Machine Learning

Summer Semester 2017, Homework 4

Prof. Dr. J. Peters, M. Ewerton, S. Parisi

TECHNISCHE UNIVERSITÄT DARMSTADT

Total points: 55 + 12 bonus
Due date: Tuesday, 18 July 2017 (before the lecture)

Name, Surname, ID Number Wang Yujue, 2573991 Zhi Rong, 2806891

Problem 4.1 Support Vector Machines [25 Points + 4 Bonus]

a) Definition [3 Points]

Briefly define SVMs. What is their advantage w.r.t. other linear approaches we discussed this semester?

SVM is widely perceived as one of the most powerful learning algorithms, and it is a very effective way to learn complex non-linear functions.
Compared to both logistic regression and neural networks, the Support Vector Machine sometimes gives a cleaner, and more powerful way of learning complex non-linear functions.
And with the margin of the support vector machine, this gives the SVM a certain robustness, because it tries to separate the data with as a large a margin as possible.

b) Quadratic Programming [2 Points]

Formalize SVMs as a constrained optimization problem.

$$min_\theta \tfrac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \tfrac{1}{2}||\theta||^2$$
$$s.t. \quad \theta^T x^{(i)} \geq 1 \quad if\, y^{(i)} = 1 \qquad \theta^T x^{(i)} \geq -1 \quad if\, y^{(i)} = 1$$

c) Slack Variables [2 Points]

Explain the concept behind slack variables and reformulate the optimization problem accordingly.

If our data is not linearly separable.
Solution: Allow for data points to violate the margin.
We allow for small violations ξ_i from perfect separation

$$w^T x_i + b \geq 1 - \xi_i \text{ if } y_i = 1 \qquad w^T x_i + b \leq -1 + \xi_i \text{ if } y_i = -1 \qquad \xi_i \geq 0$$
$$y_i\left(w^T x_i + b\right) \geq 1 - \xi_i \qquad \xi \geq 0$$

ξ_i are slack variables
Penalize the deviations

$$min_{w,b} \tfrac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i$$
$$s.t. \qquad y_i\left(w^T x_i + b\right) \geq 1 - \xi_i \qquad \xi \geq 0$$

Maximize the margin while minimizing the penalty for all data points that are not outside the margin.

d) Slack Variables [7 Points]

Solve the optimization problem of the previous question. Show all the intermediate steps and write down the final solution.

$$min_{w,b}\frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i \tag{10}$$

$$s.t. \qquad y_i\left(w^T x_i + b\right) \geq 1 - \xi_i \qquad \xi \geq 0 \tag{11}$$

I would rather set x_1 to be one, so the entire term $w^T x_i + b$ can simply be $w^T x_i$. Where $w_1 = b$. Lagrange multiplier

$$L(w,\lambda) = L(w_1, w_2, ..., w_n, \lambda_1, \lambda_2, ..., \lambda_n) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\lambda_i[y_i\left(w^T x_i\right) - 1 + \xi_i] \tag{12}$$

Now we can calculate the gradient:

$$\nabla_{w,\lambda}L(w_1, w_2, ..., w_n, \lambda_1, \lambda_2, ..., \lambda_n) = \left(\frac{\partial L}{\partial w}, \frac{\partial L}{\partial \lambda}\right) = \left(\sum_{i=1}^{N} w_i - \sum_{i=1}^{N}\lambda\lambda_i y_i x_i, \sum_{i=1}^{N} y_i\left(w^T x_i\right) - 1 + \xi_i\right) \tag{13}$$

therefore

$$\nabla_{w,\lambda}L(w_1, w_2, ..., w_n, \lambda_1, \lambda_2, ..., \lambda_n) = 0 \tag{14}$$

leads to

$$\sum_{i=1}^{N} w_i - \sum_{i=1}^{N}\lambda_i y_i x_i = 0 \tag{15}$$

$$\sum_{i=1}^{N} y_i\left(w^T x_k\right) - 1 + \xi_i = 0 \tag{16}$$

derive w with an equivalent notation, where $0 \leq \lambda_i \leq C$

$$w = \sum_{i=1}^{N}\lambda_i y_i x_i \tag{17}$$

And of cause it can be performed by a dual function, the separating hyperplane is given by the support vectors. It results following, where $0 \leq \lambda_i \leq C$

$$w = \sum_{i=1}^{N_S}\lambda_i y_i x_i \tag{18}$$

e) The Dual Problem [4 Bonus Points]

What are the advantages of solving the dual instead of the primal?

