

CMPUT 466/566 Project Report

Project Title: Support Vector Machine and Neural Network
Implementation from Scratch

Team: Debraj Ray (csid: debraj1)
Mohan Sai Singamsetti (csid: singamse)

Introduction

Our objective for the course-project was to apply the concepts that we have learned in class to implement machine learning algorithms from scratch. That is, NO use of standard ML libraries like Scykit Learn, Pytorch, Tensorflow etc. After discussing with the instructor, we decided to implement Support Vector Machine (linear and nonlinear) and Neural Network using math libraries such as NumPy, and Scipy only. In the following paragraphs we will introduce these machine learning models and our motivation to implement them.

SVM or support vector machine is a binary classifier that is trained using supervised machine learning techniques. They are non-probabilistic models and are particularly useful because they have good performance even with a limited number of training samples. SVM works by fitting a hyperplane between two classes of data such that the distance to the nearest data points of each class (margin) is maximized. SVM is really interesting to us because it uses the idea of constrained convex optimization to solve the maximization problem. We believe it is an important concept to understand well and has wide-range of applications in the machine learning field. Another interesting idea of SVM is the Kernel Trick used to separate two non-linearly separable classes of data. The idea is that instead of trying to fit the hyperplane in the original feature space, we transform the data into a higher dimension and then fit the hyperplane. This results in nonlinear classification in the original feature space. However, we will soon see in the next section that the solution to the SVM problem only depends on the inner-product of the feature vectors (or pairwise similarity). Therefore, the “trick” is that, instead of actually transforming the features to a higher dimension (which can be computationally expensive), we compute the inner product in the higher dimension using a kernel function. We explain the kernels used for this project in more details in the next section. In the next sections we dive deeper into the linear and nonlinear SVM math and implementation. Furthermore, we apply the linear SVM to solve the [Titanic problem of Kaggle](#). This challenge aims to classify people in two groups - those who are likely to survive and those who are not based on passenger data (such as name, age, gender, socio-economic class, etc...).

Artificial neural networks are inspired from biological neural networks. They are created by stacking layers of neurons sequentially. The layers are called: input layer, hidden layer(s) and output layer. All the neurons of one layer are connected with every neuron in the subsequent layer. Each connection has a tunable weight and each neuron has a bias. The objective of the neural network is to reduce the loss on the training data. The weights and biases are optimized by gradient descent in each layer and the loss is back-propagated to the previous layers. There are also tunable hyperparameters such as number of neurons per layer, learning rate of the

optimization algorithm, number of epochs, etc. These hyperparameters can be decided based on design choice or through cross validation.

To prevent overfitting, we have also explored L2-regularization for Neural Network. It shrinks the weights to small values thus helping the model to generalize better. We explain the whole process of neural network math in the next section “Literature Review and Background Math” section. Then we apply the neural network on the Titanic problem and image classification datasets - MNIST and CIFAR-10. Finally we discuss the results.

Literature Review and Background Math

- SVM (linear)

The objective of SVM is to maximize the margins of the hyperplane separating the two classes. It can be written as shown below:

$$\begin{aligned} & \text{maximize} \quad M \\ & \text{minimize} \quad \frac{t^m (w^T x^m + b)}{|w|} \\ & w, b \quad m=1 \end{aligned}$$

Next, we use an additional constraint: $\text{minimize } t^m (w^T x^m + b) = 1$, to ensure we get a unique weight and bias in the solution. With this, we can convert the above optimization to the standard constrained convex optimization formulation of SVM as given below:

$$\begin{aligned} & \text{minimize} \quad \frac{1}{2} |w|^2 \\ & w, b \end{aligned}$$

$$\text{subject to } 1 - t^m (w^T x^m + b) \leq 0 \quad \text{where } m = 1, 2 \dots M$$

M is the total number of samples and m represents a specific sample. The above formulation is also called the primal problem. To solve the optimization we next construct the Lagrangian. Here we convert the hard constraints to soft penalties.

$$L(x, \alpha) = \frac{1}{2} |w|^2 + \sum_{m=1}^M \alpha^m [1 - t^m (w^T x^m + b)]$$

Now, our goal is: $\text{maximize}_{\alpha} \text{minimize}_{w,b} L(x, \alpha)$

We find the closed form solution for w and b.

