CS189 HW1 write-up

Di Zhen(3035272293)

June 2019

1

Let

$$L(\omega_i, \alpha, \lambda) = \mid w \mid^2 - \sum_{i=1}^{m} \lambda_i (y_i (x_i \cdot \omega + \alpha) - 1)$$

To solve this problem:

$$minL(\omega, \alpha, \lambda)$$

Calculate the derivative of L at ω and α:

$$\bigtriangledown_\omega L(\omega, \alpha, \lambda) = 2\omega - \sum_{i=1}^{m} \lambda_i y_i x_i = 0$$

$$\bigtriangledown_\alpha L(\omega, \alpha, \lambda) = - \sum_{i=1}^{m} \lambda_i y_i = 0$$

We get:

$$2w = \sum_{i=1}^{m} \lambda_i y_i x_i$$

$$\sum_{i=1}^{m} \lambda_i y_i = 0$$

Then substitute them to $L(\omega, \alpha, \lambda)$ and we get:

$$\omega(\lambda, \alpha) = \sum_{i=1}^{m} \lambda_i - \frac{1}{4} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j x_i \cdot x_j$$

Therefore, the dual problem can be written as:

$$max_{\lambda_i \geq 0} \sum_{i=1}^{m} \lambda_i - \frac{1}{4} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j x_i \cdot x_j$$

$$\sum_{i=1}^{m} \lambda_i y_i = 0$$

2

Given the decision rule:

$$r(x) = \begin{cases} +1, & if \quad \omega \cdot x + \alpha \geq 0 \\ -1, & otherwise \end{cases}$$

Substitute with optimal λ^* and α^* and equation we obtain from question 1:

$$2\omega = \sum_{i=1}^{m} \lambda_i y_i x_i$$

We get:

$$r(x) = \begin{cases} +1, & if \quad \alpha^* + \frac{1}{2} \sum_{i=1}^{m} \lambda_i^* y_i x_i \cdot x \geq 0 \\ -1, & otherwise \end{cases}$$

3

Support vectors are the training samples used to maximize margin of hyperplane. When moving those support vectors, the hyperplane is moved, but when moving other points, the hyperplane could not move.
In Hard-Margin Support Vector Machines, non-support vectors still have influence on decision rule, because when training data, support vectors are determined from all datasets, we cannot just build a model with only a minority of points selected.

4 Appendix

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Jun 29 09:48:51 2019

@author: dizhen
"""

"""
MNIST - CODE

1. data loading
2. data partitioning (sets 10,000 validation set) without sklearn
3. SVM, training samples are 100,200,500,1000,2000,5000,10000
4. hyperparameter tuning, using 10000 training samples to find best C
5. build model with 10000 samples and predict test data
"""

## data loading
import sys
if sys.version_info[0] < 3:
    raise Exception("Python 3 not detected.")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import svm
from scipy import io
for data_name in ["mnist", "spam", "cifar10"]:
    data = io.loadmat("data/%s_data.mat" % data_name)
    print("\nloaded %s data!" % data_name)
    fields = "test_data", "training_data", "training_labels"
    for field in fields:
        print(field, data[field].shape)

mnist = io.loadmat("data/%s_data.mat" % "mnist")
mTraining = np.array(mnist['training_data'])
mLabel = np.array(mnist['training_labels'])
mTest = np.array(mnist['test_data'])

## data partitioning (sets 10,000 validation set) without sklearn
train_label = np.concatenate((mTraining, mLabel), axis=1)
Mcolname = np.array([str(i) for i in np.arange(mTraining.shape[1] + 1)])
train_label_df = pd.DataFrame(data = train_label, columns = Mcolname)
num_of_rows = int(train_label_df.shape[0] * 1/6)
train_label_df.sample(frac = 1)

X_train,y_train = train_label_df.loc[num_of_rows+1:,
                                     [str(i) for i in np.arange(mTraining.shape[1])]],
                                     train_label_df.loc[num_of_rows+1:,'784']
X_val,y_val = train_label_df.loc[:num_of_rows,
                                 [str(i) for i in np.arange(mTraining.shape[1])]],
                                 train_label_df.loc[:num_of_rows,'784']

