/Train.txt", "r") as tr:
    Train = tr.readlines()
#read Test data
with open("/content/drive/MyDrive/Test.txt", "r") as te:
    Test = te.readlines()
print (len(Train))


     800


drug_active = []
train_features = []
test_features = []
```

```
for i in range(len(Train)):
    lines = Train[i]
    for l in range(len(lines)):
        if l == 1:
            drug_active.append(lines[0:1])

#Split the train dataset features into list
for features in Train:
    features = features.replace("0\t", "")
    features = features.replace("1\t", "")
    features = features.replace("\n", "")
    features = features.split()
    train_features.append(features)

#Split the test dataset features into list
for features in Test:
    features = features.replace("0\t", "")
    features = features.replace("1\t", "")
    features = features.replace("\n", "")
    features = features.split()
    test_features.append(features)

print(len(train_features))
print(len(test_features))
print(len(drug_active))
```

```
800
350
800
```

```
train_sparse = []
total_features_count = 100001

for features in train_features:
  train_sparse_set = [0]*total_features_count
  for feature in features:
    train_sparse_set[int(feature)] = 1
  train_sparse.append(train_sparse_set)
```

```
test_sparse = []

for features in test_features:
  test_sparse_set = [0]*total_features_count
  for feature in features:
    test_sparse_set[int(feature)] = 1
  test_sparse.append(test_sparse_set)

print(len(train_sparse[0]))
x_train, x_test, y_train, y_test = train_test_split(train_sparse,drug_active,test_size=0.20,random_state=42)
```

```
100001
```

```
chi2_selection = SelectKBest(chi2, k=300)
train_sparse_1=chi2_selection.fit_transform(x_train,y_train)
test_sparse_1 = chi2_selection.transform(x_test)
train_sparse_final=chi2_selection.fit_transform(train_sparse,drug_active)
test_sparse_final = chi2_selection.transform(test_sparse)
print(train_sparse_1.shape)
print(test_sparse_1.shape)
```

```
(640, 300)
(160, 300)
```

```
smote_sm = SMOTE(random_state = 42)
train_sparse_sm_1,drug_active_sm_1 = smote_sm.fit_resample(train_sparse_final,drug_active)
train_sparse_sm,drug_active_sm = smote_sm.fit_resample(x_train,y_train)
print(len(train_sparse_sm))
print(len(train_sparse_sm[0]))
```

```
1152
100001
```

```
svd = TruncatedSVD(n_components=350)
svd1 = TruncatedSVD(n_components=350)
```

```
pca1 = PCA(n_components = 0.95)
sparse_train_svd = svd1.fit_transform(train_sparse,drug_active)
train_svd1 =  svd1.transform(x_train)
test_svd1 = svd1.transform(x_test)
sparse_test_svd = svd1.transform(test_sparse)

sparse_train_pca = pca1.fit_transform(train_sparse,drug_active)
train_pca1 =  pca1.transform(x_train)
test_pca1 = pca1.transform(x_test)
sparse_test_pca = pca1.transform(test_sparse)



sparse_svd = svd.fit_transform(train_sparse_sm,drug_active_sm)
pca = PCA(n_components = 0.95)
sparse_pca = pca.fit_transform(train_sparse_sm)


train_svd = svd.transform(x_train)
#print(train_svd)
train_pca = pca.transform(x_train)
print(train_pca)



test_svd = svd.transform(x_test)
#print(test_svd)
test_pca = pca.transform(x_test)
#print(test_pca)

test_sparse_svd = svd.transform(test_sparse)
test_sparse_pca = pca.transform(test_sparse)

#print(sparse_svd.shape)
print(train_svd.shape)
print(test_svd.shape)
print(test_sparse_svd.shape)
#print(test_sparse_svd.shape)
#print(sparse_pca.shape)
#print(train_pca.shape)
#print(test_pca.shape)
```

```
#print(test_sparse_pca.shape)
print(svd.explained_variance_ratio_.sum())
print(svd1.explained_variance_ratio_.sum())
```

```
     [[ 0.86343415 -0.9389797  -1.8621428  ... -0.07786184 -1.40887876
        0.27059813]
      [ 0.23559506 -0.46630539 -1.99946801 ...  1.12678214  0.73971695
       -0.70937452]
      [ 0.22928774 -0.56249729 -1.77969048 ...  0.07219735 -0.47950398
       -0.16373381]
      ...
      [ 0.49479459 -0.65691126 -2.31260623 ...  0.50008196  0.43206119
        1.51974749]
      [ 1.56153352 -1.6991631  -1.38846366 ...  0.72498023  0.69248242
       -0.28803238]
      [ 0.25934995 -0.33730987 -2.44705785 ...  0.59846312 -0.20847595
        0.25736581]]
     (640, 350)
     (160, 350)
     (350, 350)
     0.6364965222143759
     0.552302639357118
```

```
dt_model=DecisionTreeClassifier()
grid_params={'criterion':['gini','entropy'],'min_samples_split':[2,3,4,5,6,7,8]}
grid=GridSearchCV(dt_model,grid_params,cv=5,scoring='f1_macro')
grid.fit(train_pca,y_train)
```

