Machine Learning Homework 5

Ian Sinclair
ENCE 3631-1

March 29, 2022
Dr. Zhihui Zhu

Problem 1:

The objective of this computer exercise is to use Principal Component Analysis (PCA) for data dimensionality reduction (reducing the size of feature vector) and do classification afterward.

A data set containing roughly $70,000$ images of handwritten 28×28 digits from $[0, 9]$ is considered. In particular we desire to build a classifier to separate digits 4 and 9. And so to partition the data set we take only elements classified as 4 or 9 then concatenate the two into a single refined data set,

```
X, y = fetch_openml('mnist_784',version=1,return_X_y=True)

j= 10
plt.title('The jth image is a {label}'.format(label=int(y[j])))
plt.imshow(X[j].reshape((28,28)),cmap='gray')
plt.show()

X4 = X[y=='4',:]
X9 = X[y=='9',:]
y4 = 4*np.ones((len(X4),), dtype=int)
y9 = 9*np.ones((len(X9),), dtype=int)

X = np.concatenate((X4,X9),axis=0)
y = np.concatenate((y4,y9),axis=0)

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.43,random_state=0)

#Preprocessing.
X_train = preprocessing.scale(X_train)
X_test = preprocessing.scale(X_test)
```

a) The first task is to use the training set to train PCA such that 80% of the energy of the eigenvalues are retained. You can use the following code to perform PCA and compute the corresponding eigenvalues:

TASK 1:

Consider the original data set must treat each pixel independently and thereby has a large dimension for each feature. $(28 \times 28 \to \mathbb{R}^{784})$. As a result, classifiers such as SVM's can become inefficient. Critically; however, most of the pixels in each feature are irrelevant, or that a low-rank subspace can likely capture most of the information, and so we consider a Principal Component Analysis (PCA) to reduce the dimensionality of each feature, while retaining as much information as possible. Such that, if $x_i \in \mathbb{R}^{784}$ is a particular feature in the original data set, we note the transformation,

$$x_i \to \theta_i \in \mathbb{R}^K$$

Where K is the new (reduced) diminsionality of the feature. $(784 \gg K)$.
Now to approach this transformation, consider the approximation,

$$x_i \approx \mu A\theta_i, \qquad \mu \in \mathbb{R}^{784}, \quad A \in \mathbb{R}^{784\times K}, \quad \theta \in \mathbb{R}^K.$$

Or for fixed A, μ,

$$\theta_i = A^T(x_i - \mu).$$

Now in determining the optimal A, μ consider scatter matrix, S, as a scaled version of the empirical covariance matrix,

$$S = \sum_{i=1}^{n} x_i x_i^T,$$

And so it follows that the optimal A matrix results from,

$$\max_A (A^T S A), \qquad A^T A = I,$$

Or in the most general case,

$$S = U\Lambda U^T,$$

Where U is an orthogonal matrix with columns, $u_i, ...u_n$; and,

$$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}, \qquad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n,$$

And so by the orderings of eigenvalues λ it follows that the optimal A, is $A = [\lambda_1, ..., \lambda_K]$ or that A is selected as the largest amount of energy from the eigenvalues from S, that satisfies the proposed new (reduced) diminsionality across all the features (K).
As an extension and more specifically we desire the minimum dimensions for across all the features that ensures 80% of the energy from the eigenvalues is conserved. And so,

$$\sum_{k=1}^{K} \lambda_k \geq 0.80$$

Where again K is the size of the new feature vector.
Which results in the following,

```
"""
TASK 1: Selecting the number of components for PCA with 80\% of Energy Retained.
"""
pca = PCA(n_components=784)
pca.fit(X_train)

Variance = pca.explained_variance_ratio_

SumEigenValue = 0
k = 0

for lamda in Variance:
    if SumEigenValue < 0.8:
        k += 1
        SumEigenValue += lamda

pca = PCA(n_components=k)
pca.fit(X_train)

Variance = pca.explained_variance_ratio_
```

TASK 2:

Task 1 selects the optimal diminsionality K for the data set to retain 80% of the information from each feature. And so now we consider the transformation of $x_i \rightarrow \theta_i$, or the reduction of the diminsionality of the data set from $784 \rightarrow K$. Such that,

```
"""
TASK 2: Feature Reduction
"""

Theta_train = pca.transform(X_train)
Theta_test = pca.transform(X_test)
```

The combination of both tasks reduced the data set X from 784 dimensions to 107.