## SVM, training samples are 100,200,500,1000,2000,5000,10000
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

acclist=[]
def trainingAll(arr,C):
    """
    A function to train data iterately with a sample or
    different number of samples and a C value or different
    C values.
    """
    for i in arr:
        for j in C:
            svm_clf=Pipeline((
            ('scaler',StandardScaler()),
            ('linear_svc',LinearSVC(C=j,loss='hinge'))
            ))
            svm_clf.fit(X_train[:i],y_train[:i])
            pred_svm=svm_clf.predict(X_val)
            acclist.append(accuracy_score(y_val,pred_svm))
            print("svm classification result,C="+ str(j) +
                  " and training samples = "+ str(i) + ':')
            print("accuracy_score:",accuracy_score(y_val,pred_svm))

arr = np.array([100,200,500, 1000, 2000, 5000, 10000])
C = np.array([1])
trainingAll(arr,C)
"""For 100,200,500,1000,2000,5000,10000 samples, the accuracy is
0.60622,0.63976,0.73736,0.76498,0.78926,0.81452,0.85722
"""

# save
nplist = np.array(acclist)
np.savetxt("m-acc.csv", nplist, delimiter=",")

# plot
errorM = 1-nplist
fig = plt.figure()
plt.plot(arr, errorM, label="Error rate", color="black")
plt.title("Error Rate Curve With SVM")
plt.xlabel("Number Of Sample")
plt.ylabel("Error rate")
plt.show()
fig.savefig('error rate curve of M.png')

# hyperparameter tuning, using 10000 training samples to find best C
arrC = np.array([10000])
Cs = np.array([0.001, 0.01, 0.1, 1, 10])
acclist=[]
trainingAll(arrC,Cs)
"""
For C = 0.001,0.01,0.1,1,10, the accuracy is 0.83168,0.88092,0.8794,
0.85796,0.83754, so that the best C is 0.01
"""
## build model with 10000 samples and predict test data
train_label_df.sample(frac = 1)
X_train,y_train = train_label_df.loc[num_of_rows+1:,
                                     [str(i) for i in np.arange(mTraining.shape[1])]],
                                     train_label_df.loc[num_of_rows+1:,'784']
X_val,y_val = train_label_df.loc[:num_of_rows,
                                 [str(i) for i in np.arange(mTraining.shape[1])]],
                                 train_label_df.loc[:num_of_rows,'784']

svm_clf=Pipeline((
        ('scaler',StandardScaler()),
        ('linear_svc',LinearSVC(C=0.01,loss='hinge'))
        ))

svm_clf.fit(X_train[:10000],y_train[:10000])
y_pred = svm_clf.predict(X_val)
print('svm classification result:')
print("accuracy_score:",accuracy_score(y_val,y_pred))
predOfTest = svm_clf.predict(mTest)

## save to csv
def results_to_csv(y_test):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1  # Ensures that the index starts at 1.
    df.to_csv('submission_m.csv', index_label='Id')
results_to_csv(predOfTest)


### improve the model

# grid search
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
parameters = {'kernel':('linear', 'rbf'),
              'C':(0.1,0.01,0.001),
              'gamma': (1,2,3,'auto'),
              'decision_function_shape':('ovo','ovr'),
              'shrinking':(True,False)}
svm = SVC()
clf = GridSearchCV(svm, parameters)
clf.fit(X_train[:10000],y_train[:10000])

from sklearn.model_selection import cross_val_score
print("accuracy:"+
      str(np.average(cross_val_score(clf,
                                     X_train[:10000],
                                     y_train[:10000],
                                     scoring='accuracy'))))
print('Best C:',clf.best_estimator_.C)
print('Best Kernel:',clf.best_estimator_.kernel)
print('Best Gamma:',clf.best_estimator_.gamma)
clf.score(X_train[:10000], y_train[:10000])

# prediction
y_pred_2 = clf.predict(X_val)
print('svm classification result:')
print("accuracy_score:",accuracy_score(y_val,y_pred_2))
predOfTest_2 = clf.predict(mTest)

