**REPORT ON FROM SCRATCH IMPLEMENTATION OF LINEAR SUPPORT VECTOR MACHINES**

**Introduction:**

Most linear Supervised Classifying Algorithms have almost the same performance, what makes a difference is the data it’s been trained on, the features on which it is trained on and the regularization parameter.

Support Vector Machines (SVM) is one of the simplest and arguably one of the most elegant classification methods in Machine learning. They perform the classification by finding a hyper plane in n-D such that all points of one class are on one side of the hyperplane and all points of the other class are on the other side of the hyperplane. While linear perceptron and logistic regression tries to do the same, each one of them stops on finding the first solution and not the optimal one. SVM on the other hand, make use of support vectors, data points that lie close to the hyperplane, to find the maximum margin hyperplane.

Chart, scatter chart

Description automatically generated(Wikipedia, 2021)

**Problem, Solution and Algorithm**

Lets look at it in terms of our problem:

* We have a binary classification problem with a labelled data set (Xi,Yi), for fire detection.
* X is a vector with 9-features which can be plotted in a 9-D space.
* Each X has a corresponding Y which can be -1 or +1 denoting labelled class ‘no’ and ‘yes’ respectively.
* The goal is to find an optimal hyperplane for X such that:
  + A picture containing text

    Description automatically generated, where W is the normal vector to the plane.(See above fig) (Wikipedia, 2021)

**How do we classify?** Using the trainingdatawe find the optimal values of w and b which best classifies our data. These values are found by solving a **Convex Optimization Problem,** by minimizing a cost function. Once the algorithm is trained the w and b values are fixed, we classify our value X in the test set by using:

A picture containing text

Description automatically generated

**Hard Margin:**

Applicable when the data is linearly separable

Here we select two parallel hyperplanes such that the margin between the two classes is maximum.

Text

Description automatically generated with medium confidencedenotes the first hyperplane. All points with label 1 should be on or above this plane.

Text

Description automatically generated with medium confidence denotes the second hyperplane. All points with label -1 should be on or below this plane.

Our solution hyperplane lies halfway between these.  represents the distance between these two hyperplanes as indicated in the figure. So, inorder to maximize the distance, we need to minimize . This distance can be computed by distance from point to plane equation.

In addition, we need to add a constraint to ensure that data points don’t get misclassified.



A picture containing diagram

Description automatically generated. Each of these constraint ensure that the pointes labelled -1 and 1 lie on the correct side of the margin.

Combining all of this together, we get:

 (Wikipedia, 2021)

**The Loss Function**

**Soft Margin:**

All of this was assuming your data is linearly seperable. What if it’s not like it’s the case with out data. This is where the hinge loss function becomes relevant.

**Hinge Loss**= Text, letter

Description automatically generated

What does this mean? . This function is 0 when the point lies on the correct side of the margin. So, when ever a point is miss classified, the value of the loss function will be greater than 0 by a factor of the magnitude of missclassification or geometrically, how far the point is from the correct hyperplane.

In essence, the final objective function can be written as:

Diagram, schematic

Description automatically generated

**Regularization parameter C:**

Note the ‘C’ parameter in the above equation. This decides the trade-off between increasing margin size and ensuring that the points lie on the right side of the hyperplane. This means that when C is small enough, the SVM will be transformed into a hard margin case, assuming linear separation is possible. When C is larger, the importance will be on minimizing the hinge loss thus allowing a few misclassifications. So, it’s important to know if your class is linearly separable and tweak the parameter accordingly.

In our problem, just looking at a few features with high co-relation, it’s easy to see that there is a degree of linear separability but it’s not completely so. So, I have tried many values of C and fixed on 2 which allows a few misclassifications to allow the outlier but not overfit the data.

The Gradient of the Cost Function

Now that we have established the optimization function, what we need is to figure out how to optimize it. The cost function tells us how good or bad our model is doing. Considering, it’s a convex optimization problem(as shown in below fig), we can minimize it by taking the gradient.

Diagram

Description automatically generated

**Gradient Descent:**

The algorithm for Gradient Descent used in the implementation is :

1. Find the derivative of the cost function. Below shows the objective and it’s derivative

Text, letter

Description automatically generated

1. Update the weights in the opposite direction of the gradient by the learning rate. (We go in the opposite direction because gradient shows how fast and in what direction the function can increase.)

