COMP5318 - Machine Learning and Data Mining: Assignment 1

load data

In [29]:
```
import h5py
import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt
import time
print(os.listdir("./Input/train"))
print(os.listdir("./Input/test"))
```

```
['images_training.h5', 'labels_training.h5']
['images_testing.h5', 'labels_testing_2000.h5']
```

In [30]:
```
with h5py.File('./Input/train/images_training.h5','r') as H:
    data_train = np.copy(H['datatrain'])
with h5py.File('./Input/train/labels_training.h5','r') as H:
    label_train = np.copy(H['labeltrain'])
#load images_testing
with h5py.File('./Input/test/images_testing.h5','r') as H:
    datatest = np.copy(H['datatest'])
#load teating table which contain 2000 samples
with h5py.File('./Input/test/labels_testing_2000.h5','r') as H:
    labeltest = np.copy(H['labeltest'])

# using H['datatest'], H['labeltest'] for test dataset.
print(data_train.shape,label_train.shape)

print(datatest.shape,labeltest.shape)
```

```
(30000, 784) (30000,)
(5000, 784) (2000,)
```

Functions

In [31]:
```
def output(pred_data):
    #assume output is the predicted labels from classifiers
    # (5000,)
    with h5py.File('Output/predicted_labels.h5','w') as H:
        H.create_dataset('Output',data = pred_data)
```

In [32]:
```
#calculate the accuacy for the classifier
def cal_accuacy (pred_data, lable):
    count = 0

    for i in range (lable.shape[0]):
        if lable[i] == pred_data[i]:
            count +=1

    accuracy = count/lable.shape[0]
    return accuracy
```

Split both train datasets as train and validation (80%/20%)

In [33]:
```
from sklearn.model_selection import train_test_split

#apply splits function for the datasets
split_data_train, split_data_test,split_label_train, split_label_test = train_test_split(data_train,label_train,t
#check see if get target split
print(split_data_train.shape,split_data_test.shape)
print(split_label_train.shape,split_label_test.shape)
```

```
(24000, 784) (6000, 784)
(24000,) (6000,)
```

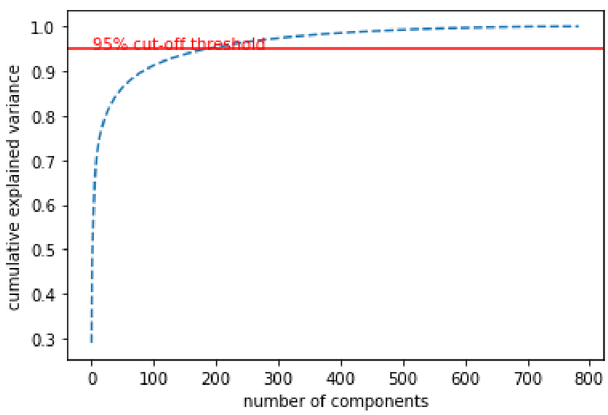
Data pre-processing

In [34]:
```
#use minmac scaler normalise the data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(-1,1))
scaler.fit(split_data_train)
data_train_norm = scaler.transform(split_data_train)
data_test_norm = scaler.transform(split_data_test)
#apply scaler for datatest
data_test_norm_og = scaler.transform(datatest)
```

In [35]:
```
from sklearn.decomposition import PCA
#find best components number,The redline represen 95% of  explained variance
pca_find = PCA().fit(split_data_train)
plt.plot(np.cumsum(pca_find.explained_variance_ratio_), linestyle='--',)

plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
plt.axhline(y=0.95, color='r', linestyle='-')
plt.text(0.5, 0.95, '95% cut-off threshold', color = 'red', fontsize=10)
```

Out[35]:
```
Text(0.5, 0.95, '95% cut-off threshold')
```



In [36]:
```
# apply PCA to splited data, test from 84 (90%) to 187(95%)
pca = PCA(n_components= 84)
pca_data_train = pca.fit_transform(data_train_norm)
pca_data_test = pca.transform(data_test_norm)
#apply pca for data_test_norm_og
pca_data_test_og = pca.transform(data_test_norm_og)
```

Classification algorithms

k-nearest neighbor classifier

In [37]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
#apply knn with 1 neighbours and test the splited raw data
knn = KNeighborsClassifier(n_neighbors=1)

