COMP5318 - Machine Learning and Data Mining: Assignment 1

load data

In [92]:
```
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 [93]:
```
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:
    data_test = np.copy(H['datatest'])
#load teating table which contain 2000 samples
with h5py.File('./Input/test/labels_testing_2000.h5','r') as H:
    label_test = np.copy(H['labeltest'])

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

print(data_test.shape,label_test.shape)
```

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

Functions

In [94]:
```
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 [95]:
```
#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 [96]:
```
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,)
```

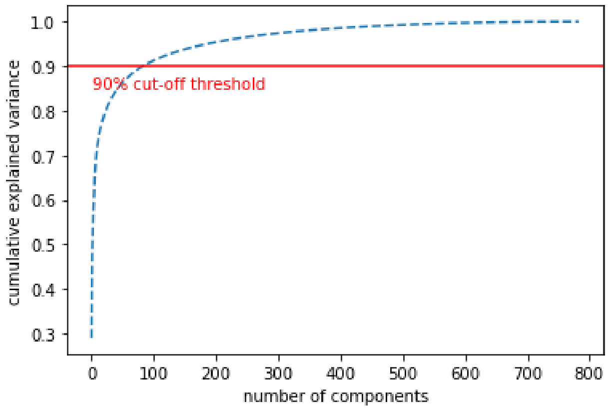
Data pre-processing

In [97]:
```
#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)
```

In [98]:
```
from sklearn.decomposition import PCA
#find best components number
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=0.90, color='r', linestyle='-')
plt.text(0.5, 0.85, '90% cut-off threshold', color = 'red', fontsize=10)
```

Out[98]:
```
Text(0.5, 0.85, '90% cut-off threshold')
```



In [99]:
```
# apply PCA to splited data
pca = PCA(n_components= 115)
pca_data_train = pca.fit(data_train_norm)
pca_data_train = pca.transform(data_train_norm)
pca_data_test = pca.transform(data_test_norm)
```

Classification algorithms

k-nearest neighbor classifier

In [100]:
```
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)

start = time.time()

knn.fit(split_data_train, split_label_train)

knn_res = knn.predict(split_data_test)

end = time.time()

acc_knn = cal_accuacy(knn_res,split_label_test)

#scores = cross_val_score(knn, split_data_train, split_label_train, cv=10, scoring='accuracy')

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

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

print(" ")

#apply knn with 1 neighbours and test the PCA data

start = time.time()

knn.fit(pca_data_train, split_label_train)

knn_res_pca = knn.predict(pca_data_test)

end = time.time()

acc_knn_pca = cal_accuacy(knn_res_pca,split_label_test)

#scores = cross_val_score(knn, pca_data_train, split_label_train, cv=10, scoring='accuracy')

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

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

#print(scores)
```

```
Time taken is 2.510 seconds
knn's Accuracy result for raw data is: 0.841

Time taken is 1.725 seconds
When k = 1, knn's Accuracy result for PCA data is: 0.842
```

In [101]:
```
#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, 19, 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]}
```

In [102]:
```
#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)

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 PCA data is: {format(acc_knn_pca, '.3f')}")
```

```
The best train/train prediction score is:
0.8535
The best k value is:
{'n_neighbors': 7}
Time taken is 34.829 seconds
knn's best Accuracy result for PCA data is: 0.849
```

Gaussian Naive bayes

In [103]:
```
from sklearn.naive_bayes import GaussianNB
#Create NB model
nb = GaussianNB()
#use raw data

start = time.time()

nb.fit(split_data_train, split_label_train)
nb_res = nb.predict(split_data_test)

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

acc_nb = cal_accuacy(nb_res,split_label_test)
print(f"naive bayes's accuracy result for raw data is: {format(acc_nb, '.3f')}")
print(" ")

start = time.time()

nb.fit(pca_data_train, split_label_train)
nb_res_pca = nb.predict(pca_data_test)

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 PCA data is: {format(acc_nb_pca, '.3f')}")
```

```
Time taken is 0.377 seconds
naive bayes's accuracy result for raw data is: 0.612

Time taken is 0.049 seconds
naive bayes's accuracy result for PCA data is: 0.767
```

In [104]:
```
#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)

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 PCA data after hyperparameter tunning is: {format(acc_nb_pca_GridS
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
The best train/train prediction score is:
0.7628333333333334
The best var_smoothing value is:
GaussianNB(var_smoothing=0.0002848035868435802)
Time taken is 14.187 seconds
naive bayes's best accuracy result for PCA data after hyperparameter tunning is: 0.768
```

Support Vector Machines

In [105]:
```
from sklearn.svm import SVC
svm = SVC()

start = time.time()

