Pattern Recognition Homework 3

İpek Erdoğan

January 18, 2021

Kernel Trick

$$\max_{a\in\mathbb{R}^{\mathbb{N}}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{m=1}^{N}\sum_{n=1}^{N} \alpha_n \alpha_m y_n y_m K(x_n, x_m)$$

which in this problem, equals to

$$\max_{a\in\mathbb{R}^{\mathbb{N}}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{m=1}^{N}\sum_{n=1}^{N} \alpha_n \alpha_m y_n y_m (x_n^T x_m + 1)^2$$

subject to

$$\sum_{n=1}^{N} y_n a_n = 0, a_n \geq 0, \forall n$$

In this problem,

$$X = \begin{bmatrix} -1 & -1 \\ -1 & 1 \\ 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Here, x_1 = [-1,-1] , x_2 = [-1,1] ,x_3 = [1,1] ,x_4 = [1,-1]

Kernel matrix which represents the inner product is [1] [2]:

$$\begin{bmatrix} (x_1^2+1)^2 & (x_1x_2+1)^2 & (x_1x_3+1)^2 & (x_1x_4+1)^2 \\ (x_2x_1+1)^2 & (x_2^2+1)^2 & (x_2x_3+1)^2 & (x_2x_4+1)^2 \\ (x_3x_1+1)^2 & (x_3x_2+1)^2 & (x_3^2+1)^2 & (x_3x_4+1)^2 \\ (x_4x_1+1)^2 & (x_4x_2+1)^2 & (x_4x_3+1)^2 & (x_4^2+1)^2 \end{bmatrix}$$

$$\begin{bmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{bmatrix}$$

According to the Mercer's conditions, this kernel matrix need to be symmetrical and positive semi definite for any given x_n. To check the positive semi definite condition of the matrix, we can check the eigenvalues. If eigenvalues are all positive, it means our matrix is positive semi definite. For the eigenvalue calculation, I used Numpy library.

```
A = np.array([[9,1,1,1],[1,9,1,1],[1,1,9,1],[1,1,1,9]])
eigenvalues, eigenvectors = numpy.linalg.LA.eig(A)
```

As a result, the eigenvalues of the matrix are = [8,12,8,8]. Since all of the eigenvalues are positive, our kernel matrix is positive semi definite. And since it's transpose is equal to itself, it's also symmetric. That means this kernel matrix satisfies the Mercer's condition.

$$K(a,b) = (\begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix}^T \begin{pmatrix} x_{21} \\ x_{22} \end{pmatrix} + 1)^2$$

$$\phi(x_1)^T\phi(x_2) = \begin{bmatrix} 1 \\ \sqrt{2}x_{11} \\ \sqrt{2}x_{12} \\ x_{11}^2 \\ \sqrt{2}x_{11}x_{12} \\ x_{12}^2 \end{bmatrix}^T \begin{bmatrix} 1 \\ \sqrt{2}x_{21} \\ \sqrt{2}x_{22} \\ x_{21}^2 \\ \sqrt{2}x_{21}x_{22} \\ x_{22}^2 \end{bmatrix}$$

$\phi(x)$ vectors of the second order polynomial non-linear transformation for the data is up above.

Logistic Regression

In this part, by using data provided us, we implement a logistic regression model by using stochastic gradient descent. The dataset has 2 features. When I checked the training labels, I saw labels were either -1 or 1. So, I decided to use tanh as the scaler function, instead of the σ. So, in this problem:

$$Z(x) = w^T x$$

$$h(x) = \tanh(Z)$$

$$h(x) \in [-1, 1]$$

and my loss function is (since I am using tanh, I choose this loss function. It gaves negative or zero into the logarithm if I use other popular logistic loss function):

$$E(w) = \frac{1}{N}\sum_{n=1}^{N} ln(1 + exp(-y_n w^T x_n))$$

For the training part, I started with random initialized weights w_1, w_2, b. For each epoch, i traversed in the shuffled dataset (I shuffled the indices of the dataset actually, not the dataset itself.) and in each iteration in the epoch, for each data point I randomly chose, I updated the weights. To take the partial derivative of the loss function and update the related weights (partial derivative of loss function according to w_1 to update w_1, etc.), I used chain rule.

