ENEE436 Proj1

November 4, 2020

1 Import libraries

[33]:
```
import numpy as np
import pandas as pd
from tqdm import tqdm
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import ListedColormap
from mlxtend.data import loadlocal_mnist

import faiss
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.decomposition import PCA
```

[34]:
```
class FaissKNeighbors:
    def __init__(self, k=5):
        self.index = None
        self.y = None
        self.k = k

    def fit(self, X, y):
        self.index = faiss.IndexFlatL2(X.shape[1])
        self.index.add(X.astype(np.float32))
        self.y = y

    def predict(self, X):
        distances, indices = self.index.search(X.astype(np.float32), k=self.k)
        votes = self.y[indices]
        predictions = np.array([np.argmax(np.bincount(x)) for x in votes])
        return predictions
```

2 Load dataset

- Downloaded the training and test dataset from http://yann.lecun.com/exdb/mnist/
- Unzipped them to folder called ‘data’ under current directory

[35]:
```
X_train, y_train = loadlocal_mnist(images_path='data/train-images.idx3-ubyte',␣
↪labels_path='data/train-labels.idx1-ubyte')
X_test, y_test  = loadlocal_mnist(images_path='data/t10k-images.idx3-ubyte',␣
↪labels_path='data/t10k-labels.idx1-ubyte')
```

[36]:
```
training_size = X_train.shape[0]
test_size = X_test.shape[0]
```

3 Training

3.1 Naive Bayesian classification with Gaussian assumption

[37]:
```
nbc_model = GaussianNB().fit(X_train, y_train)

nbc_train_y_pred = nbc_model.predict(X_train)
nbc_test_y_pred = nbc_model.predict(X_test)
```

[38]:
```
nbc_training_error = (y_train != nbc_train_y_pred).sum()
nbc_test_error = (y_test != nbc_test_y_pred).sum()
nbc_training_error_rate = nbc_training_error/training_size
nbc_test_error_rate = nbc_test_error/test_size
```

[39]:
```
print('training error:\t {} / {} ==> {}'.
↪format(nbc_training_error,training_size,nbc_training_error_rate))
print('test error:\t  {} / {} ==> {}'.format(nbc_test_error, test_size,␣
↪nbc_test_error_rate))
```

```
training error:  26106 / 60000 ==> 0.4351
test error:       4442 / 10000 ==> 0.4442
```

3.2 Nearest neighbors classification

3.2.1 N neighbor = 1, 5, 10, 20, 50, 100

[85]:
```
n = [1,5,10,20,50,100]
knn_training_error_rate = []
knn_test_error_rate = []
```

[86]:
```
for i in n:
    tmp_model = FaissKNeighbors(i)
    tmp_model.fit(X_train, y_train)
    knn_train_y_pred = tmp_model.predict(X_train)
    knn_test_y_pred = tmp_model.predict(X_test)
    knn_training_error_rate.append(((y_train != knn_train_y_pred).sum())/
↪training_size)
    knn_test_error_rate.append(((y_test != knn_test_y_pred).sum())/test_size)
```