## save to csv
def results_to_csv(y_test):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1  # Ensures that the index starts at 1.
    df.to_csv('submission_m_2.csv', index_label='Id')
results_to_csv(predOfTest_2)
```

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Jun 29 09:54:58 2019

@author: dizhen
"""

"""
SPAM - CODE
1. data loading
2. data partitioning (set 20% validation set) without sklearn
3. SVM, training samples are 100,200,500,1000,2000,all
4. 5-fold cross validation and tune hyperparameter C
5. build model with all the data and predict test data

"""
## data loading
import sys
if sys.version_info[0] < 3:
    raise Exception("Python 3 not detected.")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from scipy import io
for data_name in ["mnist", "spam", "cifar10"]:
    data = io.loadmat("data/%s_data.mat" % data_name)
    print("\nloaded %s data!" % data_name)
    fields = "test_data", "training_data", "training_labels"
    for field in fields:
        print(field, data[field].shape)

spam = io.loadmat("data/%s_data.mat" % "spam")
sTraining = np.array(spam['training_data'])
sLabel = np.array(spam['training_labels'])
sTest = np.array(spam['test_data'])

## data partitioning (set 20% validation set) without sklearn
train_label = np.concatenate((sTraining, sLabel), axis=1)
Scolname = np.array([str(i) for i in np.arange(sTraining.shape[1] + 1)])
train_label_df = pd.DataFrame(data = train_label, columns = Scolname)
num_of_rows = int(train_label_df.shape[0] * 1/5)
train_label_df.sample(frac = 1)

X_train,y_train = train_label_df.loc[num_of_rows+1:,
                                     [str(i) for i in np.arange(sTraining.shape[1])]],
                                     train_label_df.loc[num_of_rows+1:,'32']
X_val,y_val = train_label_df.loc[:num_of_rows,
                                 [str(i) for i in np.arange(sTraining.shape[1])]],
                                 train_label_df.loc[:num_of_rows,'32']

## SVM training samples are 100,200,500,1000,2000,all
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

acclist=[]
def trainingAll(arr,C):
    """
    A function to train data iterately with a sample or
    different number of samples and a C value or different
    C values.
    """
    for i in arr:
        for j in C:
            svm_clf=Pipeline((
            ('scaler',StandardScaler()),
            ('linear_svc',LinearSVC(C=j,loss='hinge'))
            ))
            svm_clf.fit(X_train[:i],y_train[:i])
            pred_svm=svm_clf.predict(X_val)
            acclist.append(accuracy_score(y_val,pred_svm))
            print("svm classification result,C="+ str(j) +
                  " and training samples = "+ str(i) + ':')
            print("accuracy_score:",accuracy_score(y_val,pred_svm))

arr = np.array([100,200,500, 1000, 2000, 4137])
C = np.array([1])
trainingAll(arr,C)
"""
For 100,200,500,1000,2000,4137 samples and C = 1, the accuracies are
0.7671497584541063,0.7855072463768116,0.8009661835748793,0.7874396135265701,
0.7990338164251207,0.8048309178743961
"""

# save
nplist = np.array(acclist)
np.savetxt("s-acc.csv", nplist, delimiter=",")
errorM = 1-nplist
fig = plt.figure()
plt.plot(arr, errorM, label="Error rate", color="black")
plt.title("Error Rate Curve With SVM")
plt.xlabel("Number Of Sample")
plt.ylabel("Error rate")
plt.show()
fig.savefig('error rate curve of S.png')

# 5-fold cross validation and tuning hyperparameter
def my_k_fold(df, k_fold, C):
    """
    A function for k fold cross validation and computing the mean
    of accuracy of each C value.
    """
    acc = []
    accmean = []
    test_size = 1/k_fold
    num_of_rows = int(df.shape[0] * test_size)
    for j in C:
        for i in np.arange(k_fold):
            # split data
            df.sample(frac = 1)
            X_train,y_train = df.loc[num_of_rows+1:,
                                     [str(i) for i in np.arange(sTraining.shape[1])]],
                                     df.loc[num_of_rows+1:,'32']
            X_val,y_val = df.loc[:num_of_rows,
                                 [str(i) for i in np.arange(sTraining.shape[1])]],
                                 df.loc[:num_of_rows,'32']

            # build model
            svm_clf = Pipeline((
        ('scaler',StandardScaler()),
        ('linear_svc',LinearSVC(C=j,loss='hinge'))
        ))
            svm_clf.fit(X_train, y_train)

            # use model to predict test data
            y_pred = svm_clf.predict(X_val)

            # accuracy
            acc.append(accuracy_score(y_val, y_pred))

        # store mean of accuracy
```

```python
        accmean.append(sum(acc)/len(acc))
        print("svm cross validation result, when C="+ str(j))
        print("mean of accuracy score:",sum(acc)/len(acc))
    return accmean

Cs = np.array([0.001, 0.01, 0.1, 1, 10,100])
my_k_fold(train_label_df,k_fold = 5,C = Cs)
"""
with C = 0.001,0.01,0.1,1,10,100, means of accuracy are 0.7843478260869565,
 0.7935265700483092, 0.7952979066022545, 0.7985024154589373, 0.7988019323671499,
 0.7969726247987119. So the best C is 0.01
"""

# build model with all the data and predict test data
num_of_rows = int(train_label_df.shape[0] * 1/5)
train_label_df.sample(frac = 1)
X_train,y_train = train_label_df.loc[num_of_rows+1:,
                                     [str(i) for i in np.arange(sTraining.shape[1])]],
                                     train_label_df.loc[num_of_rows+1:,'32']
X_val,y_val = train_label_df.loc[:num_of_rows,
                                 [str(i) for i in np.arange(sTraining.shape[1])]],
                                 train_label_df.loc[:num_of_rows,'32']

svm_clf=Pipeline((
        ('scaler',StandardScaler()),
        ('linear_svc',LinearSVC(C=0.01,loss='hinge'))
        ))

svm_clf.fit(X_train,y_train)
y_pred = svm_clf.predict(X_val)
print('svm classification result:')
print("accuracy_score:",accuracy_score(y_val,y_pred))
predOfTest=svm_clf.predict(sTest)

# save to csv
def results_to_csv(y_test):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1  # Ensures that the index starts at 1.
    df.to_csv('submission_s.csv', index_label='Id')
results_to_csv(predOfTest)


### improve model
# grid search
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
parameters = {'kernel':('linear', 'rbf'),
              'C':(0.001,0.01,0.1,1,10,100,1000),
              'gamma': (1,2,3,'auto'),
              'decision_function_shape':('ovo','ovr'),
              'shrinking':(True,False)}
svm = SVC()
clf = GridSearchCV(svm, parameters)
clf.fit(X_train,y_train)

from sklearn.model_selection import cross_val_score
print("accuracy:"+str(np.average(cross_val_score(clf,
                                                 X_train,
                                                 y_train,
                                                 scoring='accuracy'))))
print('Best C:',clf.best_estimator_.C)
print('Best Kernel:',clf.best_estimator_.kernel)
print('Best Gamma:',clf.best_estimator_.gamma)
clf.score(X_train, y_train)

# prediction
```

```
y_pred_2 = clf.predict(X_val)
print('svm classification result:')
print("accuracy_score:",accuracy_score(y_val,y_pred_2))
predOfTest_2 = clf.predict(sTest)

## save to csv
def results_to_csv(y_test):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1  # Ensures that the index starts at 1.
    df.to_csv('submission_s_2.csv', index_label='Id')
results_to_csv(predOfTest_2)
```