Both the primal as well as the derived dual formulation are quadratic programming problems. To go beyond linear classifiers, we choose dual.

f) Kernel Trick [3 Points]

Explain the kernel trick and why it is particularly convenient in SVMs.

Replace every occurrence of a scalar product between features with a kernel function.
If we can find a kernel function that is equivalent to this scalar product, we can avoid mapping into a high-dimensional space and instead compute the scalar-product directly.
The question is, is there a different choice of features or is there better sort of features than this high order polynomials because it's not clear that this high order polynomial is what we want. And in computer vision when the input is an image with lots of pixels, using high order polynomials becomes very computationally expensive because there are a lot of these higher order polynomial terms.
We can learn pretty complex non-linear decision boundary. Which is that we define these extra features using landmarks and similarity functions to learn more complex nonlinear classifiers.
We could use it to define new features for the Support Vector Machine. By using the Gaussian kernel with the SVM, we will be able to learn a non-linear decision boundary that can perform reasonably well for the dataset.

g) Implementation [8 Points]

Learn an SVM to classify the data in `iris-pca.txt`. Choose your kernel. Create a plot showing the data, the support vectors and the decision boundary. Show also the misclassified samples. Attach a snippet of your code.

I use Linear kernel and Gaussian RBF kernel respectively.
For Gaussian Kernel, $\sigma = 2$. I also tried smaller value, but in my opinion it is more likely to overfitting. Even if this is a not linearly separable case, the separation should be work as a classifier, treating the non separable points which are out of the margin as outliers seems more logical to me.

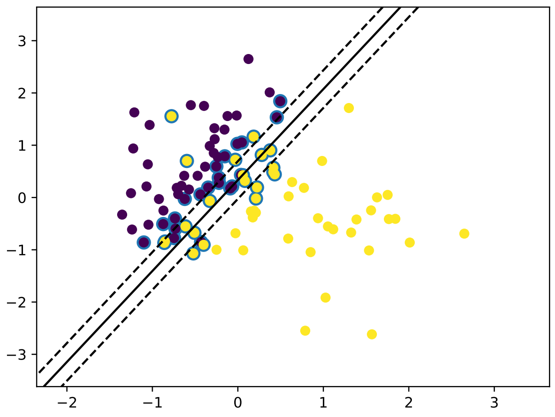

Figure 1: Linear Kernel

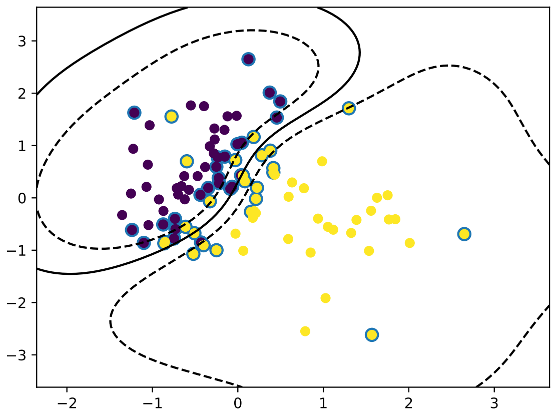

Figure 2: Gaussian Kernel

For Gaussian Kernel, $\sigma = 2$.
The support vectors are marked with blue circles.

Problem 4.2 Neural Networks [20 Points + 8 Bonus]

In this exercise, you will use the dataset `mnist_small`, divided into four files. The mnist dataset is widely used as benchmark for classification algorithms. It contains 28x28 images of handwritten digits (pairs <`input, output`> correspond to <`pixels, digit`>).

a) Multi-layer Perceptron [20 Points]

Implement a neural network with one hidden layer and train it using backpropagation on the provided dataset. Choose your loss function, activation function and optimizer, briefly explaining your choices. You can use off-the-shelf optimizers, loss and activation functions, but you have to write the network structure and the backpropagation algorithm by yourself. You are also free to choose a suitable number of neurons for the hidden layer.

Show how the misclassification error (in percentage) on the testing set evolves during the learning. Attach snippets of your code.

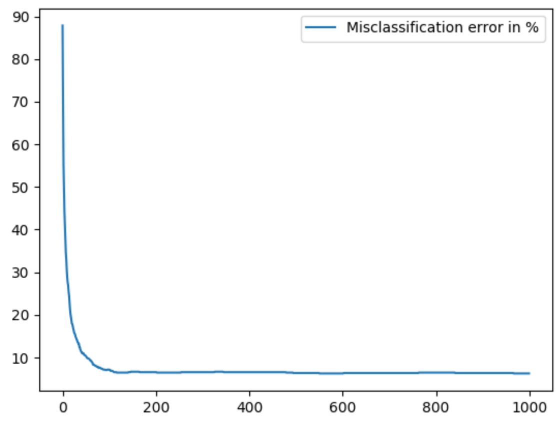

Figure 3: Misclassification Error on Test Set

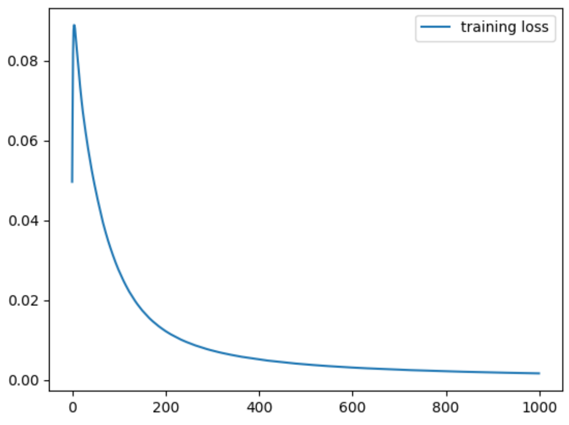

Figure 4: Loss Function

Misclassification error on test set: 6.375% Optimizer: SGD, Loss: Cross entropy loss, Activation: Relu (differentiable when x > 0) performs better than sigmoid
One hidden layers with 512 neurons
Training step: 1000

b) Deep Learning [8 Bonus Points]

In recent years, deep neural networks have become one of the most used tools in machine learning. Highlight the qualitative differences between classical neural networks and deep networks. Which limitations of classical NN does deep learning overcome? Give an intuition of the innovations introduced in deep learning compared to traditional NN. (Hint: Have a look at this paper. Search on Google Scholar and other scientific papers for more insights.)

Deep Learning can Learn abstract features and hierarchies of representations and very complex functions. It can reuse features.
But vary computational expensive and sample-wise expensive. Tend to overfitting. Hyper-parameters choose and optimization is hard and tricky. Also it encounters Vanishing gradients.

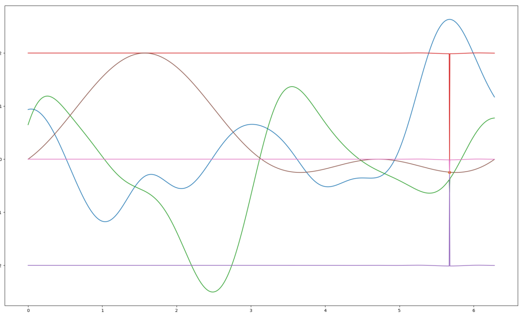

Figure 5: GP of the First Iteration

Problem 4.3 Gaussian Processes [10 Points]

a) GP Regression [10 Points]

Implement a Gaussian Process to fit the target function $y = \sin(x) + \sin^2(x)$ with $x \in [0, 0.005, 0.01, 0.015, \ldots, 2\pi]$. Use a squared exponential kernel, an initial mean of 0 and assume a noise variance of 0.001. Begin with no target data points and, at each iteration, sample a new point from the target function according to the uncertainty of your GP (that is, sample the point where the uncertainty is the highest) and update it. Plot your GP (mean and two times standard deviation) after iterations 1, 2, 4, 8 and 16. In each figure, plot also the true function as ground truth and add a new marker for each new sampled point. Attach a snippet of your code.

Please be more specific about this question. I use the algorithm in the video you suggested. But there is a vital problem with this question: the scale of X is annoying. It is hard to choose proper hyper-parameter to avoid overfitting and underfitting. For the length scale, I found it should be less than 0.5 and bigger than 0.1 to avoid overfitting and underfitting. However with this small l, regarding the squared exponential kernel, the value is very small, plus that the variance of the kernel should be set around 2 to 3 to fulfill the ground truth. The prior kernel is very small. This leads to very small posterior kernel and conditioning kernel. Which results very small mean and iteration update mean and variance in the predictive distribution. Moreover, with this very small number and unreasonable length scale. Program gives errors when computing the matrix inverse of the posterior kernel if the iteration goes over 6 times.
Anyway, I am showing a plot of one iteration with ground truth (Brown), the random prior (Blue), the sample point (Orange dot) from the target function according to the uncertainty of this GP, the posterior (Green), the 2 times standard deviation (Red and Purple) and the mean (Magenta).
Where you can see that variance collapse at the sample point, mean also adjusts, and the posterior curve change a lot according to this sampled ground truth value.

Code:

Problem 4.1

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

# read data
def read_dataset(path):
    data = []
    txt = open(path)
    for line in txt:
        a,b,c = map(float ,line.split())
        data.append([a,b,c])
    return np.asarray(data)

def gaussianKernel(x1 ,x2):
    sigma = 2
    sim = np.exp(-sum(np.subtract(x1, x2) ** 2) / (2 * sigma ** 2))
    return sim

data = read_dataset('dataSets/iris-pca.txt')
Y = data[:,2]
X = data[:,[0,1]]
s = 0.02 # mesh step size
fignum = 1

# fit the model
for kernel in ('linear', 'gaussianKernel'):
    clf = svm.SVC(kernel=kernel, C = 1)
    clf.fit(X, Y)

    # plot the line, the points, and the nearest vectors to the plane
    plt.figure(fignum)
    plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=80)
    plt.scatter(X[:, 0], X[:, 1], c=Y)

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, s), np.arange(y_min, y_max, s))
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure(fignum)
    plt.contour(xx, yy, Z, colors=['k', 'k', 'k'], linestyles=['--', '-', '--'],
                levels=[-.5, 0, .5])

    fignum = fignum + 1

plt.show()
```

Problem 4.2

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import progressbar as pb

num_class = 10
img_size = 28 * 28
num_neu = 512
epoch = 1000

def read(path):
    data = []
    with open(path,'r') as f:
        f.seek(0)
        for line in f:
            tmp = [float(i) for i in line.strip().split(',')]
            data.append(tmp)
    return data

def one_hot(v):
    label = []
    for i in range(len(v)):
        tmp = np.zeros(num_class)
        tmp[v[i]] = 1
        label.append(tmp)
    return label


def init_weights(shape):
    return tf.Variable(tf.random_normal(shape,stddev=0.1))

def model(X,w_h,w_o,b_h,b_o):
    h = tf.nn.relu(tf.add(tf.matmul(X,w_h),b_h))
    return tf.add(tf.matmul(h,w_o),b_o)

X = tf.placeholder(tf.float32,[None,img_size])
Y = tf.placeholder(tf.float32,[None,num_class])
# y_true_cls = tf.placeholder(tf.int64,[None,1])

w_h = init_weights([img_size,num_neu])
w_o = init_weights([num_neu,num_class])

b_h = init_weights([num_neu])
b_o = init_weights([num_class])

py_x = model(X, w_h, w_o, b_h, b_o)

y_pred = tf.nn.softmax(py_x)
y_pred_cls = tf.argmax(py_x,dimension=1)

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=py_x,labels=Y))
train_op = tf.train.GradientDescentOptimizer(0.05).minimize(cost)

correct_prediction = tf.equal(y_pred_cls,tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction,tf.float32))

train_x = read("dataSets/mnist_small_train_in.txt")
```

```
train_tmp = read("dataSets/mnist_small_train_out.txt")
train_y_tmp = [int(i[0]) for i in train_tmp]

test_x = read("dataSets/mnist_small_test_in.txt")
test_tmp = read("dataSets/mnist_small_test_out.txt")
test_y_tmp = [int(i[0]) for i in test_tmp]


train_y = one_hot(train_y_tmp)
test_y = one_hot(test_y_tmp)

# print(len(train_x))
Losses = []
ACC = []
with tf.Session() as sess:

    tf.global_variables_initializer().run()
    for i in range(epoch):
        for start,end in zip(range(0,len(train_x),128),
                             range(128,len(train_x)+1,128)):
            feed_dict_train = {X:train_x[start:end],
                               Y:train_y[start:end]}
            sess.run(train_op,feed_dict=feed_dict_train)

        # loss = sess.run(cost, feed_dict=feed_dict_train)
        # print("Step:{0}, Training loss:{1}".format(i,loss))
        # Losses.append(loss)

        # if i % 20 == 0:
            # acc_train = sess.run(accuracy,feed_dict=feed_dict_train)
            # print("Step:{0}, Training accuracy:{1}".format(i,acc_train))

        feed_dict_test = {X: test_x, Y: test_y}
        acc_test = sess.run(accuracy, feed_dict_test)
        ACC.append((1-acc_test)*100)
    # print("Misclassification error on test set: {0:.3f}%".format(100*(1-acc_test)))

x_step = [i for i in range(epoch)]

plt.plot(x_step,ACC,label='Misclassification␣error␣in␣%')
plt.legend()
plt.show()
```