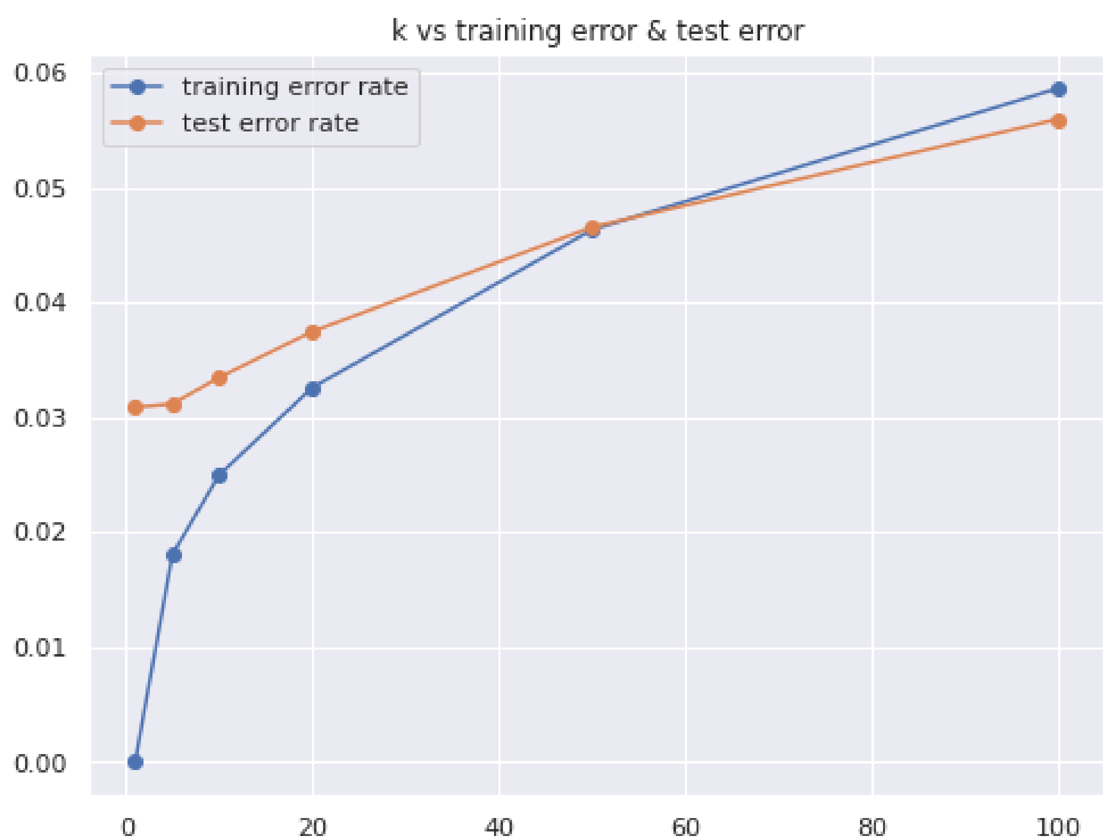
[87]:
```
d1 = {'k': n,
      'knn_train_error': knn_training_error_rate,
      'knn_test_error': knn_test_error_rate}
df1 = pd.DataFrame(data=d1)
df1
```

[87]:

	k	knn_train_error	knn_test_error
0	1	0.000000	0.0309
1	5	0.018083	0.0312
2	10	0.025000	0.0335
3	20	0.032617	0.0375
4	50	0.046367	0.0466
5	100	0.058683	0.0560

[88]:
```
plt.figure(figsize=(8,6))
plt.plot(n, knn_training_error_rate, '-o', label = "training error rate")
plt.plot(n, knn_test_error_rate, '-o', label = "test error rate")
plt.title("k vs training error & test error")
plt.legend()
```

[88]: <matplotlib.legend.Legend at 0x7fb758384d50>



3.3 Fisher's LDA classification

[44]:
```python
def twoclas_lda(class1, class2):
    filter_train = np.where((y_train == class1 ) | (y_train == class2))
    filter_test = np.where((y_test == class1) | (y_test == class2))

    X_train_filter, y_train_filter = X_train[filter_train],␣
 ↪y_train[filter_train]
    X_test_filter, y_test_filter = X_test[filter_test], y_test[filter_test]

    lda_model = LinearDiscriminantAnalysis().fit(X_train_filter, y_train_filter)
    lda_train_y_pred = lda_model.predict(X_train_filter)
    lda_test_y_pred = lda_model.predict(X_test_filter)

    lda_training_error = (y_train_filter != lda_train_y_pred).sum()
    lda_test_error = (y_test_filter != lda_test_y_pred).sum()
    lda_training_error_rate = lda_training_error/X_train_filter.shape[0]
    lda_test_error_rate = lda_test_error/X_test_filter.shape[0]

    return (lda_training_error_rate, lda_test_error_rate)
```

[45]:
```python
train_e_09, test_e_09 = twoclas_lda(0,9)
train_e_08, test_e_08 = twoclas_lda(0,8)
train_e_17, test_e_17 = twoclas_lda(1,7)
```

[46]:
```python
d_lda = {'LDA': ['0 vs 9', '0 vs 8', '1 vs 7'],
         'trainning error': [train_e_09, train_e_08, train_e_17],
         'test error': [test_e_09, test_e_08, test_e_17]}
df_lda = pd.DataFrame(data = d_lda)
df_lda
```