Diagram

Description automatically generated (M, 2019)

1. Repeat the steps until the loss is 0 or close to it or any other criteria is met (in this case, number of epochs).

[Design Decision for Gradient Descent]

Normally gradient descent is calculated using all the train data. However, this can be difficult to implement and computationally complex. Stochastic Gradient Descent or SGD is a good alternative to this and is implemented in the algorithm. It uses small step size and updates the weight after examining each of the data points. In this case we randomize the order of the training set so that with each run the order of update is different giving different and possibly better solution. It is suitable for the comprehensive randomized testing implemented for our algorithm.

Other potential method of implementation includes the Mini-batch Gradient Descent. Here, instead of going through each of the data points (SGD) or all of the Data point (Traditional GD), we apply the GD algorithm on batches by splitting the training data into mini-batches.

**Design Decisions:**

The following are the design decisions considered:

**Classifier:** Unlike linear perceptron or logistic regression, SVM tries to find the best margin that differentiates the classes and is less likely to overfit the data. On a quick analysis, it clear that the degree of linear separability of the data set is quite high, so a linear classification algorithm is more likely to perform than Decision Tree or Random Forest.

**Data normalization:** Both StandardScaler, which uses standard normal distribution, and min-max scaler was considered for data normalization. StandardScaler gave better performance overall and was fixed. It takes the mean to be zero and scales the data according to the variance.

**Weights and biases:** Weights are initialized to zero and biases are set to 1 initially. It was initialized to random values between -1 and 1 as well. There was no noticeable variation in the output and the initial value of 0 was retained to maintain code clarity.

**Gradient Descent:** Stochastic Gradient Descent was taken.The details of the same are given above under Gradient Descent.

**Loss function:** The Hinge Loss Function instead of the hard margin is used. The details are explained above.

**Learning Rate:** This decides how to update the weights based on the derivative of the loss function. The value is kept constant at the rate .0001. Since the loss values are converging around 300 epoch, this decision seems to be working reasonably. Optimized learning rate that modifies the rate to take bigger steps initially and as we move towards the minimum, take smaller steps might be more ideal.

**Regularization Parameter:** The value is kept at 2 and was varied from 1 to 1000. The best results were obtained with the final value 2.

**Epoch:** Even though the loss value plateaus at about 1.2 after 300 epochs, the value is kept at 1000 to visualize this. This also gives as a boost in performance even when compared to SGD Classifier implementation from SK Learn.

**Testing and Comparison with SK Learn Library:**

Steps taken for Testing and Evaluation:

1. Data is split into train and test set in the ratio 2:1 ten times using CreateTrainAndTest(i,X,Y) function. Here i denotes the random state.
2. With each of these combinations of test and train set, both the custom implementation and the sklearn version of the SVM classifier is run.
3. The accuracy, Precision, Recall, F1 score and AUC are noted at each run and overall for the 10 runs.

This report will go over the summary of the test results. We’ll pick test run 5 and the average of all 10 runs.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Chart

Description automatically generated Chart

Description automatically generated

Custom Implementation Sk Learn Implementation

We can see the the results are almost identical in both the cases. Train data and test data have almost the same level of f1 score indicating that overfitting is contained.

**Comparison of loss function value change over 4 runs.**

Chart

Description automatically generated Chart

Description automatically generated Chart

Description automatically generated

Once again the change is almost identical indicating that the model is stable and is able to classify the data well.

**Comparison of the Average Data over 10 runs**

Text

Description automatically generated Chart, treemap chart

Description automatically generated

Chart, treemap chart

Description automatically generated Chart

Description automatically generated Chart

Description automatically generated

***Conclusion***: The implementation of Linear SVM and the SK Learn SGD Classifier with hinge loss  
 are producing identical performance with wrt Accuracy and F1 score. The custom classifier is infact performing slightly better. With slight modification, we can generalize the implementation to support other classifications as well. Improvements can be made in the future by using an optimized learning rate and batch gradient descent.

# Bibliography

Lau, B. (2019). Andrew Ng’s Machine Learning Course in Python (Support Vector Machines). Retrieved from https://towardsdatascience.com/andrew-ngs-machine-learning-course-in-python-support-vector-machines-435fc34b7bf9

M, R. (2019, Sep). The Ascent of Gradient Descent. *clairvoyantsoft*. Retrieved from https://blog.clairvoyantsoft.com/the-ascent-of-gradient-descent-23356390836f

Ng, A. (2012). CSC 299. Retrieved from https://sgfin.github.io/files/notes/CS229\_Lecture\_Notes.pdf

NG, A. (n.d.). CSC-299. Retrieved from https://see.stanford.edu/materials/aimlcs229/cs229-notes3.pdf

Wikipedia. (2021, November 3). *Support-vector machine*. Retrieved from In Wikipedia, The Free Encyclopedia.

**Appendices:**

Find the code and implementation below: