Assignment 4: Support Vector Machines

Ash

2018/12/06

1 Loading Data

Load the MNIST dataset using Keras API:

```
#Loading relevant packages
import numpy as np
import matplotlib.pyplot as plt
import keras
import matplotlib

#Loading the MNIST dataset from keras: the training set and test set
    contains 60,000 and 10,000
#28x28 images of 10 classes of handwritten digits respectively
from keras.datasets import mnist

(X_train, y_train), (X_test, y_test) = mnist.load_data()
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

--->Output:
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

After loading the full MNIST, choose 2s and 5s in both the training set and test set to train and test the SVMs:

```
#Choosing 2s and 5s among the training set to train SVMs
digit_2 = (y_train == 2)
X_train_2, y_train_2 = X_train[digit_2], y_train[digit_2]
digit_5 = (y_train == 5)
X_train_5, y_train_5 = X_train[digit_5], y_train[digit_5]
y_train_2 = np.ones(y_train_2.shape)
y_train_5 = np.zeros(y_train_5.shape)


#Casually checking the class balance
```

```
print('Total 2s:', X_train_2.shape[0])
print('Total 5s:', X_train_5.shape[0])

#Scale the training set to the range between 0 and 1
X_train_25 = np.concatenate((X_train_2, X_train_5), axis=0)/255
y_train_25 = np.concatenate((y_train_2, y_train_5), axis=0)
print(X_train_25.shape)
print(y_train_25.shape)

#Function to plot some digits of the MNIST data set
def plot_digits(X, y, m, n):
    '''
    Inputs:
    - X is the data, tensor of Nx28x28
    - y is the label, vector Nx1
    - m is the number of plot row
    - n is the number of plot columns

    Outputs: A plot of mxn digits randomly chosen from the input sets
    '''
    ndigits = np.random.choice(a=X.shape[0], size=m*n)
    count = 1
    plt.figure(figsize=(12,8))
    for _ in ndigits:
        plt.subplot(m,n,count)
        im = X[_]
        plt.imshow(im, cmap=matplotlib.cm.binary,
                   interpolation="nearest")
        plt.axis("off")
        plt.title(str(y[_]))
        count += 1
    plt.show()

plot_digits(X_train_25,y_train_25, 4, 4)
#Reshaping the data into a 2D tensor
X_train_25 = X_train_25.reshape(X_train_25.shape[0], 28*28)

#Choosing 2s and 5s among the test set, and reshaping to 2D tensor
digit_2 = (y_test == 2)
X_test_2, y_test_2 = X_test[digit_2], y_test[digit_2]
digit_5 = (y_test == 5)
X_test_5, y_test_5 = X_test[digit_5], y_test[digit_5]
y_test_2 = np.ones(y_test_2.shape)
y_test_5 = np.zeros(y_test_5.shape)


print('Total 2s:', X_test_2.shape[0])
print('Total 5s:', X_test_5.shape[0])

X_test_25 = np.concatenate((X_test_2, X_test_5), axis=0)/255
y_test_25 = np.concatenate((y_test_2, y_test_5), axis=0)

print(X_test_25.shape)
print(y_test_25.shape)
```

```
X_test_25 = X_test_25.reshape(X_test_25.shape[0], 28*28)

---->Output:

Total 2s: 5958
Total 5s: 5421
(11379, 28, 28)
(11379,)

Figure 1

---

Total 2s: 1032
Total 5s: 892
(1924, 28, 28)
(1924,)
```

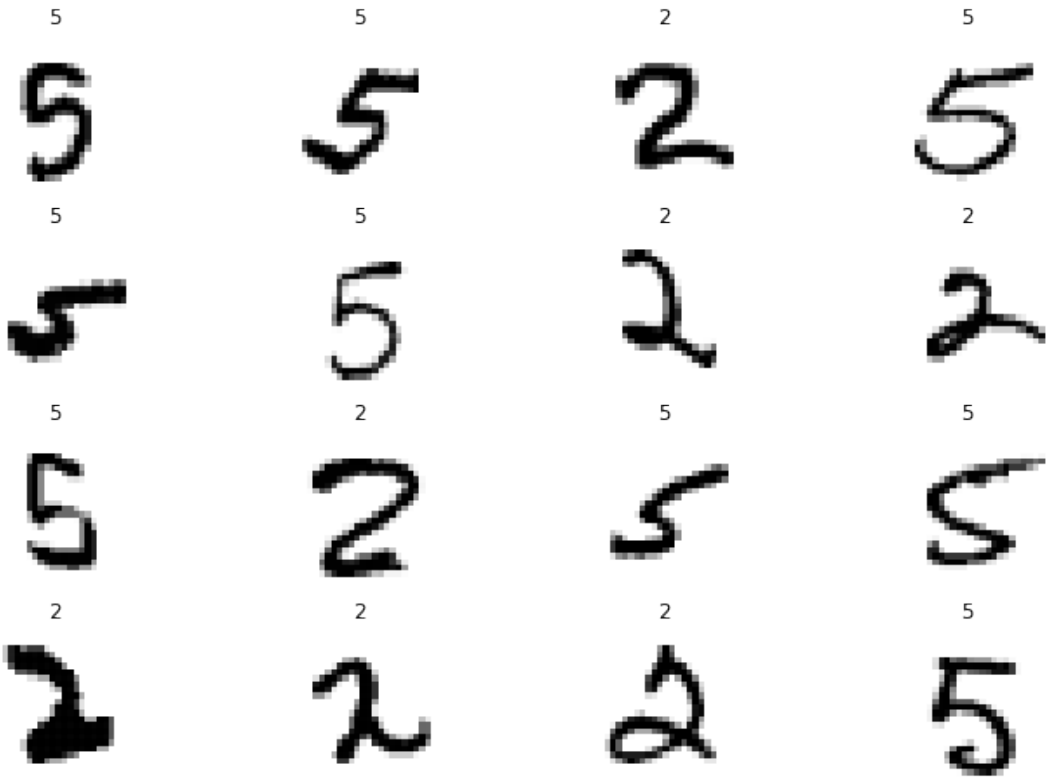


Figure 1: Random Images In The Training Set

2 Linear Kernels

```
#Using Scikit-learn implemented SVC and LinearSVC
from sklearn.svm import SVC, LinearSVC
import time

#Initiating LinearSVC with l2 regularization, dual=False to solve only the primal problem because
#n_feature < n_datapoint (28x28 < 60,000) makes dual problem optimizer slower in liblinear implementation
#(Training time doubles if dual=True, at around 0.5s)
linear_1 = LinearSVC(penalty='l2', C=1.0, dual=False)

#Fitting the 2s and 5s dataset, evaluating the accuracy on test and training set
start = time.time()
linear_1.fit(X_train_25, y_train_25)
print('Training time for LinearSVC:', time.time()-start)
print('Accuracy on train set for LinearSVC:', linear_1.score(X_train_25, y_train_25))
print('Accuracy on test set for LinearSVC:', linear_1.score(X_test_25, y_test_25))

#Initiating SVC class, with soft-margin SVC (C=1.0) and linear kernel
linear_2 = SVC(kernel='linear', C=1.0)

#Fitting the 2s and 5s dataset, evaluating the accuracy on test and training set
start = time.time()
linear_2.fit(X_train_25, y_train_25)
print('Training time for SVC, with linear kernel:', time.time()-start)
print('Accuracy on train set for SVC, with linear kernel:', linear_2.score(X_train_25, y_train_25))
print('Accuracy on test set for SVC, with linear kernel:', linear_2.score(X_test_25, y_test_25))

--->Output:

Training time for LinearSVC: 0.31829404830932617
Accuracy on train set for LinearSVC: 0.9954301783988048
Accuracy on test set for LinearSVC: 0.9823284823284824

Training time for SVC, with linear kernel: 14.031012058258057
Accuracy on train set for SVC, with linear kernel: 0.9923543369364619
Accuracy on test set for SVC, with linear kernel: 0.9838877338877339
```

For a binary classification task such as this, SVMs even with just linear kernel is doing extremely well: 99.5% accuracy on the training set and 98.2% accuracy on the test set, meaning that it's generalizing very well. With this result, it would be interesting to plot and see the some of the wrong classifications (there are 34 of them) in Figure 2. Some of these can be easily classified by humans though. In figure 3a and 3b, if we classify each digit against random noise and plot the optimal weights, we can can (very roughly) see the shapes of 2 and 5 in the optimal boundaries, given the most activated pixels for each of the digits when trained to classified against random noise. Interestingly,

there are much overlap between the "shapes" of the 2 and the 5, which indicates that the handwritten dataset for 2s and 5s vary quite a lot for each digit (which makes sense realistically since people write 2s and 5s in very different ways). When comparing this overlapping to other digits weights, we can clearly see that the SVMs are learning something useful, and it just happens that for 2s and 5s in this dataset there is much overlap, at least when using a linear kernel.

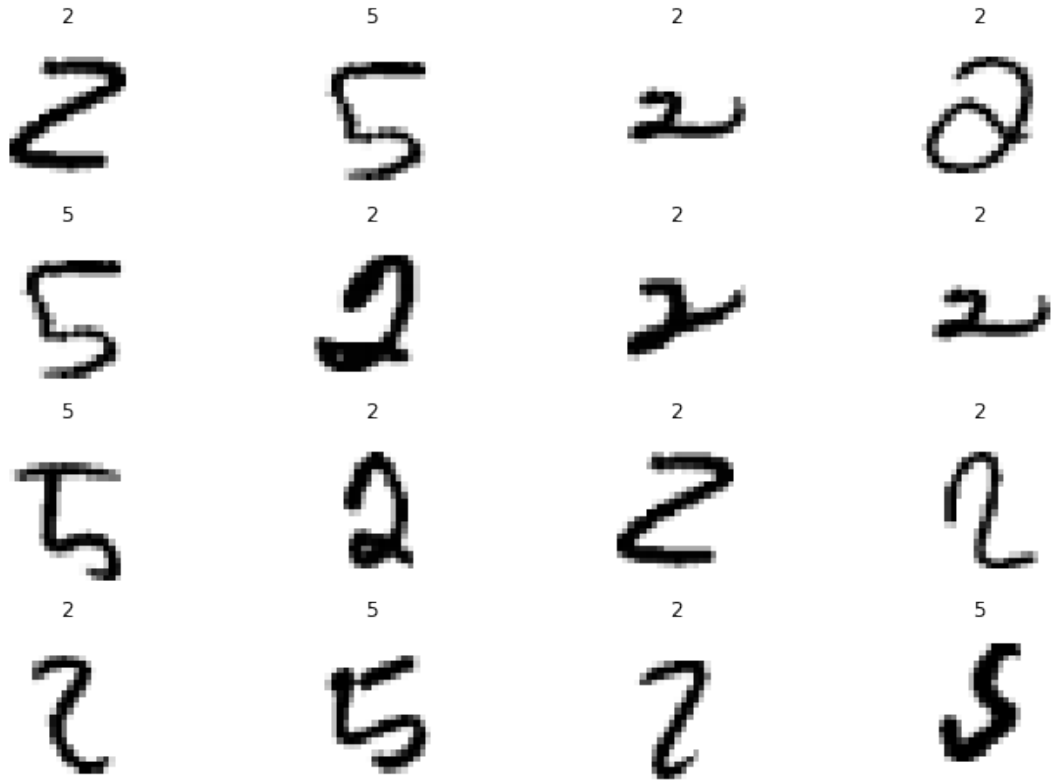
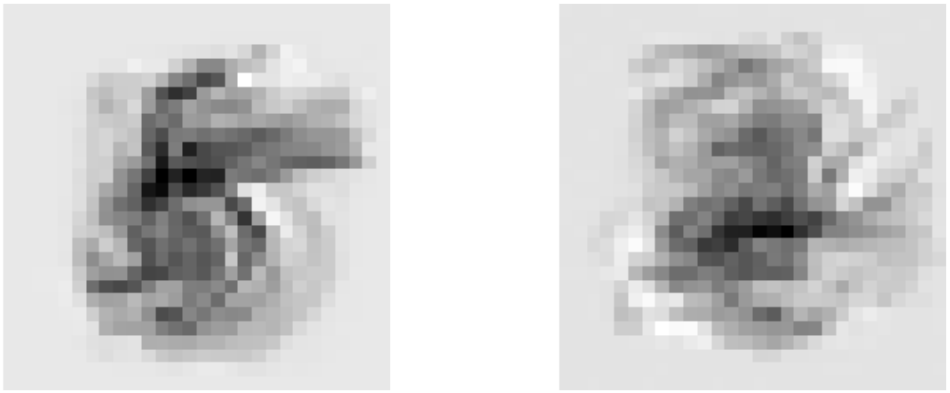


Figure 2: Wrong Classification On The Test Set

```
plt.figure(figsize=(12,8))
for _ in range(0,10):
    digit_1 = (y_train == _)
    X_train_1, y_train_1 = X_train[digit_1], y_train[digit_1]
    y_train_1 = np.ones(y_train_1.shape)

    X_train_noise = np.abs(np.random.normal(size=X_train_5.shape))
    y_train_noise = np.zeros(y_train_5.shape)
    X_train_1noise = np.concatenate((X_train_1, X_train_noise), axis=0)/255
    y_train_1noise = np.concatenate((y_train_1, y_train_noise), axis=0)
    X_train_1noise = X_train_1noise.reshape(X_train_1noise.shape[0], 28*28)

    linear = LinearSVC(penalty='l2', C=1.0, dual=False)
```



(a) 5 (b) 2

Figure 3: Optimal Weights Visualization For 2 and 5

```
    linear.fit(X_train_1noise, y_train_1noise)
    lala = linear.coef_
    plt.subplot(5,2,_+1)
    im = lala.reshape(28,28)
    plt.imshow(im, cmap=matplotlib.cm.binary)
    plt.axis("off")

plt.show()

--->Output:

Figure 3 and 4
```

3 Polynomial Kernels

```
#Initiating the SVC, with soft-margin SVC (C=1.0) and quadratic kernel, and automatic gamma
poly_quad = SVC(kernel='poly', C=1.0, degree=2, gamma='scale')

start = time.time()
poly_quad.fit(X_train_25, y_train_25)
print('Training time for SVC, with polynomial 2 kernel:', time.time()-start)
print('Accuracy on train set for SVC, with polynomial 2 kernel:', poly_quad.score(X_train_25, y_train_25))
print('Accuracy on test set for SVC, with polynomial 2 kernel:', poly_quad.score(X_test_25, y_test_25))

#Initiating the SVC, with soft-margin SVC (C=1.0) and cubic kernel, and automatic gamma
poly_cube = SVC(kernel='poly', C=1.0, degree=3, gamma='scale')
```

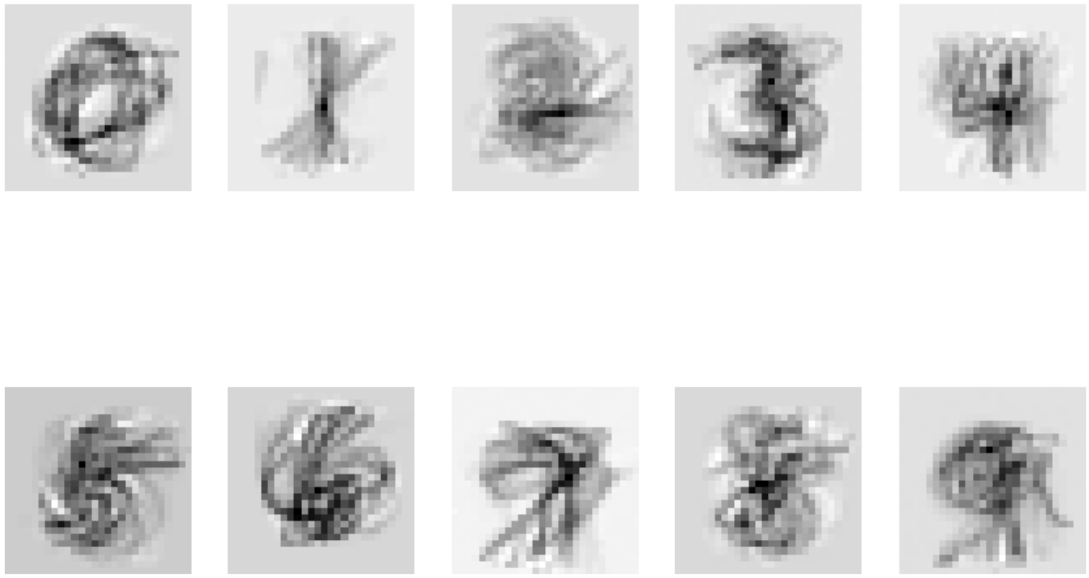


Figure 4: Optimal Weights Visualization For The MNIST Dataset

```
12
13 start = time.time()
14 poly_cube.fit(X_train_25, y_train_25)
15 print('Training time for SVC, with polynomial 3 kernel:', time.time()-start
       )
16 print('Accuracy on train set for SVC, with polynomial 3 kernel:', poly_cube
       .score(X_train_25, y_train_25))
17 print('Accuracy on test set for SVC, with polynomial 3 kernel:', poly_cube.
       score(X_test_25, y_test_25))
18
19 ---->Output:
20
21 Training time for SVC, with polynomial 2 kernel: 21.592259168624878
22 Accuracy on train set for SVC, with polynomial 2 kernel: 0.9920028121979084
23 Accuracy on test set for SVC, with polynomial 2 kernel: 0.9922037422037422
24
25 Training time for SVC, with polynomial 3 kernel: 43.53636693954468
26 Accuracy on train set for SVC, with polynomial 3 kernel: 0.9757447930398102
27 Accuracy on test set for SVC, with polynomial 3 kernel: 0.974012474012474
```

4 Gaussian Kernel

```
1 #Initiating SVC class, with soft-margin SVC (C=1.0) and gaussian kernel
2 gau_rbf = SVC(kernel='rbf', C=1.0, gamma='scale')
3
4 start = time.time()
5 gau_rbf.fit(X_train_25, y_train_25)
6 print('Training time for SVC, with Gaussian kernel:', time.time()-start)
```

```
print('Accuracy on train set for SVC, with Gaussian kernel:', gau_rbf.score
    (X_train_25, y_train_25))
print('Accuracy on test set for SVC, with Gaussian kernel:', gau_rbf.score(
    X_test_25, y_test_25))


--->Output:

Training time for SVC, with Gaussian kernel: 13.312834024429321
Accuracy on train set for SVC, with Gaussian kernel: 0.9935846735213991
Accuracy on test set for SVC, with Gaussian kernel: 0.9927234927234927
```

5 Discussion

There are two things worthy of discussion here. First, regarding the training time of each implementation, the LinearSVC class implemented with liblinear optimizer is clearly advantageous here because it can deals with a bigger dataset (scales almost linearly with the number of datapoints - see appendix) whereas the kernelized SVC class scales at least quadratically (sometimes up to cubically) with the number of datapoints, and we have a dataset of more than 10,000 examples to train here, hence the slowness of the SVC class. However, since LinearSVC (and liblinear) cannot support the kernel trick (for anything other than the simple linear kernel), in general SVC class is still the better choice.

Second, regarding the accuracy of these SVMs in binary classification of the MNIST dataset and theirs awesome generalization on the test set: it does makes sense since even without transformation the number of feature is 784, thus as analyzed in my first assignment using PCA projection it is very likely that in high dimensional space these digits are linearly separable, resulting in a really good accuracy. It is also worth pointing out that the cubic polynomial had worse results than the other kernel (97% compared to around 99% in other kernels). This is maybe because there is a default tolerance and max iteration for the optimizer when it was wandering around the high-dimensional feature spaces, and it stopped earlier in the cubic kernel (compared to the others) due to the very high dimensional space is was wandering in without improvement in the objective function (784 dimensions, expanded into cubic!). Also, the soft-margin weighting was set to 1.0 for all kernels, but if I increase the weighting of the soft-margin (allowing fewer and fewer margin violations, meaning letting fewer and fewer training data to be inside the "street" by penalizing the violations harder with bigger C, thereby decreasing the size of the "street") the linear SVM starts to overfit because the "street" is now too small and too specific for the training data, creating a worse generalization on the test set. This is clearly shown in Figure 5.

```
#Soft-margin weighting: testing 198 different weigthtings for margin
    violations
cs = np.arange(1,100,0.5)
times, accs_train, accs_test = [], [], []

for _ in cs:
```

```
    linear_cs = LinearSVC(penalty='l2', C=_, dual=False)
    start = time.time()
    linear_cs.fit(X_train_25, y_train_25)
    times.append(time.time()-start)
    accs_test.append(linear_cs.score(X_test_25, y_test_25))
    accs_train.append(linear_cs.score(X_train_25, y_train_25))

plt.figure(figsize=(10,8))
plt.subplot(2,1,1)
plt.plot(cs, times, linestyle=':', color='r', label='Time')
plt.legend()
plt.title('Training Time', loc='left')
plt.xlabel('Soft-Margin Weighting')
plt.ylabel('Time (s)')
plt.subplot(2,1,2)
plt.plot(cs, accs_train, color='b', label='Train Accuracy')
plt.plot(cs, accs_test, color='g', label='Test Accuracy')
plt.legend()
plt.title('Training & Test Accuracy', loc='left')
plt.xlabel('Soft-Margin Weighting')
plt.ylabel('Accuracy')
plt.show()

--->Output:

Figure 5
```

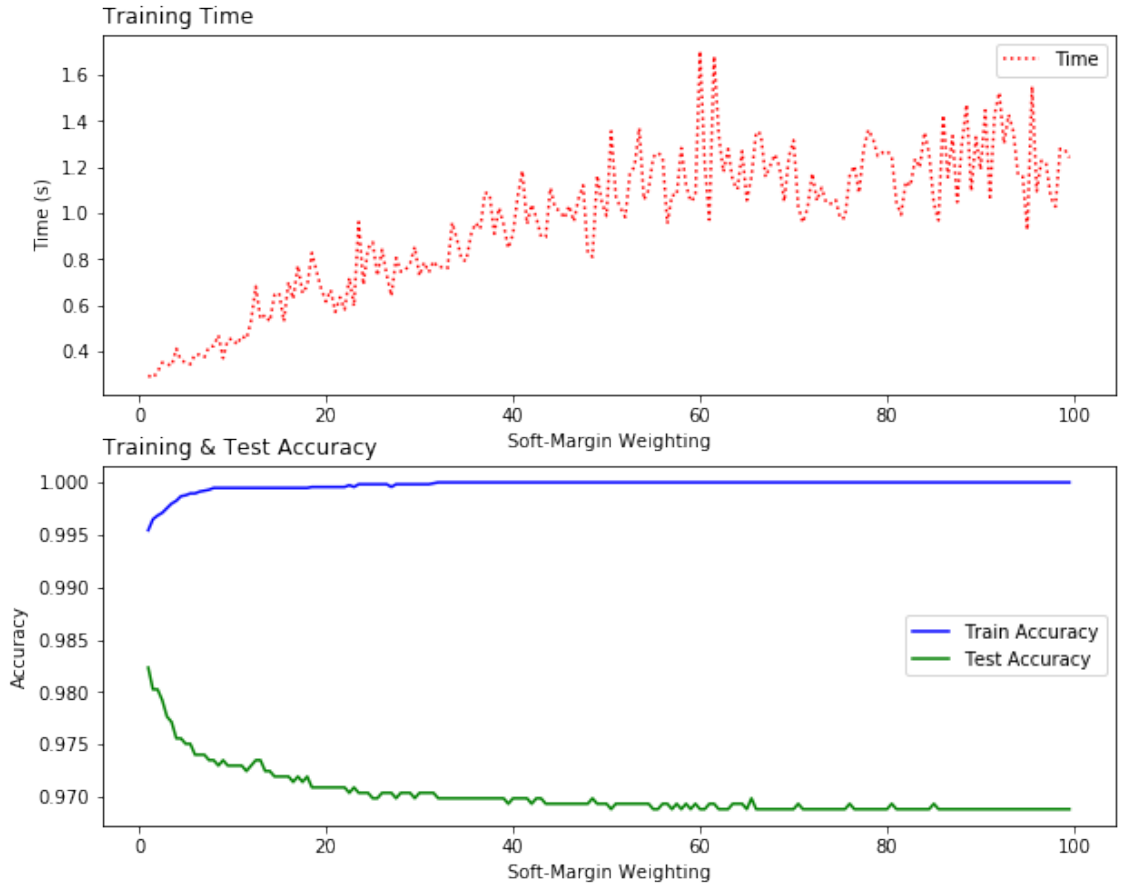
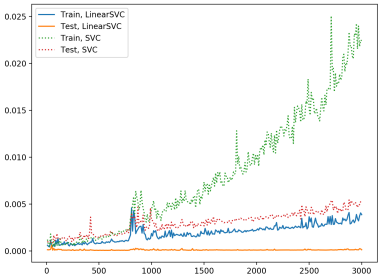
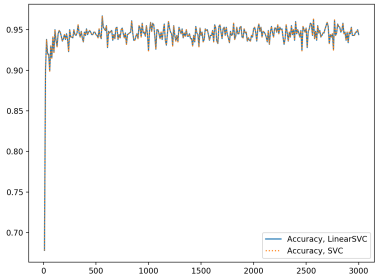


Figure 5: Overfitting With Soft-Margin Penalization

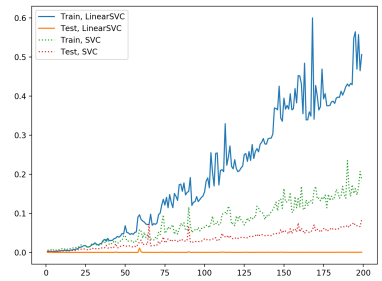


(a) Training and testing time

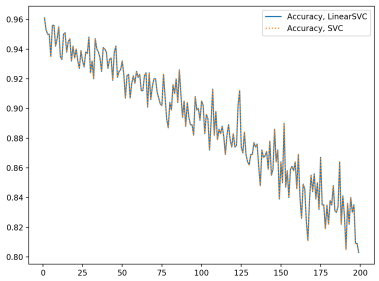


(b) Accuracy

Figure 6: Scaling With Number Of Datapoint N



(a) Training and testing time



(b) Accuracy

Figure 7: Scaling With Number Of Dimension D

Appendix

Traing and testing time of different implementations of Scikit-learn SVMs:

```
#Generating Data with N number of datapoints and D dimensions, using
    mixture of Gaussians
def data_generator(N, D=1, violation_ratio=0.05):
    noise = np.random.normal(loc=0, scale=violation_ratio, size=(N+1000,D))
    inputs = np.random.rand(N+1000, D)
    centers = np.array([inputs.max(axis=0),inputs.min(axis=0)])
#     centers = np.random.rand(2,D)
    dumb = rbf_kernel(inputs, centers, gamma=1)
    outputs = np.zeros(shape=(N+1000,1))
    outputs[dumb[:,0]>dumb[:,1]] = 1
    inputs = inputs + noise
    return centers, inputs[:N,:], outputs[:N,:].reshape(-1,), inputs[N:,:],
     outputs[N:,:].reshape(-1,)

def map_color(y):
    colors = []
    for _ in range(len(y)):
```

```
        if y[_] == 1:
            colors.append('r')
        else:
            colors.append('b')
    return colors


import time
svc_1 = LinearSVC()
svc_2 = SVC(kernel='linear')

train_t_1, test_t_1, acc_1 = [], [], []
train_t_2, test_t_2, acc_2 = [], [], []

r = range(10,3001,10)

for _ in r:
    c, X, y, X_test, y_test= data_generator(_,2)

    start = time.time()
    svc_1.fit(X,y)
    train_t_1.append(time.time()-start)
    start = time.time()
    svc_2.fit(X,y)
    train_t_2.append(time.time()-start)

    start = time.time()
    svc_1.predict(X_test)
    test_t_1.append(time.time()-start)
    start = time.time()
    svc_2.predict(X_test)
    test_t_2.append(time.time()-start)

    acc_1.append(svc_1.score(X_test,y_test))
    acc_2.append(svc_1.score(X_test,y_test))

plt.figure(figsize=(8,6))
plt.plot(r,train_t_1, label='Train, LinearSVC')
plt.plot(r,test_t_1, label='Test, LinearSVC')
plt.plot(r,train_t_2, label='Train, SVC', linestyle=':')
plt.plot(r,test_t_2, label='Test, SVC', linestyle=':')
plt.legend()
plt.show()

svc_1 = LinearSVC()
svc_2 = SVC(kernel='linear')

train_t_1, test_t_1, acc_1 = [], [], []
train_t_2, test_t_2, acc_2 = [], [], []

r = range(1,200)

for _ in r:
    c, X, y, X_test, y_test= data_generator(1000,_)
```

```
70
71      start = time.time()
72      svc_1.fit(X,y)
73      train_t_1.append(time.time()-start)
74      start = time.time()
75      svc_2.fit(X,y)
76      train_t_2.append(time.time()-start)
77
78      start = time.time()
79      svc_1.predict(X_test)
80      test_t_1.append(time.time()-start)
81      start = time.time()
82      svc_2.predict(X_test)
83      test_t_2.append(time.time()-start)
84
85      acc_1.append(svc_1.score(X_test,y_test))
86      acc_2.append(svc_1.score(X_test,y_test))
87
88  plt.figure(figsize=(8,6))
89  plt.plot(r,train_t_1, label='Train, LinearSVC')
90  plt.plot(r,test_t_1, label='Test, LinearSVC')
91  plt.plot(r,train_t_2, label='Train, SVC', linestyle=':')
92  plt.plot(r,test_t_2, label='Test, SVC', linestyle=':')
93  plt.legend()
94  plt.show()
95
96
97  ---->Output:
98  Figure 6 and Figure 7
```