[46]:

	LDA	trainning error	test error
0	0 vs 9	0.004970	0.011564
1	0 vs 8	0.011296	0.010235
2	1 vs 7	0.006996	0.010633

3.4 PCA

3.4.1 n_components = 2

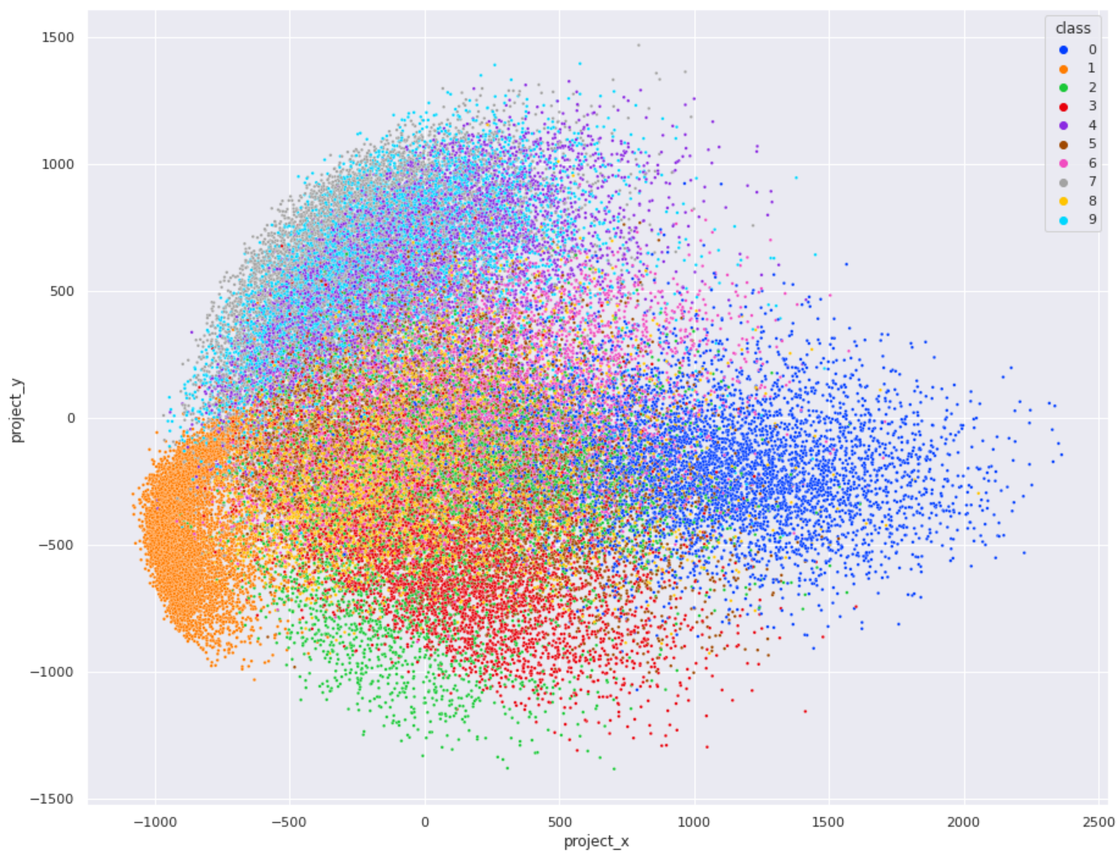
[47]:
```python
pca2 = PCA(n_components=2)
pca2.fit(X_train)
pca2_X_train = pca2.transform(X_train)

pca2_x_x = []
pca2_x_y = []
```

```
for i in pca2_X_train:
    pca2_x_x.append(i[0])
    pca2_x_y.append(i[1])
```

[48]:
```
pca2_d = {'project_x': pca2_x_x, 'project_y': pca2_x_y, 'class': y_train}
pca2_df = pd.DataFrame(data=pca2_d)
```

[76]:
```
sns.set(rc={'figure.figsize':(15,12)})
sns.scatterplot(data=pca2_df, x="project_x", y="project_y", hue="class", s=7,␣
 ↪palette = sns.color_palette('bright'))
```