$$\frac{\delta L}{\delta W} = 0$$

$$\Rightarrow w_{optimum} + \sum_{m=1}^M \alpha^m [-t^m(x^m)] = 0$$

$$\Rightarrow w_{optimum} = \sum_{m=1}^M \alpha^m t^m x^m$$

$$\frac{\delta L}{\delta b} = 0$$

$$\sum_{m=1}^M \alpha^m t^m = 0$$

Replacing these solutions in the new goal, we construct the dual problem

$$\begin{aligned} & \underset{\alpha}{\text{maximize}} \quad \sum_{m=1}^M \alpha^m - \frac{1}{2} \sum_{n=1}^M \sum_{m=1}^M \alpha^n \alpha^m t^n t^m [x^n]^T x^m \\ & \alpha \end{aligned}$$

$$\begin{aligned} & \text{subject to,} \quad \alpha^m \geq 0 \\ & \sum_{m=1}^M \alpha^m t^m = 0 \end{aligned}$$

Also, we know:

$$\begin{aligned} & t^m (w^T x^m + b) = l \\ \Rightarrow & t^m (w_{optimum}^T x^m + b_{optimum}) = l \end{aligned}$$

Multiplying both sides by t^m

$$\begin{aligned} & w_{optimum}^T x^m + b_{optimum} = t^m \\ \Rightarrow & b_{optimum} = t^m - w_{optimum}^T x^m \end{aligned}$$

For inference, we would use the following:

$$y = w_{optimum}^T x^m + b_{optimum}$$

Finally, as a regularization method to prevent overfitting, we allow α to be bounded within $[0, C]$ where C is a hyperparameter.

- SVM (nonlinear)

The above derivation shows that the Lagrangian dual only requires the inner-product of feature vectors. Let's assume we use a nonlinear transformation $\Phi(x)$ to move to a higher dimension. Then, our dual problem would become:

$$\begin{aligned} & \text{maximize} \quad \sum_{m=1}^M \alpha^m - \frac{1}{2} \sum_{n=1}^M \sum_{m=1}^M \alpha^n \alpha^m t^n t^m [\Phi(x^n)]^T \Phi(x^m) \\ & \alpha \end{aligned}$$

$$\begin{aligned} & \text{subject to,} \quad \alpha^m \geq 0 \\ & \quad \sum_{m=1}^M \alpha^m t^m = 0 \end{aligned}$$

We will not evaluate $w_{optimum}$ directly. Rather use it to do inference and calculate b on demand.

$$\begin{aligned} b_{optimum} &= t^m - w_{optimum}^T \Phi(x^m) \\ &= t^m - \sum_{n=1}^M \alpha^n t^n \Phi(x^n)^T \Phi(x^m) \end{aligned}$$

For inference, we would use the following:

$$\begin{aligned} y &= w_{optimum}^T \Phi(x^m) + b_{optimum} \\ &= \sum_{n=1}^M \alpha^n t^n \Phi(x^n)^T \Phi(x^m) + b_{optimum} \end{aligned}$$

In this project we use two different Kernel functions to compute the nonlinear inner product.

1. Polynomial Kernel

$K(x,z) = (1 + x^T z)^n$, where n is the degree of the polynomial (hyperparameter).

2. Gaussian Kernel / Radial Basis Function

$K(x,z) = \exp(-\gamma|x - z|^2)$, where γ can be considered a measure of variance of the gaussian and is a hyperparameter.

- Artificial Neural Network

Neural networks are known as function approximators which map between the input and output by using the training data. Neural network optimization is treated as a search based algorithm, which navigates the space of possible weights for model, which can help to make good predictions.

In Neural Networks, gradient descent based approach is used as optimization algorithm and weights are updated using backpropagation. The term gradient refers to the derivative of an error function w.r.t to the weight parameters in every layer.

The term error function is also known as loss function/objective function. The objective of a neural network is to minimize the loss function, which means finding the candidate solution which has the lowest loss value.

Forward Propagation:

Input value to the network is represented as X. All the learnable parameters in a neural network are known as weights(W) and bias(b). The cumulative value in the forward propagation is represented by z.

$$Z = X * W + b$$
$$A = f(Z)$$

Activation functions(f):

The activation function(f) decides whether a neuron should be activated or not, which will help to understand the importance of that particular neural activation in the network. Activation function also helps to learn non-linear representation in the network i.e. find a decision boundary which is not linearly separable. In neural networks we mainly use softmax, sigmoid and ReLu activation functions.