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Jun 30 09:25:17 2019

@author: dizhen
"""
"""
CIFAR-10 - CODE
1. data loading
2. data partitioning (set 5000 validation set) without sklearn
3. SVM, training samples are 100,200,500,1000,2000,5000
4. build model by 5000 samples and predict test data
"""

# data loading
import sys
if sys.version_info[0] < 3:
    raise Exception("Python 3 not detected.")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from scipy import io
for data_name in ["mnist", "spam", "cifar10"]:
    data = io.loadmat("data/%s_data.mat" % data_name)
    print("\nloaded %s data!" % data_name)
    fields = "test_data", "training_data", "training_labels"
    for field in fields:
        print(field, data[field].shape)

cifar10 = io.loadmat("data/%s_data.mat" % "cifar10")
cTraining = np.array(cifar10['training_data'])
cLabel = np.array(cifar10['training_labels'])
cTest = np.array(cifar10['test_data'])

# data partitioning (set 5000 validation set) without sklearn
train_label = np.concatenate((cTraining, cLabel), axis=1)
Ccolname = np.array([str(i) for i in np.arange(cTraining.shape[1] + 1)])
train_label_df = pd.DataFrame(data = train_label, columns = Ccolname)
num_of_rows = int(train_label_df.shape[0] * 1/10)
train_label_df.sample(frac = 1)

X_train,y_train = train_label_df.loc[num_of_rows+1:,
                                     [str(i) for i in np.arange(cTraining.shape[1])]],
                                     train_label_df.loc[num_of_rows+1:,'3072']
X_val,y_val = train_label_df.loc[:num_of_rows,
                                 [str(i) for i in np.arange(cTraining.shape[1])]],
                                 train_label_df.loc[:num_of_rows,'3072']

# SVM,training samples are 100,200,500,1000,2000,5000
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

arr = np.array([100, 200, 500, 1000, 2000, 5000])
arrC = np.array([5000])
C = np.array([1])

acclist=[]
def trainingAll(arr,C):
    """
    A function to train data iterately with a sample or
    different number of samples and a C value or different
    C values.
    """
    for i in arr:
        for j in C:
            svm_clf=Pipeline((
            ('scaler',StandardScaler()),
            ('linear_svc',LinearSVC(C=j,loss='hinge'))
            ))
            svm_clf.fit(X_train[:i],y_train[:i])
            pred_svm=svm_clf.predict(X_val)
            acclist.append(accuracy_score(y_val,pred_svm))
            print("svm classification result,C="+ str(j) +
                  " and training samples = "+ str(i) + ':')
            print("accuracy_score:",accuracy_score(y_val,pred_svm))

trainingAll(arr,C)
"""For 100,200,500,1000,2000,5000 samples, C = 1, the accuracies are
0.1888,0.1816, 0.2012,0.2118,0.2314,0.2422
"""
# save
nplist = np.array(acclist)
np.savetxt("c-acc.csv", nplist, delimiter=",")

errorM = 1-nplist
# Plot
fig = plt.figure()
plt.plot(arr, nplist, label="Error rate", color="black")
plt.title("Error Rate Curve With SVM")
plt.xlabel("Number Of Sample")
plt.ylabel("Error rate")
plt.show()
fig.savefig('error rate curve of C.png')

# build model by 5000 samples and predict test data
num_of_rows = int(train_label_df.shape[0] * 1/5)
train_label_df.sample(frac = 1)
X_train,y_train = train_label_df.loc[num_of_rows+1:,
                                     [str(i) for i in np.arange(cTraining.shape[1])]],
                                     train_label_df.loc[num_of_rows+1:,'32']
X_val,y_val = train_label_df.loc[:num_of_rows,
                                 [str(i) for i in np.arange(cTraining.shape[1])]],
                                 train_label_df.loc[:num_of_rows,'32']

svm_clf=Pipeline((
        ('scaler',StandardScaler()),
        ('linear_svc',LinearSVC(C=1,loss='hinge'))
        ))
svm_clf.fit(X_train[:5000],y_train[:5000])
y_pred = svm_clf.predict(X_val)
print('svm classification result:')
print("accuracy_score:",accuracy_score(y_val,y_pred))
predOfTest = svm_clf.predict(cTest)

# save to csv
def results_to_csv(y_test):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1  # Ensures that the index starts at 1.
    df.to_csv('submission_c.csv', index_label='Id')
results_to_csv(predOfTest)




### improve model
# grid search
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
parameters = {'kernel':('linear', 'rbf'),
              'C':(0.001,0.01,0.1),
              'gamma': (1,2,3,'auto'),
              'decision_function_shape':('ovo','ovr'),
              'shrinking':(True,False)}
svm = SVC()
clf = GridSearchCV(svm, parameters)
clf.fit(X_train[:5000],y_train[:5000])

from sklearn.model_selection import cross_val_score
print("accuracy:"+str(np.average(cross_val_score(clf,
                                                 X_train[:5000],
                                                 y_train[:5000],
                                                 scoring='accuracy'))))
print('Best C:',clf.best_estimator_.C)
print('Best Kernel:',clf.best_estimator_.kernel)
print('Best Gamma:',clf.best_estimator_.gamma)
clf.score(X_train[:5000], y_train[:5000])