#use raw data
svm.fit(split_data_train, split_label_train)
svm_res = svm.predict(split_data_test)

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

acc_svm = cal_accuacy(svm_res,split_label_test)
print(f"Support Vector Machines's accuracy result for raw data is: {format(acc_svm, '.3f')}")
print(" ")

start = time.time()

#use PCA data
svm.fit(pca_data_train, split_label_train)
svm_res_nom = svm.predict(pca_data_test)

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 PCA data is: {format(acc_svm_nom, '.3f')}")
```

```
Time taken is 56.991 seconds
Support Vector Machines's accuracy result for raw data is: 0.872

Time taken is 11.055 seconds
Support Vector Machines's accuracy result for normalised PCA data is: 0.875
```

In [106]:

```
#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]}
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)
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 PCA data after hyperparameter tunning is: {format(acc_
```

```
The best train/train prediction score is:
0.861
The best parameters value is:
SVC(C=0.01, kernel='linear')
Time taken is 167.833 seconds
Support Vector Machines's best accuracy result for PCA data after hyperparameter tunning is: 0.853
```

In [107]:

```
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)
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 PCA data after hyperparameter tunning is: {format(acc_
```

```
The best train/train prediction score is:
0.8913749999999998
The best parameters value is:
SVC(C=10, gamma=0.01)
Time taken is 936.909 seconds
Support Vector Machines's best accuracy result for PCA data after hyperparameter tunning is: 0.896
```

In [108]:

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

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)
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 PCA data after hyperparameter tunning is: {format(acc_
```

```
0.8902083333333332
The best parameters value is:
SVC(C=1, gamma=0.01, kernel='poly')
Time taken is 584.183 seconds
Support Vector Machines's best accuracy result for PCA data after hyperparameter tunning is: 0.893
```

Comparing between classfiers

In [180]:
```
#add the best paramater for each classifier

pca_knn = PCA(n_components= 95)
pca_data_train_knn = pca_knn.fit(data_train_norm)
pca_data_train_knn = pca_knn.transform(data_train_norm)
pca_data_test_knn = pca_knn.transform(data_test_norm)

knn_best = KNeighborsClassifier(n_neighbors=7)

start_knn = time.time()

knn_best.fit(pca_data_train_knn, split_label_train)

knn_res_best = knn_best.predict(pca_data_test_knn)

end_knn = time.time()

acc_knn_best = cal_accuacy(knn_res_best,split_label_test)
```

In [181]:
```

pca_nb = PCA(n_components= 84)
pca_data_train_nb = pca_nb.fit(data_train_norm)
pca_data_train_nb = pca_nb.transform(data_train_norm)
pca_data_test_nb = pca_nb.transform(data_test_norm)

nb_best = GaussianNB(var_smoothing=0.0002848035868435802)

start_nv = time.time()


nb_best.fit(pca_data_train_nb, split_label_train)

nb_res_best = nb_best.predict(pca_data_test_nb)

end_nv = time.time()

acc_nb_best = cal_accuacy(nb_res_best,split_label_test)
```

In [182]:
```

pca_svm = PCA(n_components= 145)

pca_data_train_svm = pca_svm.fit(data_train_norm)

pca_data_train_svm = pca_svm.transform(data_train_norm)

pca_data_test_svm = pca_svm.transform(data_test_norm)

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

start_svm = time.time()

#use PCA data

svm_best.fit(pca_data_train_svm, split_label_train)

svm_res_best = svm_best.predict(pca_data_test_svm)


end_svm = time.time()


acc_svm_best = cal_accuacy(svm_res_best,split_label_test)
```

In [183]:
```
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"naive bayes's Time taken is {format(end_nv-start_nv, '.3f')} seconds")


print(f"Support Vector Machines's Time taken is {format(end_svm-start_svm,'.3f')} seconds")


print(f"k-nearest neighbor classifier's best accuracy result for PCA data after hyperparameter tunning is: {forma

print(f"naive bayes's best accuracy result for PCA data after hyperparameter tunning is: {format(acc_nb_best, '.3

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