```
     GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
                  param_grid={'criterion': ['gini', 'entropy'],
                              'min_samples_split': [2, 3, 4, 5, 6, 7, 8]},
                  scoring='f1_macro')
```

```
dt_model1=DecisionTreeClassifier()
grid_params={'criterion':['gini','entropy'],'min_samples_split':[2,3,4,5,6,7,8]}
grid1=GridSearchCV(dt_model1,grid_params,cv=5,scoring='f1_macro')
grid1.fit(train_svd,y_train)
```

```
  GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
                  param_grid={'criterion': ['gini', 'entropy'],
```

```
                            'min_samples_split': [2, 3, 4, 5, 6, 7, 8]},
                   scoring='f1_macro')



print("The best score is: ",grid.best_score_)
print("The best Parameters are: ",grid.best_params_)
print("The best score is: ",grid1.best_score_)
print("The best Parameters are: ",grid1.best_params_)

     The best score is:  0.7446101632018353
     The best Parameters are:  {'criterion': 'entropy', 'min_samples_split': 2}
     The best score is:  0.7396086483824178
     The best Parameters are:  {'criterion': 'gini', 'min_samples_split': 5}


naive = BernoulliNB()
naive.fit(train_pca,y_train)
naive_pred=naive.predict(test_pca)
naive_acc=accuracy_score(y_test,naive_pred)*100
print("The training accuracy for decision tree model is: ",naive_acc)
print(classification_report(y_test,naive_pred))

     The training accuracy for decision tree model is:  96.25
                   precision    recall  f1-score   support

                0       0.97      0.99      0.98       146
                1       0.90      0.64      0.75        14

         accuracy                           0.96       160
        macro avg       0.93      0.82      0.86       160
     weighted avg       0.96      0.96      0.96       160



naive1 = BernoulliNB()
naive1.fit(train_svd,y_train)
naive_pred1=naive1.predict(test_svd)
naive_acc1=accuracy_score(y_test,naive_pred1)*100
print("The training accuracy for decision tree model is: ",naive_acc1)
print(classification_report(y_test,naive_pred1))
```

```
     The training accuracy for decision tree model is:  96.25
                   precision    recall  f1-score   support

                0       0.97      0.99      0.98       146
                1       0.83      0.71      0.77        14

            accuracy                           0.96       160
           macro avg       0.90      0.85      0.87       160
        weighted avg       0.96      0.96      0.96       160
```

```
naive_1 = GaussianNB()
naive_1.fit(train_svd,y_train)
naive_pred2=naive_1.predict(test_svd)
naive_acc_1=accuracy_score(y_test,naive_pred2)*100
print("The training accuracy for decision tree model is: ",naive_acc_1)
print(classification_report(y_test,naive_pred2))
```

```
     The training accuracy for decision tree model is:  91.875
                   precision    recall  f1-score   support

                0       0.92      1.00      0.96       146
                1       1.00      0.07      0.13        14

            accuracy                           0.92       160
           macro avg       0.96      0.54      0.55       160
        weighted avg       0.93      0.92      0.89       160
```

```
dt_test_final=naive.predict(test_sparse_pca)
dt_test_final1=naive1.predict(test_sparse_svd)
dt_test_final_1=naive_1.predict(test_sparse_svd)
```

```
dt_model=DecisionTreeClassifier(criterion='entropy',min_samples_split=8)
dt_model.fit(train_pca,y_train)
dt_train_pred=dt_model.predict(test_pca)
dt_acc=accuracy_score(y_test,dt_train_pred)*100
print("The training accuracy for decision tree model is: ",dt_acc)
print(classification_report(y_test,dt_train_pred))
```

```
The training accuracy for decision tree model is:  94.375
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       146
           1       0.69      0.64      0.67        14

    accuracy                           0.94       160
   macro avg       0.83      0.81      0.82       160
weighted avg       0.94      0.94      0.94       160
```

```
dt_model1=DecisionTreeClassifier(criterion='gini',min_samples_split=4)
print(train_svd.shape)
print(test_svd.shape)
dt_model1.fit(train_svd,y_train)
dt_train_pred1=dt_model1.predict(test_svd)
dt_acc=accuracy_score(y_test,dt_train_pred)*100
print("The training accuracy for decision tree model is: ",dt_acc)
print(classification_report(y_test,dt_train_pred1))
```

```
(640, 350)
(160, 350)
The training accuracy for decision tree model is:  94.375
              precision    recall  f1-score   support

           0       0.94      1.00      0.97       146
           1       1.00      0.29      0.44        14

    accuracy                           0.94       160
   macro avg       0.97      0.64      0.71       160
weighted avg       0.94      0.94      0.92       160
```