Softmax

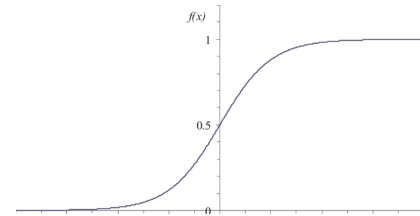
Softmax functions are mainly used as activation functions in the output layer of neural network models which gives probability prediction for multi-class classification problems.

$$a_i = \frac{e^{z_i}}{\sum_{k=1}^n e^{z_k}}$$

Sigmoid

The sigmoid is the most commonly used activation function and is monotonically increasing everywhere. This squashes the neural activation value in a range between 0 to 1.

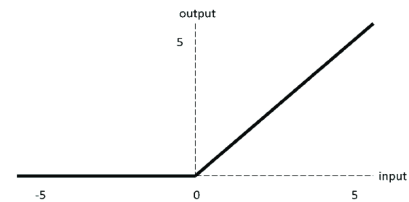
$$a = \frac{1}{1 + e^{-z}}$$



ReLU

This is also a non-linear activation function that has the advantage of removing the vanishing gradients problem faced by other activation functions. This activation function excludes the negative range of values.

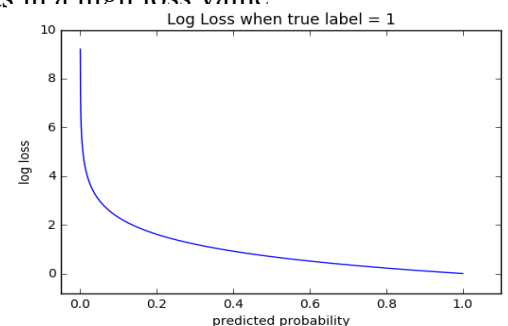
$$a = \max(0, z)$$



Loss Function(Cross entropy)

Cross-entropy is a commonly used loss function for classification problems. Cross-entropy loss measures the performance of a classification model whose output is a probability between 0 and 1. When the predicted probability is close to 1 then the cross-entropy loss is low. Whereas, if the predicted probability diverges from the actual label, then it results in a high loss value.

$$\text{Loss} = - \sum_{i=1}^n [y_i * \log(a_i)]$$



Backward Propagation

Once we have computed the loss of the network from the forward propagation, we compute the gradients for the weights and bias of the last layer using the formula given below. Here A_n is the output of the n^{th} layer. 'A' is also represented as 'H' in the coding python notebook.

$$\delta Z_n = (A_n - Y)$$

$$\delta W_n = \frac{1}{m} * dZ_n * A_{n-1}^T$$

$$\delta B_n = \frac{1}{m} * \text{sum}(dZ_n, 1)$$

We compute the gradients of the weights and bias for rest of the layers till the beginning of the network using the formulae below, $i = 1, 2, \dots, n-1$

$$\delta Z_{n-i} = W_{n-i+1}^T * dZ_{n-i+1} * df_{n-i}(Z_{n-i})$$

$$\delta W_{n-i} = \frac{1}{m} * dZ_{n-i} * A_{n-i-1}^T$$

$$\delta B_{n-i} = \frac{1}{m} * \text{sum}(dZ_{n-i}, 1) \dots (1)$$

Compute gradient with L2 regularization. L2 regularization works by adding an additional term to the loss function that penalizes large weight values. The gradients of the weights can be computed using the formulae below, whereas the gradients of the bias term remain the same as equation 1. Here λ is the term that regularizes the weights in the network.

$$\delta W_n = \frac{1}{m} * dZ_n * A_1^T + \frac{\lambda * W_n}{m}$$

Once we have computed the gradients of the weights and bias. We update the weights and bias using the gradient descent. Here, α is known as the learning rate.

$$W = W - \alpha. \delta W$$

$$B = B - \alpha. \delta B$$

Problem Formulation

- SVM (linear)

We generated three sets of two dimensional data representing training set, validation set and test set. The dataset is small and manually generated. It consists of 21 training samples, 8 validation samples and 15 test samples. Although the dataset is small, it is still sufficient to demonstrate our

correct implementation. The input to the model during training is the training set and the correct label of each training sample, plus a regularization hyperparameter C (discussed in the previous section). The hyperparameter is tuned by cross validation. The output from the training is the optimal weights and bias. These optimal weights and bias terms are used to run classification on the test set.

In addition, we deployed the linear SVM to solve the titanic problem. This challenge aims to classify people in two groups - those who are likely to survive and those who are not based on passenger data (such as name, age, gender, socio-economic class, etc...). The training set consists of 348 samples and the test set consists of 286 samples. We did not do hyperparameter tuning here. We reused the value of C that we got after training on the smaller dataset (explained in the previous paragraph).