```
k-nearest neighbor classifier's Time taken is 2.566 seconds
naive bayes's Time taken is 0.036 seconds
Support Vector Machines's Time taken is 17.364 seconds
k-nearest neighbor classifier's best accuracy result for PCA data after hyperparameter tunning is: 0.850
naive bayes's best accuracy result for PCA data after hyperparameter tunning is: 0.774
Support Vector Machines's best accuracy result for PCA data after hyperparameter tunning is: 0.898
```

In [184]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(f"k-nearest neighbor classifier")
print(" ")
print(confusion_matrix(split_label_test,knn_res_best))
print(" ")
print(classification_report(split_label_test, knn_res_best))
print(" ")
print(f"naive bayes")
print(confusion_matrix(split_label_test,nb_res_best))
print(" ")
print(classification_report(split_label_test, nb_res_best))
print(" ")
print(f"Support Vector Machines")
print(" ")
print(confusion_matrix(split_label_test,svm_res_best))
print(" ")
print(classification_report(split_label_test, svm_res_best))
```

```
k-nearest neighbor classifier

[[534   1  13  12   2   0  40   0   7   0]
 [  2 594   3  10   1   0   1   0   0   0]
 [  8   0 431   4  72   0  67   0   2   0]
 [ 30   6   7 484  24   0  11   0   3   0]
 [  3   0  61  16 422   0  52   0   2   0]
 [  0   0   0   2   0 517   1  65   2  24]
 [119   1  71  11  57   0 376   0   2   0]
 [  0   0   0   0   0   3   0 601   1  28]
 [  3   0   6   4   3   0   1   2 556   2]
 [  0   0   0   0   0   2   0  29   0 586]]

              precision    recall  f1-score   support

           0       0.76      0.88      0.82       609
           1       0.99      0.97      0.98       611
           2       0.73      0.74      0.73       584
           3       0.89      0.86      0.87       565
           4       0.73      0.76      0.74       556
           5       0.99      0.85      0.91       611
           6       0.68      0.59      0.63       637
           7       0.86      0.95      0.90       633
           8       0.97      0.96      0.97       577
           9       0.92      0.95      0.93       617

    accuracy                           0.85      6000
   macro avg       0.85      0.85      0.85      6000
weighted avg       0.85      0.85      0.85      6000


naive bayes
[[465   0  13  44   2   5  40   0  40   0]
 [  0 557  15  24   0   0   6   0   9   0]
 [  6   0 356   0  78   1 114   0  29   0]
 [ 32   2   5 460  22   2  33   0   9   0]
 [  3   0  45  27 374   1  93   0  13   0]
 [  3   0   0   0   0 450  17 119  17   5]
 [110   0  61  21  54   4 351   0  36   0]
 [  0   0   0   0   0  26   2 564   1  40]
 [  6   0   4   3   6  10  26  11 510   1]
 [  1   0   0   0   0  11   3  40   7 555]]

              precision    recall  f1-score   support

           0       0.74      0.76      0.75       609
           1       1.00      0.91      0.95       611
           2       0.71      0.61      0.66       584
           3       0.79      0.81      0.80       565
           4       0.70      0.67      0.68       556
           5       0.88      0.74      0.80       611
           6       0.51      0.55      0.53       637
           7       0.77      0.89      0.83       633
           8       0.76      0.88      0.82       577
           9       0.92      0.90      0.91       617

    accuracy                           0.77      6000
```

```
   macro avg       0.78      0.77      0.77      6000
weighted avg       0.78      0.77      0.77      6000


Support Vector Machines

[[527   0  14  13   1   0  48   0   6   0]
 [  1 597   1   9   2   0   1   0   0   0]
 [  4   0 486   2  49   0  40   0   3   0]
 [ 17   4   2 511  17   0  13   0   1   0]
 [  1   0  41  17 456   0  39   0   2   0]
 [  0   0   0   0   0 584   0  17   5   5]
 [ 80   1  46  17  36   0 455   0   2   0]
 [  0   0   0   0   0   6   0 612   2  13]
 [  3   0   1   1   1   1   4   0 566   0]
 [  0   0   0   0   0   7   0  18   0 592]]

              precision    recall  f1-score   support

           0       0.83      0.87      0.85       609
           1       0.99      0.98      0.98       611
           2       0.82      0.83      0.83       584
           3       0.90      0.90      0.90       565
           4       0.81      0.82      0.82       556
           5       0.98      0.96      0.97       611
           6       0.76      0.71      0.74       637
           7       0.95      0.97      0.96       633
           8       0.96      0.98      0.97       577
           9       0.97      0.96      0.96       617

    accuracy                           0.90      6000
   macro avg       0.90      0.90      0.90      6000
weighted avg       0.90      0.90      0.90      6000
```

In [185]:
```
#optut the best presdicted data
output(svm_res_best)
```

Hardware and software specifications

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