$$\frac{\partial E(w)}{\partial h}\frac{\partial h}{\partial Z}\frac{\partial Z}{\partial w}$$

for w_1, gradient is

$$\nabla(w_1) = \frac{-y}{exp(h(x))} * (h^2 + 1) * x[0]$$

for w_2, gradient is

$$\nabla(w_2) = \frac{-y}{exp(h(x))} * (h^2 + 1) * x[1]$$

and for b, gradient is

$$\nabla(b) = \frac{-y}{exp(h(x))} * (h^2 + 1)$$

At the end of the each iteration, I updated the weights and the bias by the following formula:

$$w_{new} = w - \alpha * \nabla(w)$$

$$b_{new} = b - \alpha * \nabla(b)$$

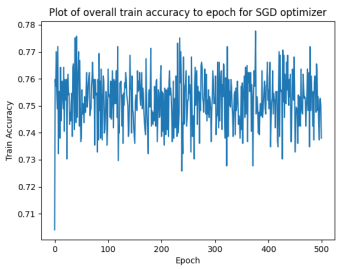
Here, α is the learning rate. I tried my model with different learning rates. I observed that, when I select learning rate high, the steps get bigger and more rapid changes can be observed from the accuracy graphs.

Results

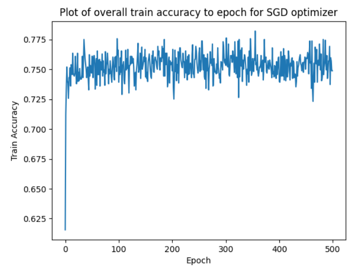
You can see the results of my model with epochs = [500,1000] and α=[0.01,0.001]. Overall (average) results of the model are:

Training Accuracy : 0.769
Epoch Training Loss: 0.081
Test Accuracy : 0.95
Epoch Test Loss: 0.165