#apply knn with 1 neighbours and test the PCA data

start = time.time()
#apply splited data_train and labl_train to train the data via knn
knn.fit(pca_data_train, split_label_train)
#make prediction for splicted data as reference
knn_res_pca = knn.predict(pca_data_test)
#make prediction for 5000 test data
knn_res_pca_og = knn.predict(pca_data_test_og)

end = time.time()
#print accuacy for splited test data
acc_knn_pca = cal_accuacy(knn_res_pca,split_label_test)

print(f"Time taken is {format(end-start, '.3f')} seconds")

print(f"When k = 1, knn's Accuracy result for train data is: {format(acc_knn_pca, '.3f')}")

acc_knn_pca_og = cal_accuacy(knn_res_pca_og,labeltest)

print(f"When k = 1, knn's Accuracy result for test data is: {format(acc_knn_pca_og, '.3f')}")
```

```
Time taken is 3.123 seconds
When k = 1, knn's Accuracy result for train data is: 0.840
When k = 1, knn's Accuracy result for test data is: 0.828
```

In [38]:
```
#find hyperparameter and use grid search with 10-fold stratified cross validation to find best performance
from sklearn.model_selection import GridSearchCV
k_range = list(range(1, 21, 2))
#create a parameter grid use to map the parameter names to the values
param_grid = dict(n_neighbors=k_range)
print(param_grid)
```

```
{'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]}
```

In [39]:
```
#create grid for knn

start = time.time()

grid_knn = GridSearchCV(knn, param_grid, cv=10, n_jobs=-1)
grid_knn.fit(pca_data_train, split_label_train)
#best score
print("The best train/train prediction score is:" )
print(grid_knn.best_score_)
#use grid search find best k calue in 2,4,6,8,10
print("The best k value is:" )
print(grid_knn.best_params_)

knn_res_pca = grid_knn.predict(pca_data_test)

knn_res_pca_og = grid_knn.predict(pca_data_test_og)

end = time.time()

print(f"Time taken is {format(end-start, '.3f')} seconds")

acc_knn_pca = cal_accuacy(knn_res_pca,split_label_test)

print(f"knn's best Accuracy result for train data is: {format(acc_knn_pca, '.3f')}")

acc_knn_pca_og = cal_accuacy(knn_res_pca_og,labeltest)

print(f"knn's Accuracy result for test data is: {format(acc_knn_pca_og, '.3f')}")
```

```
The best train/train prediction score is:
0.853125
The best k value is:
{'n_neighbors': 7}
Time taken is 41.013 seconds
knn's best Accuracy result for train data is: 0.849
knn's Accuracy result for test data is: 0.842
```

Gaussian Naive bayes

In [40]:
```
from sklearn.naive_bayes import GaussianNB
#Create NB model
nb = GaussianNB()
#use raw data

start = time.time()

start = time.time()

nb.fit(pca_data_train, split_label_train)

nb_res_pca = nb.predict(pca_data_test)

nb_res_pca_og = nb.predict(pca_data_test_og)

end = time.time()

print(f"Time taken is {format(end-start, '.3f')} seconds")

acc_nb_pca = cal_accuacy(nb_res_pca,split_label_test)

print(f"naive bayes's accuracy result for train data is: {format(acc_nb_pca, '.3f')}")

acc_nb_pca_og = cal_accuacy(nb_res_pca_og,labeltest)

print(f"naive bayes's accuracy result for test data is: {format(acc_nb_pca_og, '.3f')}")
```

```
Time taken is 0.068 seconds
naive bayes's accuracy result for train data is: 0.770
naive bayes's accuracy result for test data is: 0.757
```

In [41]:
```
#use grid search with 10-fold stratified cross validation to find best performance
#The var_smoothing parameter's default value is 10^-9 and We will apply the grid search from 0 to 10^-9
params = {'var_smoothing': np.logspace(0,-9, num=100)}

start = time.time()

#the GridSearchCV will give the best value
nb_grid = GridSearchCV(nb, param_grid=params, verbose=1, cv=10, n_jobs=-1)
nb_grid.fit(pca_data_train, split_label_train)
print("The best train/train prediction score is:" )
print(nb_grid.best_score_)
print("The best var_smoothing value is:" )
print(nb_grid.best_estimator_)
#use value find via best_estimator to do new prediction
nb_res_pca_GridS = nb_grid.predict(pca_data_test)

nb_res_pca_GridS_og = nb_grid.predict(pca_data_test_og)

end = time.time()
print(f"Time taken is {format(end-start, '.3f')} seconds")

acc_nb_pca_GridS = cal_accuacy(nb_res_pca_GridS,split_label_test)

print(f"naive bayes's best accuracy result for train data after hyperparameter tunning is: {format(acc_nb_pca_Gri

