Machine Learning Homework 3

Ian Sinclair
ENCE 3631-1

March 29, 2022
Dr. Zhihui Zhu

Problem 1:

In this problem you will use SVMs to build a simple text classification system.

a) 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*$= 2$.

For each kernel, report the best value of C, the test error, and the number of data points that are support vectors (returned via *clf.support_vectors_*. Turn in your code.

A data set containing roughly $70,000$ images of handwritten 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 = preprocessing.scale(X)

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)
```

This results in a data set containing roughly $14,000$ entries that are classified as either 4 or 9.
Now the data is partitioned again into different training and testing sets.

```
X“train,X“test,y“train,y“test = train“test“split(X,y,test“size=0.43,random“state=0)
```

Where the test data has approximately $6,000$ data points, and the training data has approximately $8,000$ entries.
Now approaching 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.

```
X“fit,X“holdout,y“fit,y“holdout = train“test“split(X“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 Classifier:

```
#Partitions training data to sets used to train the classifier and test data for parameters.
X“fit,X“holdout,y“fit,y“holdout = train“test“split(X“train,y“train,test“size=0.3,random“state=0)


#Inhomogeneous linear SVM classifier
min“error = 1;
C“ = 1

#Select C
for n in range(−14,15,1):                                  #Training on range [0.00006, 16384 : *2].
    clf = svm.SVC(C=2**n,kernel= poly ,degree = 1)         #Builds linear classifier
    clf.fit(X“fit,y“fit)                          #Trains classifier using the partitioned training data
    Pe = 1−clf.score(X“holdout,y“holdout)              #Tests classifier using the holdout set.
    if Pe "less than" 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(X“train,y“train)
#Error and support vectors
Pe = 1−clf.score(X“test,y“test)
num“SV = clf.support“vectors“
```

Best value of C	Test Error	Number of Support Vectors
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,

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

Which resulted in,

Quadratic Kernel SVM Classifier

```
#Quadratic Kernel SVM Classifier
min“error = 1;
C“ = 1

#Select C
for n in range(−14,15,1):
    clf = svm.SVC(C=2**n,kernel= poly ,degree = 2)
    clf.fit(X“fit,y“fit)
    Pe = 1−clf.score(X“holdout,y“holdout)
    if Pe   min“error:
        min“error = Pe
        C“ = 2**n

print(C“)
#Retrain
clf = svm.SVC(C=C“,kernel= poly ,degree = 2)
clf.fit(X“train,y“train)
#Error and support vectors
Pe = 1−clf.score(X“test,y“test)
num“SV = clf.support“vectors“
```

Best value of C	Test Error	Number of Support Vectors
4	0.0101231652	1069

b) Repeat the above using the radial basis function (RBF) kernel $k(u, v) = e^{\gamma}||u - v||_2^2$ (by setting the parameters kernel=$'$ rbf', $gamma = gamma$. You will now need to determine the best value for both C and γ. Report the best value of C and γ, the test error, and the number of support vectors.

This classifier is also trained using the holdout method; however, it required that two parameters be trained, C, γ. And so to find the optimal value we test build table points such that, $C << 1$, $C >> 1$, $\gamma << 1$ and $\gamma >> 1$. In particular following the below table point structure,

γ/C	2^{-10}	2^{-9}	2^{-8}	$\cdots$	2^{10}
2^{-10}					
2^{-9}					
2^{-8}					
$\vdots$					
2^{10}					

Radial Basis Function:

```
#Radial Basis Function
min“error = 1;
C“ = 1
gamma“ = 1

#Select C“ and gamma“
for k in range(-10,11,1):
    for n in range(-10,11,1):
        clf = svm.SVC(C=2**n,kernel= rbf ,gamma = 2**k)
        clf.fit(X“fit,y“fit)
        Pe = 1-clf.score(X“holdout,y“holdout)
        if Pe   min“error:
            min“error = Pe
            C“ = 2**n
            gamma“ = 2**k

print(C“)
print(gamma“)
#Retrain
clf = svm.SVC(C=C“,kernel= rbf ,gamma = gamma“)
clf.fit(X“train,y“train)
#Error and support vectors
Pe = 1-clf.score(X“test,y“test)
num“SV = clf.support“vectors“
```

Best value of γ	Best value of C	Test Error	Number of Support Vectors
0.003906	4	0.018221697	2032

Problem 2:

Suppose that the VC dimension of our hypothesis set H is $d_{VC} = 3$ (e.g.,linear classifiers in $\mathbb{R}^2$) and that we have an algorithm for selecting some $h^* \in H$ based on a training sample of size n (i.e., we have n example input-output pairs to train on).

a) Using the generalization bound given in class, give an upper bound (which depends on $\hat{R}_n(h^*)$) on $R(h^*)$ that holds with probability at least 0.95 in the case where $n = 100$. Repeat for $n = 1,000$ and $n = 10,000$.