- SVM (nonlinear)

The nonlinear SVM has been trained similar to the linear SVM. We generated three sets of two dimensional data representing training set, validation set and test set. However we required two such sets - one for training SVM with polynomial kernel and the other for training SVM with gaussian kernel. The dataset for the polynomial kernel SVM consists of 16 training samples, 16 validation samples and 12 test samples. The dataset for the gaussian kernel SVM consists of 19 training samples, 13 validation samples and 12 test samples. For the polynomial kernel we tuned the hyperparameter - degree of the polynomial - using cross validation. Similarly for the gaussian kernel we tuned the hyperparameter - γ which is a measure of the variance of the distribution - with cross validation.

- Artificial Neural Network

The Neural Network is evaluated on binary and Image classification tasks. For the binary classification task we have used the same titanic challenge from kaggle. For the image classification tasks we have chosen CIFAR-10 and MNIST dataset.

The input to the neural network for the titanic challenge contains 11 features and the output is binary indicating whether a passenger survived or not. We have one hidden layer with 10 neurons.

For the CIFAR-10 the input dimensions for the neural network is $32 \times 32 \times 3$ (i.e. 3072 input neurons) and the output layer has 10 neurons. This original dataset has a total of 50,000 training samples, 10,000 validation samples and 10,000 testing samples, but we have only sampled 10,000 images for training, 1,000 for validation and 1,000 for testing. This is because we found our implementation is not fast enough to process the large dataset in reasonable time. We have 2 hidden layers with 50 and 30 neurons respectively.

The MNIST dataset has 60,000 training samples and 10,000 test samples. However, we have limited our training set to 5000 because of our relatively slower implementation. The input to the neural network for MNIST is 28x28(784 neurons) and the output layer has 10 neurons. We have one hidden layer with 45 neurons.

Implementation Approach & Metrics

- SVM (linear)

The training step is executed by trying to maximize the Lagrangian dual. Here, we take the help of the library Scipy to run the maximization (by doing negative minimization). Internally, it calls our lagrangian function implementation. The two constraints of the convex optimization are applied using two methods: Scipy.optimize.Bounds and Scipy.optimize.LinearConstraint. The

former ensures $0 \leq \alpha^m \leq C$ and the later ensures $\sum_{m=1}^M \alpha^m t^m = 0$. After training is completed,

we use the optimal α to find w_{optimal} and b_{optimal} using the formulas explained in the previous section. We also estimate C using cross validation. Finally, we use w_{optimal} and b_{optimal} and the estimated C to measure accuracy on the test set. The measure of success is Accuracy which is total number of correct predictions / total number of test samples.

- SVM (linear) on the Kaggle Titanic challenge

The linear SVM was applied to solve the Titanic problem of Kaggle. We first did some data analysis and feature engineering, and then started training the SVM. Unfortunately, we found the linear SVM that we implemented was much slower than the Scykit Learn implementation. Due to this, the training was taking an unacceptable amount of time. So, we had to reduce the size of the training set to speed up. The test set was however not modified in any way.

- SVM (nonlinear)

The implementation of nonlinear SVM is very similar to the linear SVM, except we also need to implement the Kernel functions and the Kernel trick. We have implemented a polynomial kernel function and a gaussian (RBF) kernel function. As explained in the math, the kernel trick is implemented in the calculation of the Lagrangian in the dual, calculation of optimal bias term and inference calculation.

The polynomial kernel requires a hyperparameter n which represents the degree of the polynomial. The gaussian kernel requires a hyperparameter γ which is a measure of the variance of the distribution. These parameters are estimated using cross validation. Similar to linear SVM, we use Accuracy as the evaluation metric.