500 Epoch

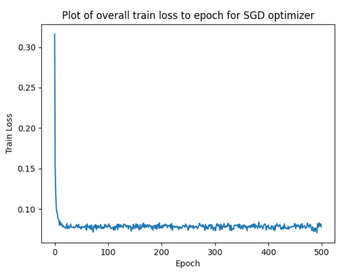


(a) lr=0.01

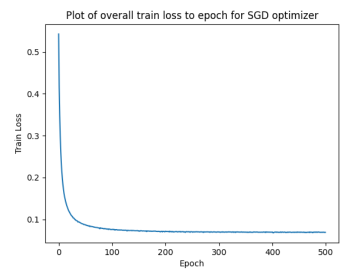


(b) lr=0.001

Figure 1: Training Accuracies with lr=0.01 and lr=0.001

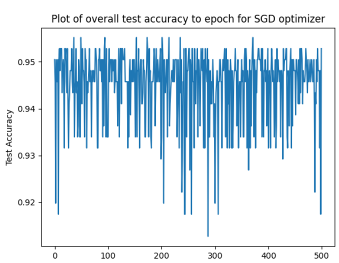


(a) lr=0.01

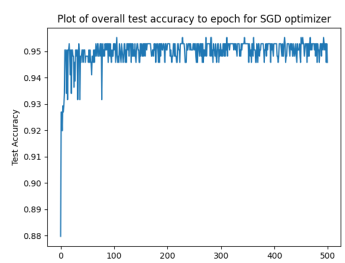


(b) lr=0.001

Figure 2: Training Losses with lr=0.01 and lr=0.001

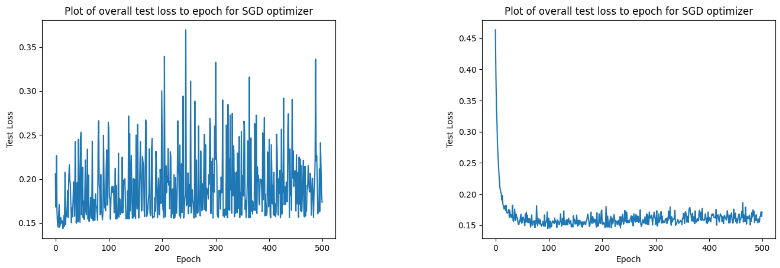


(a) lr=0.01



(b) lr=0.001

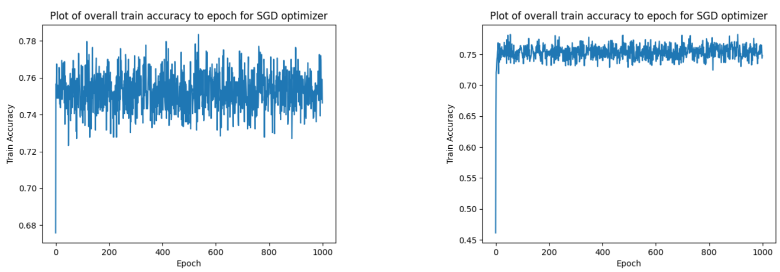
Figure 3: Testing Accuracies with lr=0.01 and lr=0.001



(a) lr=0.01 (b) lr=0.001

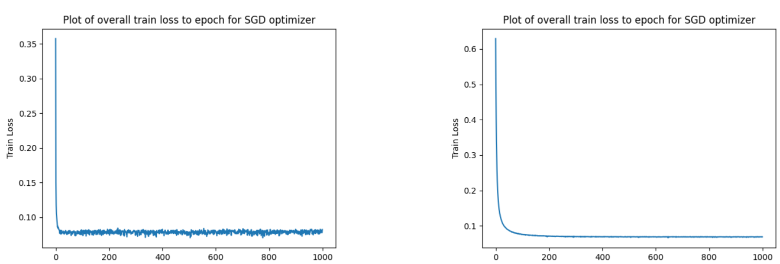
Figure 4: Testing Losses with lr=0.01 and lr=0.001

1000 Epoch



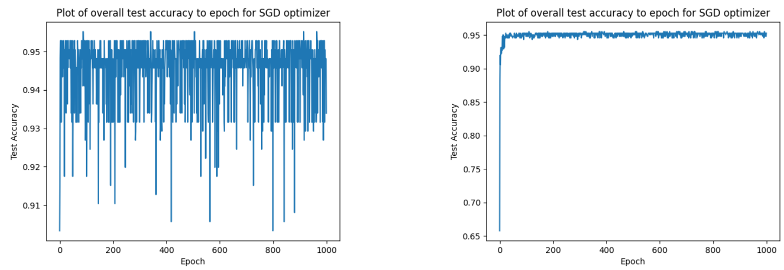
(a) lr=0.01 (b) lr=0.001

Figure 5: Training Accuracies with lr=0.01 and lr=0.001



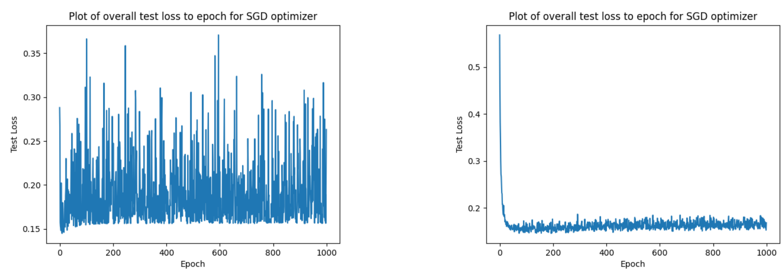
(a) lr=0.01 (b) lr=0.001

Figure 6: Training Losses with lr=0.01 and lr=0.001



(a) lr=0.01 (b) lr=0.001

Figure 7: Testing Accuracies with lr=0.01 and lr=0.001



(a) lr=0.01 (b) lr=0.001

Figure 8: Testing Losses with lr=0.01 and lr=0.001

References

[1] Domke, J., "Kernel Methods and SVMs", `https://people.cs.umass.edu/~domke/courses/sml2011/07kernels.pdf`.

[2] John Shawe-Taylor, S. S., "Kernel methods and support vector machines", `http://www.cst.ecnu.edu.cn/~slsun/pubs/KernelMethods.pdf`.

Lecture slides have been used during this study.