[76]: <AxesSubplot:xlabel='project_x', ylabel='project_y'>



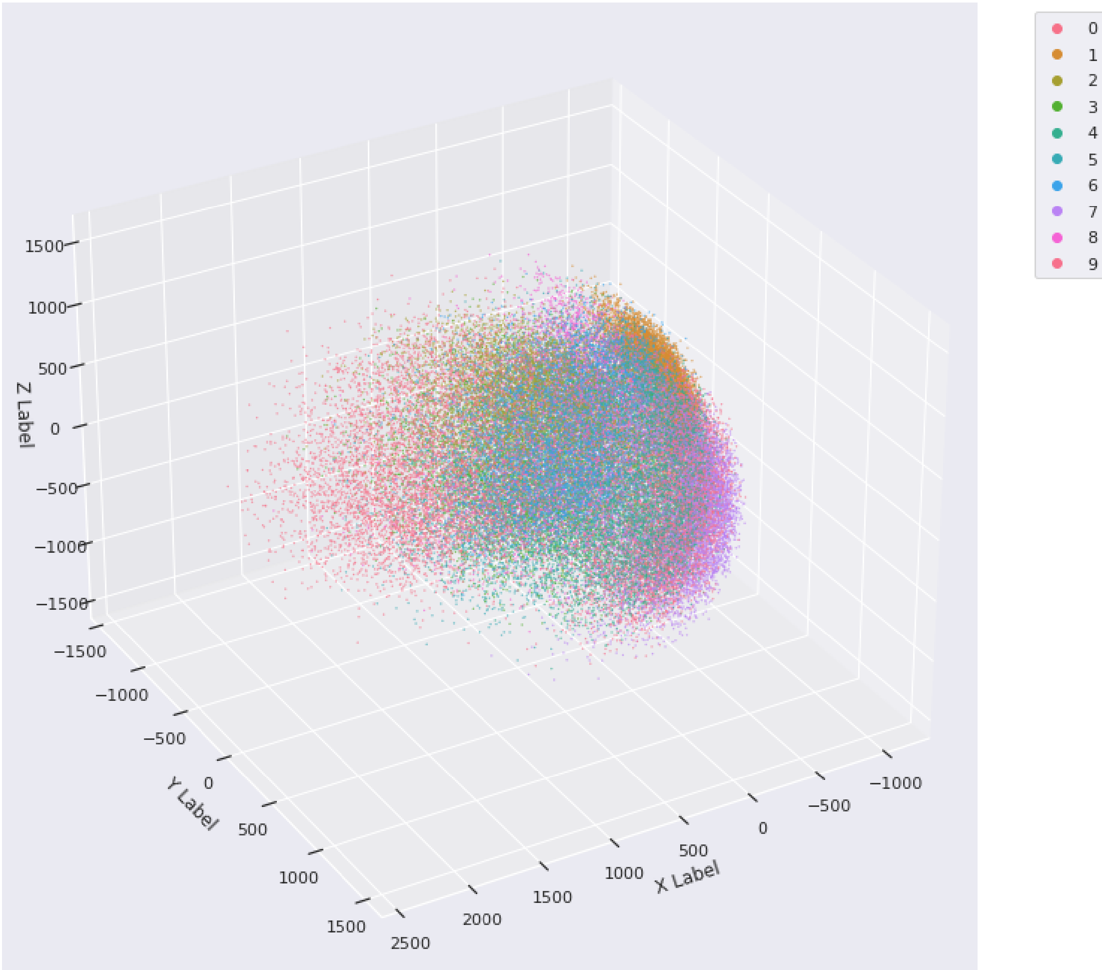
3.4.2 n_components = 3

[50]:
```
pca3 = PCA(n_components=3)
pca3.fit(X_train)
pca3_X_train = pca3.transform(X_train)
```

```
pca3_x_x = []
pca3_x_y = []
pca3_x_z = []

for i in pca3_X_train:
    pca3_x_x.append(i[0])
    pca3_x_y.append(i[1])
    pca3_x_z.append(i[2])
```

[51]:
```
pca3_d = {'project_x': pca3_x_x, 'project_y': pca3_x_y, 'project_z': pca3_x_z,␣
 ↪'class': y_train}
pca3_df = pd.DataFrame(data=pca3_d)
```