Problem 4.3

```
import numpy as np
import matplotlib.pyplot as plt


def SE_Kernel(t1 ,t2, sigma_f, l):
    K_ff = sigma_f ** 2 * np.exp(-1 / (2 * l ** 2) * (t1 - t2) ** 2)
    return K_ff

def Delta(t1, t2):
    if t1 == t2:
        d = 1
    else:
        d = 0
    return d

def Noise_Kernel(t1 ,t2, sigma_n):
    K_nn = sigma_n ** 2 * Delta(t1, t2)
    return K_nn

x = np.arange(0.0, 2.0 * np.pi, 0.005)
y = np.sin(x) + np.sin(x)**2
sigma_f = 1 # set appropriate std according to the ground truth
l = 0.5 # choose proper hyperparameter to avoid overfitting and underfitting
sigma_n = np.sqrt(0.001) # noise variance

# first plot prior
N = len(x)
K_ff = np.zeros([N, N])
K_nn = np.zeros([N, N])
mean = np.zeros(N)

for i in range(0, N):
    for j in range(0, N):
        K_ff[i, j] = SE_Kernel(x[i], x[j], sigma_f, l)
        K_nn[i, j] = Noise_Kernel(x[i], x[j], sigma_n)

f = np.random.multivariate_normal(mean, K_ff)
plt.figure()
plt.plot(x, f)

K_yy = K_ff + K_nn
K_fy = np.zeros([N, 1])
sigma2 = K_ff # initial value
#mu = mean # initial value
for ite in range(0, 1):
    s_p = np.argmax(np.abs(f))
    x_s = x[s_p] # sample point
    y_s = np.sin(x_s) + np.sin(x_s)**2 # sample
    for i in range(0, N):
        K_fy[i] = SE_Kernel(x[i], x_s, sigma_f, l)

    mu = np.dot(np.transpose(K_fy), np.linalg.inv(K_yy)) * (y_s - np.mean(y))
    mu = np.reshape(mu, N)
    sigma2[s_p,:] = K_ff[:,s_p] - np.dot(np.dot(np.transpose(K_fy), np.linalg.inv(K_yy)), K_fy)
    f = np.random.multivariate_normal(mu, sigma2)
    plt.plot(x_s, y_s, 'o')
    plt.plot(x, f) # prediction
    plt.figure()
    ite = ite + 1

std = np.zeros(N)
for j in range(0, N):
    std[j] = np.sqrt(sigma2[j, j])

plt.plot(x, mu + 2 * std), plt.plot(x, mu - 2 * std)
plt.plot(x, y), plt.plot(x, mu)

plt.show()
```