acc_nb_pca_GridS_og = cal_accuacy(nb_res_pca_GridS_og,labeltest)

print(f"naive bayes's best accuracy result for test data after hyperparameter tunning is: {format(acc_nb_pca_Grid
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
The best train/train prediction score is:
0.7697083333333333
The best var_smoothing value is:
GaussianNB(var_smoothing=8.111308307896872e-05)
Time taken is 9.544 seconds
naive bayes's best accuracy result for train data after hyperparameter tunning is: 0.770
naive bayes's best accuracy result for test data after hyperparameter tunning is: 0.755
```

Support Vector Machines

In [42]:
```
from sklearn.svm import SVC
svm = SVC()

start = time.time()

#use PCA data
svm.fit(pca_data_train, split_label_train)

svm_res_nom = svm.predict(pca_data_test)

svm_res_nom_og = svm.predict(pca_data_test_og)

end = time.time()
print(f"Time taken is {format(end-start, '.3f')} seconds")

acc_svm_nom = cal_accuacy(svm_res_nom,split_label_test)

print(f"Support Vector Machines's accuracy result for normalised train data is: {format(acc_svm_nom, '.3f')}")

acc_svm_nom_og = cal_accuacy(svm_res_nom_og,labeltest)

print(f"Support Vector Machines's accuracy result for normalised test data is: {format(acc_svm_nom_og, '.3f')}")
```

```
Time taken is 13.704 seconds
Support Vector Machines's accuracy result for normalised train data is: 0.873
Support Vector Machines's accuracy result for normalised test data is: 0.865
```

In [43]:

```
#find hyperparameter and use grid search with 10-fold stratified cross validation to find best performance
param_linear =  {'C': [0.01, 0.1, 1, 10, 100]}
start = time.time()
grid_svm_linear= GridSearchCV(SVC(kernel="linear"), param_grid = param_linear, cv=10, n_jobs=-1)
grid_svm_linear.fit(pca_data_train, split_label_train)
print("The best train/train prediction score is:" )
print(grid_svm_linear.best_score_)
#find the best parameters value
print("The best parameters value is:" )
print(grid_svm_linear.best_estimator_)
svm_res_linear = grid_svm_linear.predict(pca_data_test)

svm_res_linear_og = grid_svm_linear.predict(pca_data_test_og)

end = time.time()
print(f"Time taken is {format(end-start, '.3f')} seconds")

acc_svm_linear = cal_accuacy(svm_res_linear,split_label_test)

print(f"Support Vector Machines's best accuracy result for train data after hyperparameter tunning is: {format(ac

acc_svm_linear_og = cal_accuacy(svm_res_linear_og,labeltest)

print(f"Support Vector Machines's best accuracy result for test data after hyperparameter tunning is: {format(acc
```

```
The best train/train prediction score is:
0.859125
The best parameters value is:
SVC(C=0.01, kernel='linear')
Time taken is 947.646 seconds
Support Vector Machines's best accuracy result for train data after hyperparameter tunning is: 0.848
Support Vector Machines's best accuracy result for test data after hyperparameter tunning is: 0.843
```

In [44]:

```
param_rbf =  {'C': [0.01, 0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1]}

start = time.time()
grid_svm_rbf= GridSearchCV(SVC(kernel="rbf"), param_grid = param_rbf, cv=10, n_jobs=-1)
grid_svm_rbf.fit(pca_data_train, split_label_train)
print("The best train/train prediction score is:" )
print(grid_svm_rbf.best_score_)
#find the best parameters value
print("The best parameters value is:" )
print(grid_svm_rbf.best_estimator_)
svm_res_rbf = grid_svm_rbf.predict(pca_data_test)
svm_res_rbf_og = grid_svm_rbf.predict(pca_data_test_og)

end = time.time()
print(f"Time taken is {format(end-start, '.3f')} seconds")
acc_svm_rbf = cal_accuacy(svm_res_rbf,split_label_test)
print(f"Support Vector Machines's best accuracy result for train data after hyperparameter tunning is: {format(ac
acc_svm_rbf_og = cal_accuacy(svm_res_rbf_og,labeltest)
print(f"Support Vector Machines's best accuracy result for test data after hyperparameter tunning is: {format(acc
```