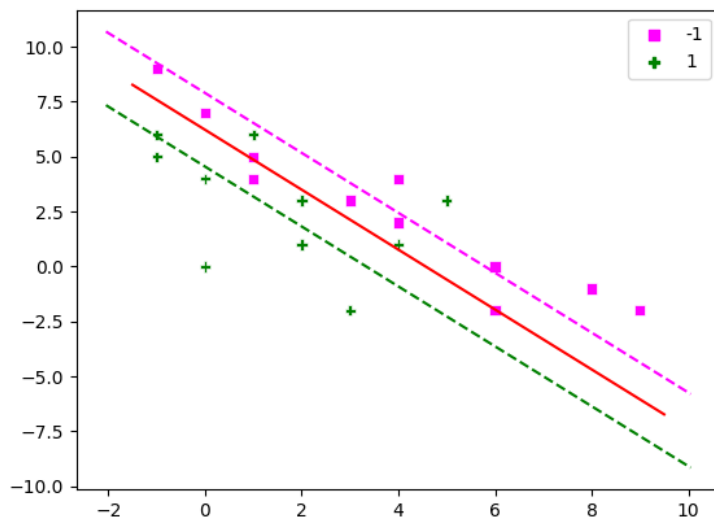
- Neural Network

For the image classification tasks - CIFAR-10 and MNIST - we have implemented data normalization. This converted the numerical data into a common scale between [0 - 1]. For the titanic problem, we have the same feature engineering as is done for the SVM linear model.

Then, we have implemented the activation functions - softmax, sigmoid, and ReLU. Also we have applied L2-regularization as a constraint which penalizes the model with heavy weights thus preventing overfitting. We have chosen L2-regularization coefficient λ as 0.01, based on the validation accuracy on the titanic task. We have reused this value for the CIFAR-10 and MNIST tasks as the training on classic image classification tasks take a lot of time . We initialize the weights and bias with random values. To find the optimal weight and bias values of the model we have used batch gradient descent as the optimization algorithm which updates the weight and bias values by computing the gradients using all the training samples in a batch. We have chosen a learning rate of 0.0075. The performance of the neural network is evaluated with the metric - Accuracy.

Results

- SVM (linear)



Clearly, the SVM algorithm is able to create a reasonable hyperplane with acceptable misclassifications on the training data. The validation accuracy peaks for $C=100$. Therefore we use the weights and biases that we got during training with $C=100$.

Validation Accuracy is 50.0 for C value 0.0001

Validation Accuracy is 62.5 for C value 0.001

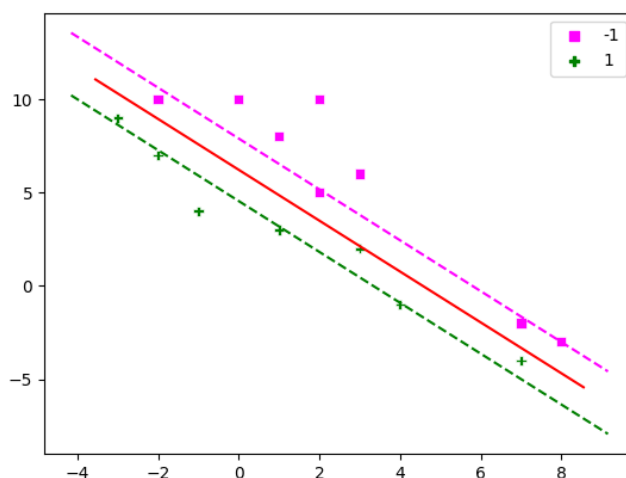
Validation Accuracy is 75.0 for C value 0.1

Validation Accuracy is 100.0 for C value 100

Validation Accuracy is 62.5 for C value 1000

Validation Accuracy is 37.5 for C value 10000

The test set was classified with 100% accuracy. Note, the test set contains unseen data and does not overlap the train / validation set. The following plot shows the hyperplane plotted on the test set.



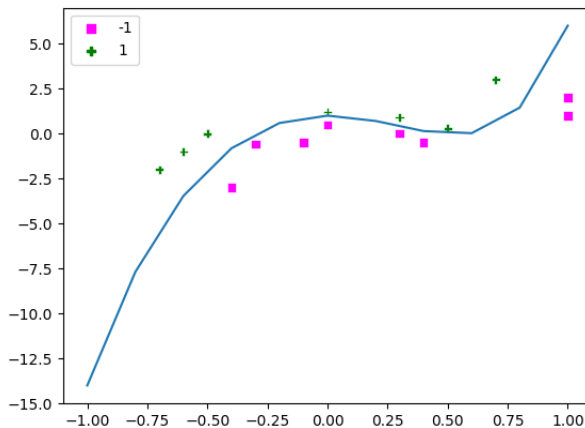
- SVM (linear) on the Kaggle Titanic challenge

Accuracy on the test set was found to be 80.7%. Perhaps the test accuracy would be higher if we could train the SVM on the entire training dataset. However, since training was running quite slowly we had to reduce the size of the training set to ensure training completion in about an hour.