[81]:
```
fig = plt.figure(figsize=(15,12))
ax = fig.add_subplot(111, projection = '3d')
cmap = ListedColormap(sns.color_palette("husl", 256).as_hex())

# pca3_df_tmp = pca3_df
pca3_df_tmp = pca3_df

x = pca3_df_tmp['project_x']
y = pca3_df_tmp['project_y']
z = pca3_df_tmp['project_z']
hu = pca3_df_tmp['class']

sc = ax.scatter(x, y, z, s=0.1, c=hu, marker='o', cmap=cmap, alpha=1)
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

plt.legend(*sc.legend_elements(), bbox_to_anchor=(1.05, 1), loc=2)
ax.view_init(30, 60)
```



3.4.3 Naive Bayesian classification & KNN with PCA n_components = 5, 10, 20, 50, 100

[53]:
```
n = [5,10,20,50,100]
pca_nbc_training_error_rate = []
pca_nbc_test_error_rate = []
pca_knn_training_error_rate = []
pca_knn_test_error_rate = []
```

[54]:
```
for i in tqdm(n):
    # Create PCA
    curr_pca = PCA(n_components=i)
    curr_pca.fit(X_train)

    # transform X_train and X_test with PCA
    pca_X_train = curr_pca.transform(X_train)
```

```
    pca_X_test = curr_pca.transform(X_test)

    # train naive bayesian classification model and KNN model
    pca_nbc_model = GaussianNB().fit(pca_X_train, y_train)
    pca_knn_model = FaissKNeighbors(5)
    pca_knn_model.fit(pca_X_train, y_train)

    # Make prediction with naive bayesian classification
    pca_nbc_train_y_pred = pca_nbc_model.predict(pca_X_train)
    pca_nbc_test_y_pred = pca_nbc_model.predict(pca_X_test)
    pca_nbc_training_error = (y_train != pca_nbc_train_y_pred).sum()
    pca_nbc_test_error = (y_test != pca_nbc_test_y_pred).sum()
    pca_nbc_training_error_rate.append(pca_nbc_training_error/training_size)
    pca_nbc_test_error_rate.append(pca_nbc_test_error/test_size)

    # Make prediction with KNN
    pca_knn_train_y_pred = pca_knn_model.predict(pca_X_train)
    pca_knn_test_y_pred = pca_knn_model.predict(pca_X_test)
    pca_knn_training_error_rate.append(((y_train != pca_knn_train_y_pred).
↪sum())/training_size)
    pca_knn_test_error_rate.append(((y_test != pca_knn_test_y_pred).sum())/
↪test_size)
```

```
100%|     | 5/5 [00:32<00:00,  6.59s/it]
```

[55]:
```
d3 = {'PCA': n,
      'knn_train_error': pca_knn_training_error_rate,
      'knn_test_error': pca_knn_test_error_rate,
      'nbc_train_error': pca_nbc_training_error_rate,
      'nbc_test_error': pca_nbc_test_error_rate}
df3 = pd.DataFrame(data=d3)
df3
```

[55]:

	PCA	knn_train_error	knn_test_error	nbc_train_error	nbc_test_error
0	5	0.186367	0.2527	0.354583	0.3420
1	10	0.045517	0.0726	0.229500	0.2218
2	20	0.018867	0.0307	0.158967	0.1466
3	50	0.014133	0.0253	0.128583	0.1223
4	100	0.015950	0.0274	0.131233	0.1212

[82]:
```
plt.figure(figsize=(8,6))
plt.plot(n, pca_nbc_training_error_rate, '-o', label = "Naive Bayesian training␣
↪error rate")
plt.plot(n, pca_nbc_test_error_rate, '-o', label = "Naive Bayesian test error␣
↪rate")
plt.plot(n, pca_knn_training_error_rate, '-o', label = "KNN training error␣
↪rate")
```

```
plt.plot(n, pca_knn_test_error_rate, '-o', label = "KNN test error rate")
plt.title("Naive Bayesian, KNN's training & test error under different PCA")
plt.legend()
```

[82]: <matplotlib.legend.Legend at 0x7fb75853cbd0>