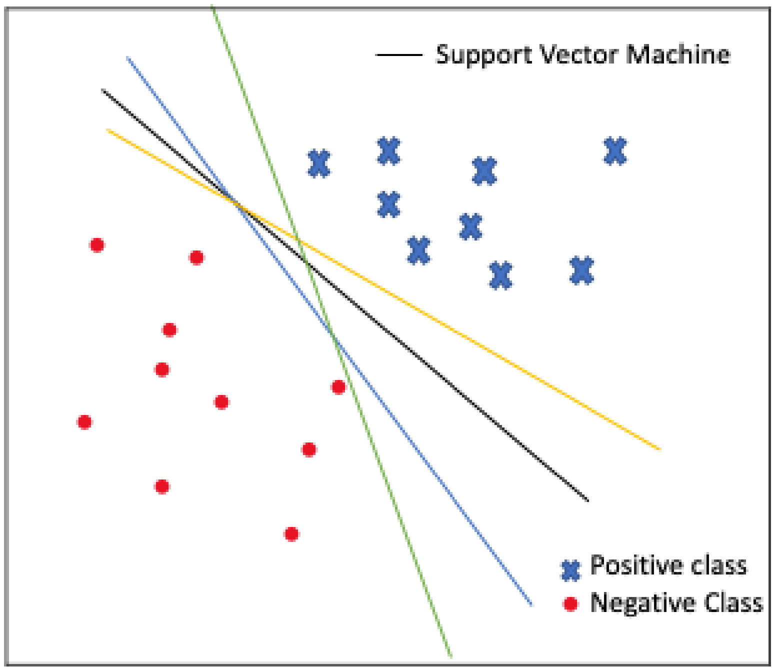
## 1 Introduction

Given a linearly separable binary class dataset (Figure 1), there are several possible straight lines that can correctly classify each datapoint. However, there is only one straight line (solid black), that not only classifies each datapoint correctly, but also with a high degree of confidence. Intuitively, it is the line which separates the datapoints with the widest possible margin between the two classes. This is exactly the decision rule of **Support Vector Machines** (SVM) and therefore, they are also known as **Maximum Margin Classifiers**. In the following sections, we will formalize this decision rule mathematically.

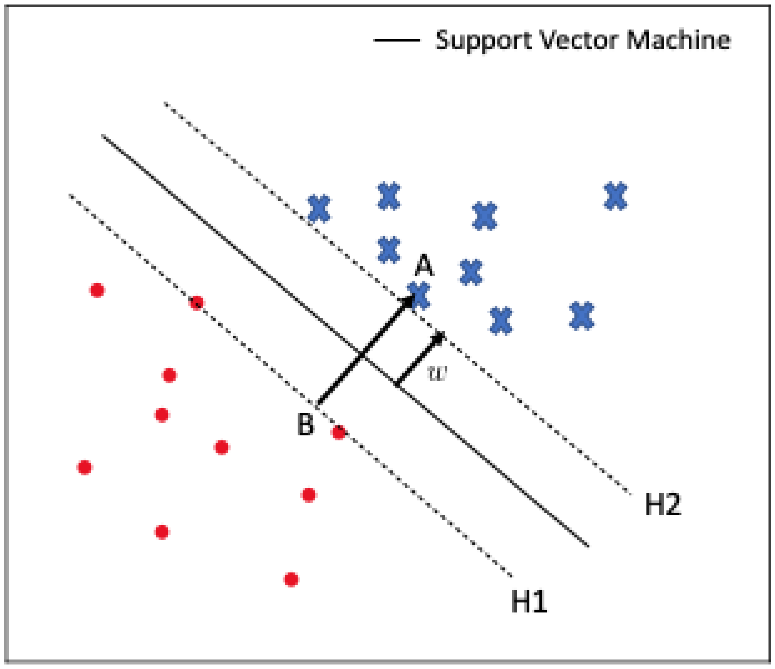
Figure 1: Decision boundary of linear SVM binary classifier



## 1.1 Hard margin classifiers

Figure 2 shows a binary class dataset $(x_i, y_i)$ *where* $y_i \epsilon \{+1, -1\}$. The decision boundary (in solid black) that we aim to formalize, is spanned with dotted black margins $H1$ and $H2$.

Figure 2: Decision boundary of linear SVM binary classifier



Let us define a vector $w$ that is constrained to be perpendicular to the decision boundary. To determine the class of any unknown sample $u$, all we need to do is project it on $w$. If the dot product is greater than, let's say a threshold $c$; it lies on the right side of the decision boundary and will be classified as a positive sample. Mathematically, $w \cdot u \geq c$
Or without loss of generality,

$$w \cdot u + b \geq 0, where\ c = -b,\ \ then\ u\ is\ positive,\ else\ negative. \tag{1}$$

To apply this decision rule, we need to work out the components of the *weights* vector $w$ and *bias* term $b$. Let's say we add a constraint such that for all positive samples $x_+$, $w \cdot x_+ + b >= 1$ and for all negative examples $x_-$, $w \cdot x_- + b <= -1$. The constraints for positive and negative samples can be merged into a single equation as follows,

$$y_i(w \cdot x) + b - 1 \geq 0, \textit{where } y_i \epsilon \ \{+1, -1\} \tag{2}$$