- SVM (nonlinear)

Plot of training data used to train SVM with polynomial kernel. Note, the line plot is the true function which is a 5th degree polynomial: $y = 3x^5 + 4x^4 + 7x^3 - 9x^2 + 1$

However, the SVM has no knowledge of the true function. It is only aware of the training samples and their labels (pink squares and green pluses).



We use cross validation to determine the degree of the polynomial. As can be seen in the output shown below, the best validation accuracy was reported for degree 5.

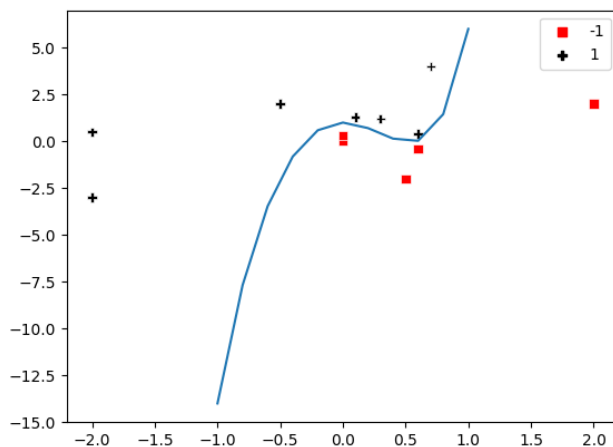
Accuracy on validation set of Polynomial SVM for degree 2 is 64.70588235294117

Accuracy on validation set of Polynomial SVM for degree 3 is 82.35294117647058

Accuracy on validation set of Polynomial SVM for degree 4 is 82.35294117647058

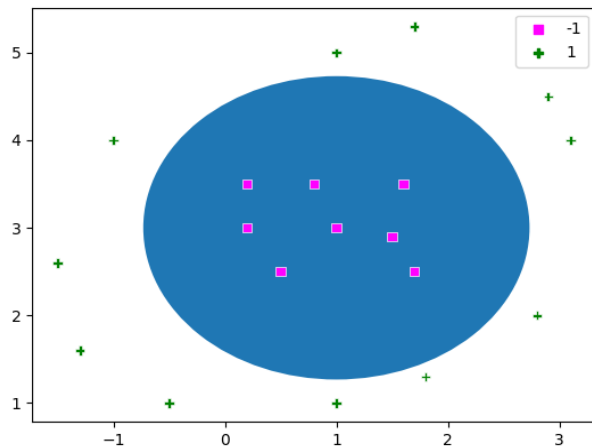
Accuracy on validation set of Polynomial SVM for degree 5 is 88.23529411764706

Given below is a plot of the test data points and the SVM was able to classify all the points correctly.



Next, we show the data used for training the SVM with the RBF kernel. The true decision boundary is a circle: $(x-1)^2 + (y-3)^2 = 3$

However, the SVM is unaware of the true decision boundary. It is only aware of the training samples and their labels (pink squares and green pluses).



By checking accuracy on the validation set we found the best value of $\gamma=0.01$

Accuracy on validation set of RBF SVM for gamma 0.001 is 46.15384615384615

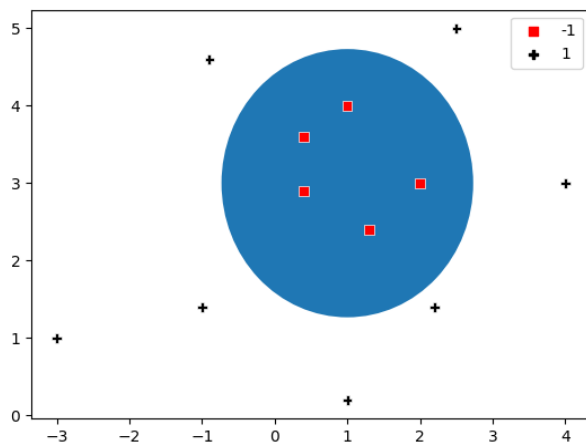
Accuracy on validation set of RBF SVM for gamma 0.01 is 92.3076923076923

Accuracy on validation set of RBF SVM for gamma 0.1 is 84.61538461538461

Accuracy on validation set of RBF SVM for gamma 1 is 46.15384615384615

Accuracy on validation set of RBF SVM for gamma 10 is 46.15384615384615

Given below is a plot of the test data points and the SVM was able to classify all the points correctly.



- Neural Network

We have plotted the model learning curve(loss vs epochs) of the neural networks with and without L2-regularizations.