```
The best train/train prediction score is:
0.8912083333333334
The best parameters value is:
SVC(C=10, gamma=0.01)
Time taken is 908.991 seconds
Support Vector Machines's best accuracy result for train data after hyperparameter tunning is: 0.893
Support Vector Machines's best accuracy result for test data after hyperparameter tunning is: 0.874
```

In [45]:
```
param_poly = {'C': [0.01, 0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1, 1]}

start = time.time()
grid_svm_poly= GridSearchCV(SVC(kernel="poly"), param_grid = param_poly, cv=10, n_jobs=-1)
grid_svm_poly.fit(pca_data_train, split_label_train)
print(grid_svm_poly.best_score_)
#find the best parameters value
print("The best parameters value is:" )
print(grid_svm_poly.best_estimator_)
svm_res_poly = grid_svm_poly.predict(pca_data_test)
svm_res_poly_og = grid_svm_poly.predict(pca_data_test_og)
end = time.time()
print(f"Time taken is {format(end-start, '.3f')} seconds")
acc_svm_poly = cal_accuacy(svm_res_poly,split_label_test)
print(f"Support Vector Machines's best accuracy result for train data after hyperparameter tunning is: {format(ac
acc_svm_poly_og = cal_accuacy(svm_res_poly_og,labeltest)
print(f"Support Vector Machines's best accuracy result for test data after hyperparameter tunning is: {format(acc
```

```
0.8885416666666666
The best parameters value is:
SVC(C=1, gamma=0.01, kernel='poly')
Time taken is 355.864 seconds
Support Vector Machines's best accuracy result for train data after hyperparameter tunning is: 0.890
Support Vector Machines's best accuracy result for test data after hyperparameter tunning is: 0.869
```

Comparing between classfiers

In [46]:
```
#add the best paramater for each classifier

start_knn = time.time()

pca_knn = PCA(n_components= 95)
pca_data_train_knn = pca_knn.fit_transform(data_train_norm)
pca_data_test_knn_og = pca_knn.transform(data_test_norm_og)

knn_best = KNeighborsClassifier(n_neighbors=7)

knn_best.fit(pca_data_train_knn, split_label_train)

knn_res_best = knn_best.predict(pca_data_test_knn_og)

end_knn = time.time()

acc_knn_best = cal_accuacy(knn_res_best,labeltest)
```

In [47]:
```
start_nv = time.time()

pca_nb = PCA(n_components= 84)
pca_data_train_nb = pca_nb.fit_transform(data_train_norm)
pca_data_test_nb_og = pca_nb.transform(data_test_norm_og)

nb_best = GaussianNB(var_smoothing=0.0002848035868435802)

nb_best.fit(pca_data_train_nb, split_label_train)

nb_res_best = nb_best.predict(pca_data_test_nb_og)

end_nv = time.time()

acc_nb_best = cal_accuacy(nb_res_best,labeltest)
```

In [48]:
```
start_svm = time.time()

pca_svm = PCA(n_components= 187)
pca_data_train_svm = pca_svm.fit_transform(data_train_norm)
pca_data_test_svm_og = pca_svm.transform(data_test_norm_og)

svm_best = SVC(C=10, gamma=0.01,kernel="rbf")

#use PCA data

svm_best.fit(pca_data_train_svm, split_label_train)

svm_res_best = svm_best.predict(pca_data_test_svm_og)


end_svm = time.time()


acc_svm_best = cal_accuacy(svm_res_best,labeltest)
```

In [50]:
```
from sklearn.metrics import accuracy_score

print(f"k-nearest neighbor classifier's Time taken is {format(end_knn-start_knn, '.3f')} seconds")
print(f"k-nearest neighbor classifier's best accuracy result for test data after hyperparameter tunning is: {form
print("")
print(f"naive bayes's Time taken is {format(end_nv-start_nv, '.3f')} seconds")
print(f"naive bayes's best accuracy result for test data after hyperparameter tunning is: {format(acc_nb_best, '.
print("")
print(f"Support Vector Machines's Time taken is {format(end_svm-start_svm,'.3f')} seconds")
print(f"Support Vector Machines's best accuracy result for test data after hyperparameter tunning is: {format(acc
```

```
k-nearest neighbor classifier's Time taken is 2.523 seconds
k-nearest neighbor classifier's best accuracy result for test data after hyperparameter tunning is: 0.839

naive bayes's Time taken is 0.299 seconds
naive bayes's best accuracy result for test data after hyperparameter tunning is: 0.754

Support Vector Machines's Time taken is 28.272 seconds
Support Vector Machines's best accuracy result for test data after hyperparameter tunning is: 0.879
```

In [51]:
```
#optut the best presdicted data
output(svm_res_best)
```

Hardware and software specifications

1. CPU: AMD 5900X
2. Ram: 16GB
3. GPU: RTX 3060TI