# prediction
y_pred_2 = clf.predict(X_val)
print('svm classification result:')
print("accuracy_score:",accuracy_score(y_val,y_pred_2))
predOfTest_2 = clf.predict(cTest)

# save to csv
def results_to_csv(y_test):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1  # Ensures that the index starts at 1.
    df.to_csv('submission_s_2.csv', index_label='Id')
results_to_csv(predOfTest_2)
```

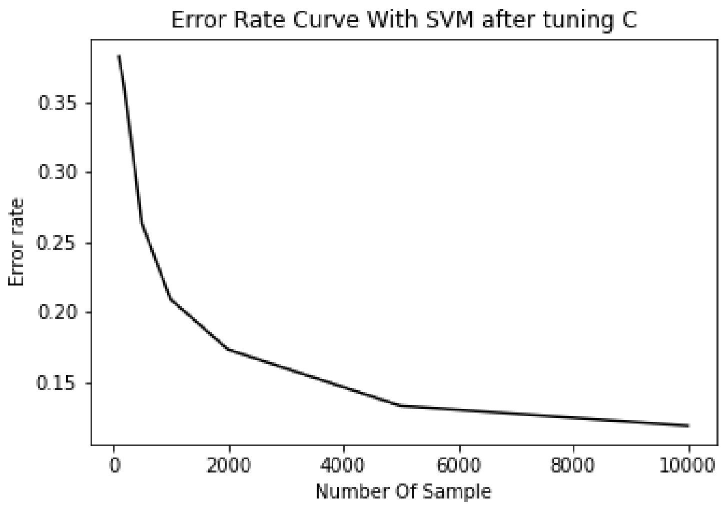