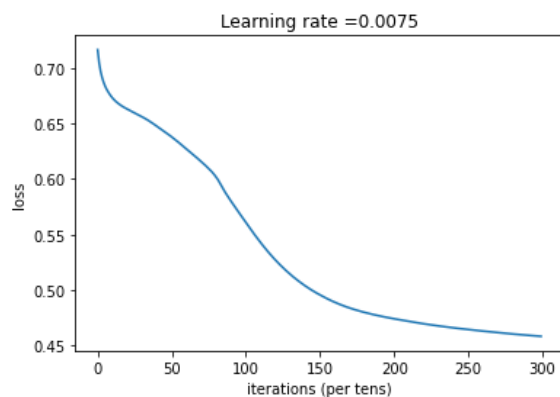
Titanic Task:

The training plot without L2-regularization:

The training loss after 3000 iterations: 0.4582

Model accuracy on train data: 0.8247

Model accuracy on test data: 0.8041



The training plot with L2-regularization:

The loss value after training 3000 iterations: 0.4582

Model accuracy on train data: 0.8247

Model accuracy on test data: 0.8177

λ value for best validation accuracy: 0.01

Validation results:

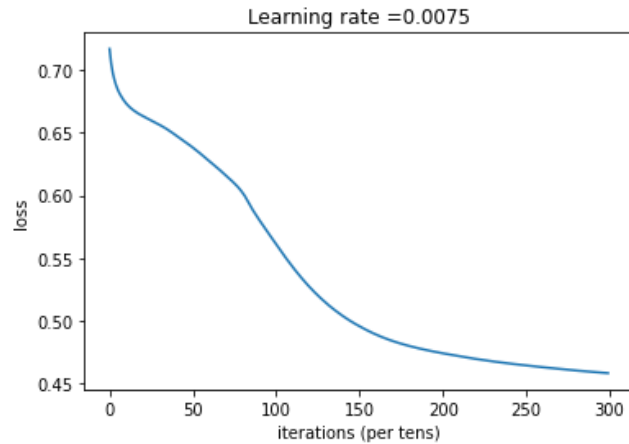
The validation performance for 0.01 with accuracy 0.74

The validation performance for 0.1 with accuracy 0.74

The validation performance for 1 with accuracy 0.74

The validation performance for 10 with accuracy 0.72

The validation performance for 100 with accuracy 0.7



We did not find any significant difference with or without L2 regularization in the Titanic task. We could achieve an accuracy of about 81% on the test set using our implementation of the Neural Network.

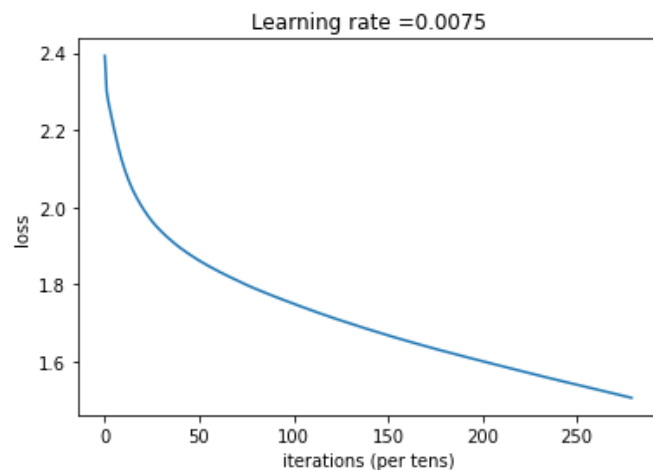
CIFAR10 Dataset:

The training plot without L2-regularization:

The loss value after 2800 iterations: 1.5068

The Model accuracy on train data: 0.4646

The Model accuracy on test data:0.4126



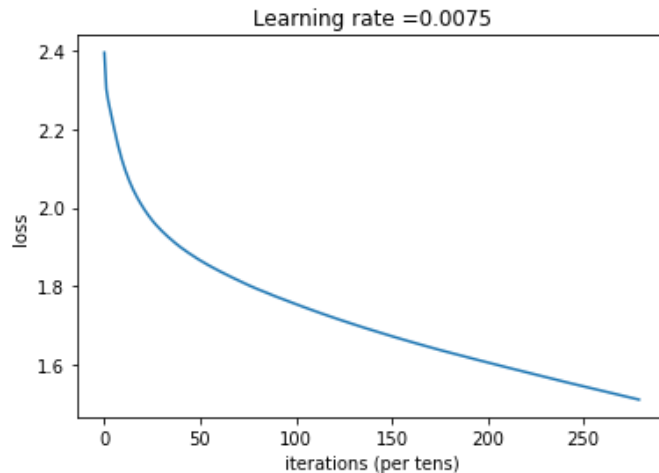
The model training with L2-regularization($\lambda = 0.01$).