Original Dimension	Reduced Dimension
$\mathbb{R}^{784}$	$\mathbb{R}^{107}$

b) Repeat $HW3\ (a)$ on the transformed dataset Theta_train and Theta_test: train an SVM using the kernels $k(u,v) = (u^T v + 1)^p$, $p = 1, 2$, that is, the inhomogeneous linear and quadratic kernel. To do this, you will want to set the parameters $kernel =' poly'$ and $degree = 1$ or $degree = 2$. You can refer to my codes for $HW3\ (a)$ if you want, but I am not good at coding. Actually, most of you did a good job at $HW3$. For each kernel, report the best value of C and the $test\ error$, and compare the results in $HW3\ (a)$. **Turn in your code.**

Note, similar to HW 3, we approach a method to train the classifiers, we use the 'holdout method' and so we partition the training data into a $'fit'$ data set that will be used to train the classifier under a particular set of parameters and a $'holdout'$ set that will be used to test the parameters without needing to use the actual testing set.
Margins are taken such that the holdout set is 30% of the fit data set.

```
"""
Problem 1: Part B, inhomogeneous linear and quadratic kernel
"""

"""
Holdout Method
"""
#Partitions training data to sets used to train the classifier and test data for parameters.
Theta_fit,Theta_holdout,y_fit,y_holdout = train_test_split(Theta_train,y_train,test_size=0.3,random_state=0)
```

Now each type of classifier can be trained separately using the partitioned data sets.

i) *Inhomogeneous linear kernel:*

For the inhomogeneous linear kernel method there is only one parameter, C, that needs to be tuned. And so using the holdout method, we test values for C based on the 1 dimensional table point network

$$C = 2^k, \qquad k \in [-10, 10], \qquad k \in \mathbb{Z}.$$

This builds a significant range for $C << 1$ and $C >> 1$. Finally, the value for C that results in the smallest testing error is used to train the final classifier.

Inhomogeneous Linear Kernel SVM Classifier:

```
"""
Inhomogeneous Linear Kernel
"""

min_error = 1;
C_ = -1

#Select C
for n in range(-10,11,1):                    #Training on range [0.00006, 16384 : *2].
    clf = svm.SVC(C=2**n,kernel='poly',degree = 1) #Builds linear classifier
    clf.fit(Theta_fit,y_fit)                       #Trains classifier using the partitioned training data
```

```
    Pe = 1-clf.score(Theta_holdout,y_holdout)          #Tests classifier using the holdout set.
    if Pe < min_error:                                 #Maintains the C_ value corresponding to the minimum error.
        min_error = Pe
        C_ = 2**n

print(C_)
#Retrain
clf = svm.SVC(C=C_,kernel='poly',degree = 1)
clf.fit(Theta_train,y_train)
#Error and support vectors
Pe = 1-clf.score(Theta_test,y_test)
num_SV = clf.support_vectors_
```

Which results in the following,

	Dimensionality	Optimal C	Test Error	Number of Support Vectors
Reduced Dimensionality	107	2	0.035599	955
Original Data set HW 3	784	1	0.029188	1087

ii) *Quadratic Kernel.*

A similar system is used to train the Quadratic Kernel classifier, such that parameter C is tested by,

$$C = 1.2^k, \qquad k \in [-20, 20], \qquad k \in \mathbb{Z}.$$

Which resulted in,

Quadratic Kernel SVM Classifier

```
"""
Inhomogeneous Quadratic Kernel
"""

min_error = 1;
C_ = -1

#Select C
for n in range(-20,21,1):
    clf = svm.SVC(C=1.2**n,kernel='poly',degree = 2)
    clf.fit(Theta_fit,y_fit)
    Pe = 1-clf.score(Theta_holdout,y_holdout)
    if Pe < min_error:
        min_error = Pe
        C_ = 1.2**n

print(C_)
#Retrain
clf = svm.SVC(C=C_,kernel='poly',degree = 2)
clf.fit(Theta_train,y_train)
#Error and support vectors
Pe = 1-clf.score(Theta_test,y_test)
num_SV = clf.support_vectors_
```