Consider a set of $m = |\mathcal{H}|$ hypothesises in $\mathcal{H}$ and an algorithms that selects an optimal hypothesis h^* built from n training samples.
Then by Hoeffding's inequality,

$$\mathbb{P}\big[|\hat{R}_n(h^*) - R(h^*)| > \epsilon\big] \leq \underbrace{2me^{-2\epsilon^2 n}}_{\delta}$$

Then by selecting $\delta = 2me^{-2\epsilon^2 n}$, with a probability of $1 - \delta$ it follows that,

$$R(h^*) \leq \hat{R}_n(h^*) + \sqrt{\frac{1}{2n} \log \frac{2m}{\delta}}$$

However this bound is redundant for $|\mathcal{H}| \to \infty$, and so building towards a model of the above expression that does not depend on m, consider,

$$B(n,k) := \textit{maximum number of dichotomies on } n \textit{ points, with break point } k$$

Then by definition $m_{\mathcal{H}}(n) \leq B(n,k)$ and by Sauer's lemma

$$m_{\mathcal{H}}(n) \leq B(n,k) \leq \sum_{i=0}^{k-1} \binom{n}{i}$$

Which leads to the **VC Generalization Bound**, proved by *Vapnik* and *Chervonenkis* in 1971,

With a probability of $1-\delta$,

$$R(h^*) \leq \hat{R}_n(h^*) + \sqrt{\frac{8}{n} \log \frac{4m_{\mathcal{H}}(2n)}{\delta}},$$

And now because,

$$m_{\mathcal{H}}(n) \leq \sum_{i=0}^{k-1} \binom{n}{i} \leq n^{k-1} + 1,$$

it follows that,

$$R(h^*) \leq \hat{R}_n(h^*) + \sqrt{\frac{8d_{vc}}{n} \log \frac{8n}{\delta}},$$

Finally taking $d_{vc} = 3$, consider we desire to ensure the above expression holds with 95% probability. And so,

$$0.95 = 1 - \delta \longrightarrow \delta = 0.05.$$

Now it follows that,

$$R(h^*) \leq \hat{R}_n(h^*) + \sqrt{\frac{8(3)}{n} \log \frac{8n}{0.05}} = \hat{R}_n(h^*) + \sqrt{\frac{24}{n} \log 160n}$$

Now taking $n = 100, 1000, 10000$, and substituting into the above bound,

n	100	1000	$10,000$
$R(h^*) \leq$	$\hat{R}_n(h^*) + 1.5242$	$\hat{R}_n(h^*) + 0.53627$	$\hat{R}_n(h^*) + 0.18516$

b) Again using the generalization bound given in class, how large does n need to be to obtain a generalization bound of the form

$$R(h^*) \leq \hat{R}_n(h^*) + 0.1,$$

that holds with probability at least 0.95? How does this compare to the "rule of thumb" given in class?

Consider by the VC generalization bound, with a probability of $1-\delta$

$$R(h^*) \leq \hat{R}_n(h^*) + \sqrt{\frac{8d_{vc}}{n}\log\frac{8n}{\delta}},$$

And so for $d_{vc} = 3$, and $\delta = 0.05$, it follows that,

$$R(h^*) \leq \hat{R}_n(h^*) + \sqrt{\frac{24}{n}\log 160n},$$

And so by inspection,

$$0.1 = \sqrt{\frac{24}{n}\log 160n} \longrightarrow n = 37455$$

Additionally, the "rule of thumb" given in class states that the number of samples should be $n = 10d_{vc}$ or ten times the VC dimension. And so this implies,

Rule of thumb: $n = 10(d_{VC}) = 10(3) = 30.$

Therefore,

	Generalized VC bound	Rule of thumb
n	37455	30

Appendix:

Python Code: Problem 1

```
# -*- coding: utf-8 -*-
"""
Created on Sat May  8 19:02:40 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.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()

#Compare result with any without preprocessing.
X = preprocessing.scale(X)

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)



Training SVM using the holdout method


#Partitions training data to sets used to train the classifier and test data for parameters.
X“fit,X“holdout,y“fit,y“holdout = train“test“split(X“train,y“train,test“size=0.3,random“state=0)


#Inhomogeneous linear SVM Classifier
min“error = 1;
C“ = 1

#Select C
for n in range(-14,15,1): #Training on range [0.00006, 16384 : *2].
    clf = svm.SVC(C=2**n,kernel= poly ,degree = 1) #Builds linear classifier
    clf.fit(X“fit,y“fit) #Trains classifier using the partitioned training data
    Pe = 1-clf.score(X“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(X“train,y“train)
#Error and support vectors
Pe = 1-clf.score(X“test,y“test)
num“SV = clf.support“vectors“



#Quadratic Kernel SVM Classifier
min“error = 1;
C“ = 1

#Select C
```

```
for n in range(-14,15,1):
    clf = svm.SVC(C=2**n,kernel= poly ,degree = 2)
    clf.fit(X“fit,y“fit)
    Pe = 1-clf.score(X“holdout,y“holdout)
    if Pe   min“error:
        min“error = Pe
        C“ = 2**n

print(C“)
#Retrain
clf = svm.SVC(C=C“,kernel= poly ,degree = 2)
clf.fit(X“train,y“train)
#Error and support vectors
Pe = 1-clf.score(X“test,y“test)
num“SV = clf.support“vectors“




#Radial Basis Function
min“error = 1;
C“ = 1
gamma“ = 1

#Select C“ and gamma“
for k in range(-10,11,1):
    for n in range(-10,11,1):
        clf = svm.SVC(C=2**n,kernel= rbf ,gamma = 2**k)
        clf.fit(X“fit,y“fit)
        Pe = 1-clf.score(X“holdout,y“holdout)
        if Pe   min“error:
            min“error = Pe
            C“ = 2**n
            gamma“ = 2**k

print(C“)
print(gamma“)
#Retrain
clf = svm.SVC(C=C“,kernel= rbf ,gamma = gamma“)
clf.fit(X“train,y“train)
#Error and support vectors
Pe = 1-clf.score(X“test,y“test)
num“SV = clf.support“vectors“
```