The loss value after 2800 iterations: 1.5099

The Model accuracy on train data: 0.4639

The Model accuracy on validation set: 0.4120

The Model accuracy on test data:0.4118



For the CIFAR-10 dataset, we have applied early stopping of training when the validation accuracy does not improve in each epoch. For both cases, the model training was stopped at 2800 iterations. We believe the low accuracy in the CIFAR-10 dataset is perhaps because a simple neural network is not fit for this task. Perhaps we would need a convolutional neural network to do better.

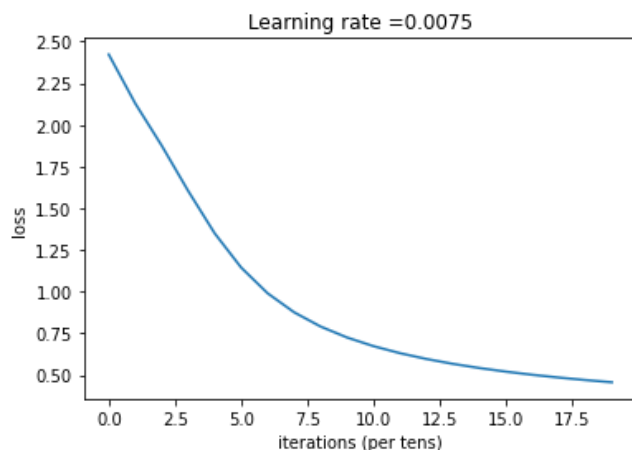
MNIST Dataset:

The training plot without L2-regularization:

The training loss after 2000 iterations: 0.4570

The Model accuracy on train data: 0.8774

The Model accuracy on test data:0.8674

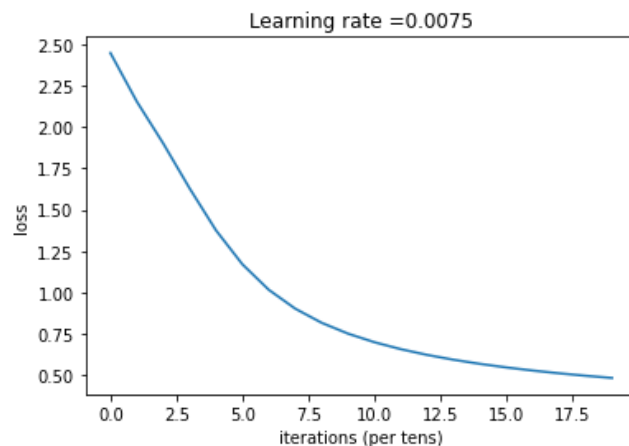


The model training with L2-regularization($\lambda = 0.01$).

The loss value after 2000 iterations: 0.4574

The Model accuracy on train data: 0.8774

The Model accuracy on test data: 0.8674



The neural networks did achieve good performance on MNIST dataset with an accuracy of approx 86% in both the cases with and without L2-regularization. Unfortunately, we could not find a better model with L2 regularization even for MNIST.

From the above tasks we are confident that our implementation of the Neural Network is correct. However, we could not fully understand why L2 regularization did not help in any of the tasks. We could not investigate further due to limited time.

Contributions

Debraj worked on the implementation of both linear and non-linear SVM.

Mohan Sai worked on the implementation of Neural Network.

Both have jointly prepared the report.

Resources



Datasets:

Titanic Challenge: <https://www.kaggle.com/competitions/titanic/data?select=test.csv>

CIFAR-10: <https://www.cs.toronto.edu/~kriz/cifar.html>


MNIST: <http://yann.lecun.com/exdb/mnist/>

For SVM we referred to the following:

1. Lecture Notes.
2.  Lili Mou Machine Learning Course - Class 19: Linear SVM
3.  Lili Mou Machine Learning Course - Class 20: Kernel SVM

For Neural Networks we referred to the following:

1. Lecture Notes

2. <https://en.wikipedia.org/wiki/Backpropagation>
3.  Lili Mou Machine Learning Course - Class 18: Neural Networks