```
dt_test_pred=dt_model.predict(test_sparse_pca)
dt_test_pred1=dt_model1.predict(test_sparse_svd)


with open("HW2_Result3.txt", "w") as f:
    for s in dt_test_pred:
```

```
        f.write(str(s) +"\n")

with open("HW2_Result4.txt", "w") as f:
    for s in dt_test_pred1:
        f.write(str(s) +"\n")

with open("HW2_Result2.txt", "w") as f:
    for s in dt_test_final:
        f.write(str(s) +"\n")

with open("HW2_Result1.txt", "w") as f:
    for s in dt_test_final1:
        f.write(str(s) +"\n")

with open("HW2_Result5.txt", "w") as f:
    for s in dt_test_final_1:
        f.write(str(s) +"\n")


classifiers_list = {
    "Decision Tree": DecisionTreeClassifier(),
    "Naive Bayes : GaussianNB": GaussianNB(),
    "Naive Bayes : BernoulliNB": BernoulliNB()
}

classifiers_count = len(classifiers_list.keys())
df_results = pd.DataFrame(data=np.zeros(shape=(classifiers_count,5)), columns = ['classifier', 'Recall', 'F1', 'Precision', 'Accuracy'])

for c_name, classifier in classifiers_list.items():
  classifier.fit(train_svd, y_train)
  prediction = []
  prediction = classifier.predict(test_svd)
  cv1 = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
  scores = cross_val_score(classifier, train_svd, y_train, cv=cv1)
  print ('Classifier+Smote + SVD', c_name)
  print ('Cross validation', scores)
  print(classification_report(y_test,prediction))

  classifier.fit(train_pca, y_train)
```

```
prediction1 = []
prediction1 = classifier.predict(test_pca)
cv1 = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
scores = cross_val_score(classifier, train_pca, y_train, cv=cv1)
print ('Classifier +Smote + PCA', c_name)
print ('Cross validation', scores)
print(classification_report(y_test,prediction1))

classifier.fit(train_sparse_sm_1, drug_active_sm_1)
prediction2 = []
prediction2 = classifier.predict(test_sparse_1)
cv1 = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
scores = cross_val_score(classifier,train_sparse_sm_1 , drug_active_sm_1, cv=cv1)
print ('Classifier + Chi', c_name)
print ('Cross validation', scores)
print(classification_report(y_test,prediction2))

classifier.fit(train_svd1, y_train)
prediction3 = []
prediction3 = classifier.predict(test_svd1)
cv1 = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
scores = cross_val_score(classifier, train_svd1, y_train, cv=cv1)
print ('Classifier + SVD', c_name)
print ('Cross validation', scores)
print(classification_report(y_test,prediction3))

classifier.fit(train_pca1, y_train)
prediction4 = []
prediction4 = classifier.predict(test_pca1)
cv1 = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
scores = cross_val_score(classifier, train_pca1, y_train, cv=cv1)
print ('Classifier + PCA', c_name)
print ('Cross validation', scores)
print(classification_report(y_test,prediction4))
```

```
      macro avg       0.51      0.51      0.50       160
   weighted avg       0.84      0.76      0.80       160

   Classifier+Smote + SVD Naive Bayes : BernoulliNB
```

```
Cross validation [0.8984375 0.953125  0.8671875 0.9140625 0.90625  ]
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       146
           1       0.83      0.71      0.77        14

    accuracy                           0.96       160
   macro avg       0.90      0.85      0.87       160
weighted avg       0.96      0.96      0.96       160

Classifier +Smote + PCA Naive Bayes : BernoulliNB
Cross validation [0.953125  0.9453125 0.890625  0.9140625 0.921875 ]
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       146
           1       0.90      0.64      0.75        14

    accuracy                           0.96       160
   macro avg       0.93      0.82      0.86       160
weighted avg       0.96      0.96      0.96       160


Classifier + Chi Naive Bayes : BernoulliNB
Cross validation [0.74394464 0.6816609  0.70588235 0.73010381 0.73702422]
              precision    recall  f1-score   support

           0       0.99      0.95      0.97       146
           1       0.63      0.86      0.73        14

    accuracy                           0.94       160
   macro avg       0.81      0.90      0.85       160
weighted avg       0.95      0.94      0.95       160

Classifier + SVD Naive Bayes : BernoulliNB
Cross validation [0.9296875 0.953125  0.90625   0.9609375 0.9375   ]
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       146
           1       0.80      0.57      0.67        14

    accuracy                           0.95       160
   macro avg       0.88      0.78      0.82       160
weighted avg       0.95      0.95      0.95       160
```

```
Classifier + PCA Naive Bayes : BernoulliNB
Cross validation [0.9375    0.9609375 0.9140625 0.9453125 0.9296875]
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       146
           1       0.88      0.50      0.64        14

    accuracy                           0.95       160
   macro avg       0.91      0.75      0.80       160
weighted avg       0.95      0.95      0.94       160
```

```
naive_final = BernoulliNB()
print(train_sparse_sm_1.shape)
print(test_sparse_final.shape)
print(len(drug_active_sm_1))
naive_final.fit(train_svd1, y_train)
naive_pred_final=naive_final.predict(sparse_test_svd)
with open("HW2_Result_final_2.txt", "w") as f:
    for s in naive_pred_final:
        f.write(str(s) +"\n")
```

```
(1444, 300)
(350, 300)
1444
```

✓ 0s completed at 5:48 PM