Figure 1: Error rate curve of training MNIST dataset by SVM after tuning hyperparameter C.

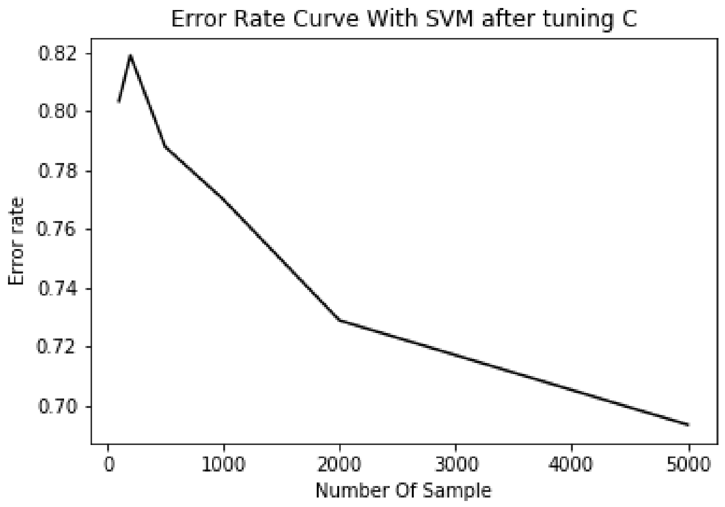


Figure 2: Error rate curve of training CIFAR-10 by SVM after tuning hyperparameter C.

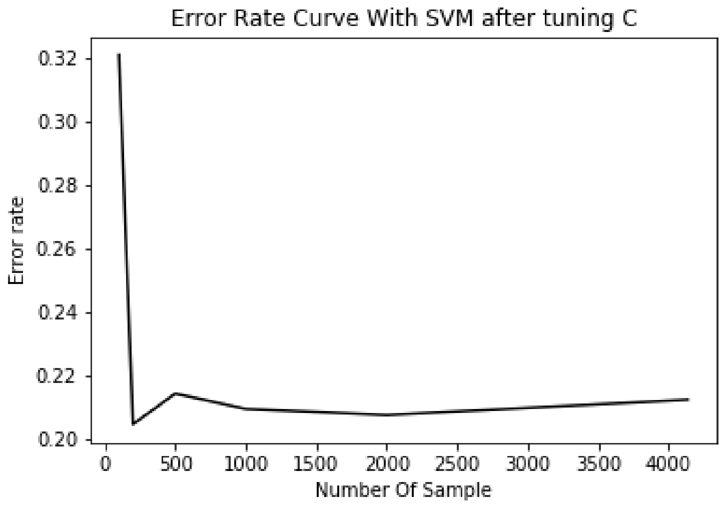
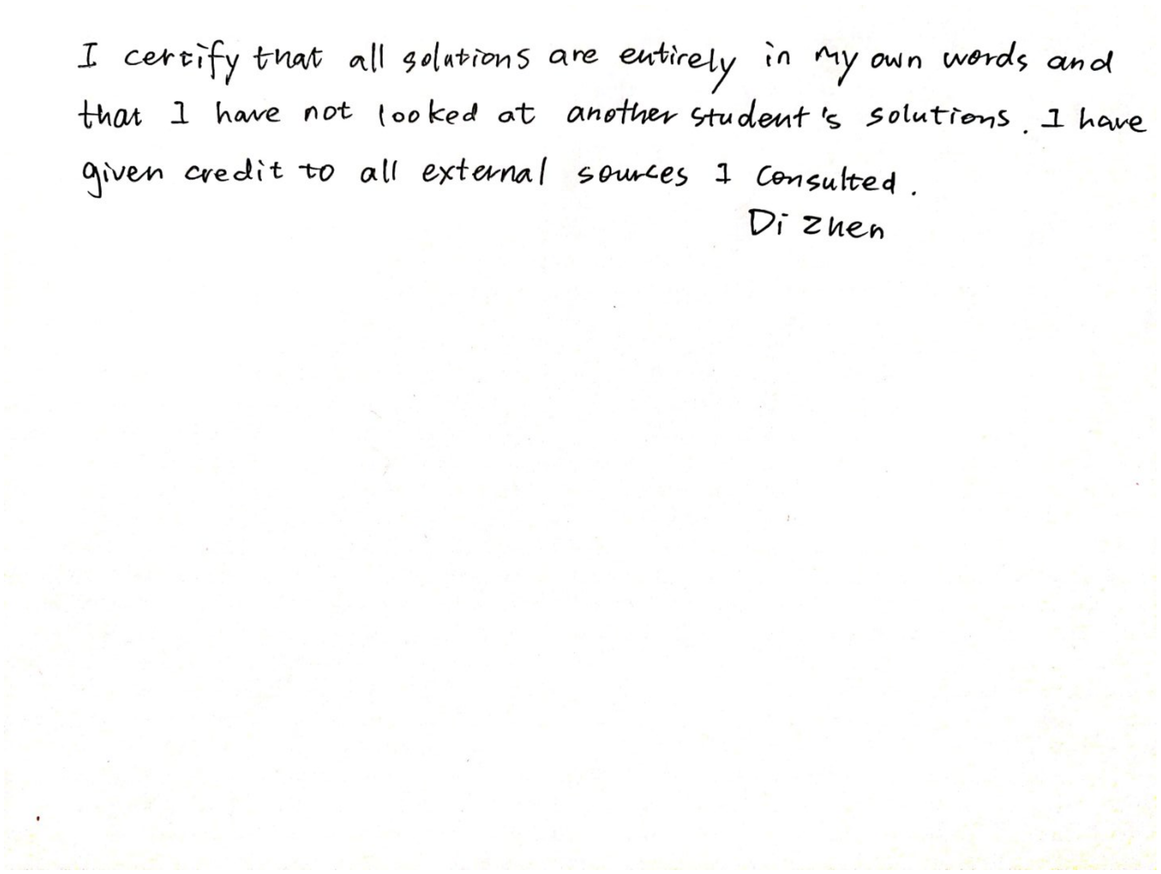


Figure 3: Error rate curve of training SPAM dataset by SVM after tuning hyperparameter C.

5



I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted.

Di Zhen