	Dimensionality	Optimal C	Test Error	Number of Support Vectors
Reduced Dimensionality	107	8.916	0.014679	960
Original Data set HW 3	784	4	0.010123	1069

Appendix

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 25 17:46:44 2021

@author: IanSi
"""

import numpy as np
import pandas as pd
from sklearn import svm
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.datasets import fetch_openml

X, y = fetch_openml('mnist_784',version=1,return_X_y=True)

j= 10
plt.title('The jth image is a {label}'.format(label=int(y[j])))
plt.imshow(X[j].reshape((28,28)),cmap='gray')
plt.show()

X4 = X[y=='4',:]
X9 = X[y=='9',:]
y4 = 4*np.ones((len(X4),), dtype=int)
y9 = 9*np.ones((len(X9),), dtype=int)

X = np.concatenate((X4,X9),axis=0)
y = np.concatenate((y4,y9),axis=0)

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.43,random_state=0)

#Preprocessing.
X_train = preprocessing.scale(X_train)
X_test = preprocessing.scale(X_test)


"""
TASK 1: Selecting the number of components for PCA with 80% of Energy Retained.
"""

pca = PCA(n_components=784)
pca.fit(X_train)

Variance = pca.explained_variance_ratio_

SumEigenValue = 0

k = 0
for lamda in Variance:
    if SumEigenValue < 0.8:
        k += 1
        SumEigenValue += lamda


pca = PCA(n_components=k)
pca.fit(X_train)

Variance = pca.explained_variance_ratio_

#Total Vaiance = 0.80235

"""
TASK 2: Feature Reduction
"""

Theta_train = pca.transform(X_train)
Theta_test = pca.transform(X_test)


"""
Problem 1: Part B, inhomogeneous linear and quadratic kernel
"""

"""
```

```
Holdout Method
"""
#Partitions training data to sets used to train the classifier and test data for parameters.
Theta_fit,Theta_holdout,y_fit,y_holdout = train_test_split(Theta_train,y_train,test_size=0.3,random_state=0)

"""
Inhomogeneous Linear Kernel
"""

min_error = 1;
C_ = -1

#Select C
for n in range(-10,11,1):                          #Training on range [0.00006, 16384 : *2].
    clf = svm.SVC(C=2**n,kernel='poly',degree = 1) #Builds linear classifier
    clf.fit(Theta_fit,y_fit)                       #Trains classifier using the partitioned training data
    Pe = 1-clf.score(Theta_holdout,y_holdout)      #Tests classifier using the holdout set.
    if Pe < min_error:                             #Maintains the C_ value corresponding to the minimum error.
        min_error = Pe
        C_ = 2**n

print(C_)
#Retrain
clf = svm.SVC(C=C_,kernel='poly',degree = 1)
clf.fit(Theta_train,y_train)
#Error and support vectors
Pe = 1-clf.score(Theta_test,y_test)
num_SV = clf.support_vectors_

#num_SV = 955, Pe = 0.03492491985827573, C_ = 2


"""
Inhomogeneous Quadratic Kernel
"""

min_error = 1;
C_ = -1

#Select C
for n in range(-20,21,1):
    clf = svm.SVC(C=1.2**n,kernel='poly',degree = 2)
    clf.fit(Theta_fit,y_fit)
    Pe = 1-clf.score(Theta_holdout,y_holdout)
    if Pe < min_error:
        min_error = Pe
        C_ = 1.2**n

print(C_)
#Retrain
clf = svm.SVC(C=C_,kernel='poly',degree = 2)
clf.fit(Theta_train,y_train)
#Error and support vectors
Pe = 1-clf.score(Theta_test,y_test)
num_SV = clf.support_vectors_

#num_SV = 960, Pe = 0.014679, C_ = 8.9160045
```