Also, for all samples lying on the margins $H1$ and $H2$ or anywhere between them, we add a constraint,

$$y_i(w \cdot x) + b - 1 = 0 \tag{3}$$

Our goal is to find the widest margin or maximize the distance between $H1$ and $H2$, with honouring the constraints defined in equations 2 and 3. Consider a known sample $A$ lying on $H2$. As the *weights* vector $w$ is perpendicular to margins $H1$ and $H2$ and from constraint in equation 3, the distance of point A to $H1$, represented by line segment $AB$ is,

$$(A - B) \cdot \frac{w}{\|w\|} = \frac{1 - b}{\|w\|} + \frac{1 + b}{\|w\|} = \frac{2}{\|w\|} \tag{4}$$

We seek to maximize $\frac{2}{\|w\|}$ or,

$$\begin{gathered} \min_{w,b} \frac{1}{2}\|w\|^2 \\ s.t.\ y_i(w.x_i + b) \geq 1,\ i = 1, ..n \end{gathered} \tag{5}$$

Note, squaring $\|w\|$ is just so as to represent it in a quadratic form. It does not change the optimal solution. The optimization problem in equation 5 is the 'primal' objective function of a linear SVM classifier. It is a convex optimization problem with linear constraints and can be solved efficiently with quadratic programming code. Although not absolutely required for linearly separable data, the computational complexity of the optimization problem in equation 5 can be made easier by solving its Lagrangian dual form. As it is a convex optimization problem with constraints, the duality gap is 0 and $p^* = d^*$, where $p^*$ and $d^*$ are optimal solutions of primal and dual forms respectively; and the dual problem can be solved in lieu of the primal. The Lagrangian for our optimization problem is

$$\begin{aligned} L(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n}\alpha_i[y_i(w.x_i + b) - 1] \\ \alpha_i \geq 0, i = 1, ...n \end{aligned} \tag{6}$$

where $\alpha_i$ are lagrange multipliers. To find the dual form of the problem, we need to set the partial derivatives of $L$ with respect to $w$ and $b$ to 0.

$$\begin{aligned} \nabla_w L(w,b,\alpha) = w - \sum_{i=1}^{n}\alpha_i y_i x_i = 0 \\ w = \sum_{i=1}^{n}\alpha_i y_i x_i \end{aligned} \tag{7}$$

and,

$$\nabla_b L(w,b,\alpha) = \sum_{i=1}^{n}\alpha_i y_i = 0 \tag{8}$$

Substituting for equations 7 and 8 in equation 6, and simplifying,

$$L(w,b,\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j x_i \cdot x_j \tag{9}$$

The dual optimization problem can thus be described as,

$$\begin{aligned} \max_{\alpha} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j x_i \cdot x_j \\ s.t.\ \alpha_i \geq 0, i = 1, ..n, \\ \sum_{i=1}^{n}\alpha_i y_i = 0 \end{aligned} \tag{10}$$

It can be shown that $\alpha^*$ (optimal $\alpha$) satisfies the KKT (Karush-Kuhn-Tucker) dual complementarity condition. This implies that, if $\alpha_i$ for a training instance $x_i$ is $> 0$, then the constraint function $y_i(w.x_i + b) = 0$ is 'active' or it holds the equality rather than the inequality condition. It is worthwhile noting that the optimal parameters $\alpha_i$ obtained by solving the dual problem can be used to reconstruct the optimal $w$ and $b$(equation 7). The optimal $w$ is dependent only on those few training instances where $\alpha_i > 0$. These training instances are the datapoints that lie on the margins and are known as 'support vectors'. All the other training instances that lie farther from the margins, have no bearing on the decision function. Predictions for unknown instances can now be made by substituting the reconstructed $w$ and $b$ parameters in equation 1.

The **Maximum Margin Classifier** obtained via solving either the primal or dual forms as represented in equations 5 and 10, can be efficiently computed for perfectly separable datasets. However, they inadvertently fail to converge when applied to real world datasets; which most likely contain one or more outliers. Therefore, it is also known as a **Hard Margin Classifier** as it does not permit even a single misclassification.

## 1.2 Soft margin classifiers

To make the primal form work for a non perfect separation of classes, the optimization problem can reformulated as;

$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ &s.t.\, y_i(w.x_i) + b >= 1 - \xi_i, \\ &\xi_i >= 0,\, \forall i = 1, ..n \end{aligned} \tag{11}$$

With the introduction of the slack variable $\xi$ for each training instance, the classifier is now permitted to make a few mistakes, subject to an increase in penalty by $C\xi_i$. The regularizer term $C$ controls the relative weighting of the twin goals of minimizing $\|w\|$ while also correctly classifying as many training instances as possible. Equation 11 represents the primal formulation of the so called **Soft Margin Classifier** as opposed to its stricter counterpart in equation 5. The dual formulation is very similar to that in equation 10; but with an additional constraint accounting for the regularization parameter $C$.

$$\begin{aligned} \max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j x_i \cdot x_j \\ s.t.\ \alpha_i \geq 0, i = 1, ..n, \\ \sum_{i=1}^{n} \alpha_i y_i = 0 \\ 0 \geq \alpha_i \geq C\ \forall i \end{aligned} \tag{12}$$

## 2 Implementation of Soft margin classifier

In this section we will understand how to estimate **generalization error**. We will then implement a linear SVM Soft Margin Classifier on a binary class dataset using Scikit-learn. Under the hood, Scikit-learn implements libsvm tools, which in turn solves the dual problem of the Soft Margin Classifier. The regularization parameter $C$ will be optimized by K-fold cross validation technique (explained in section 2.1). We will also solve primal and dual forms of SVM using CVXOPT, which is a Python package for convex optimization, and compare with Scikit-learn implementation.

### 2.1 Generalization error and Cross Validation

The goal of supervised machine learning is to decipher meaningful patterns in data (if any), so as to be able to make accurate predictions. The term *generalization* refers to the model's performance on unseen data. A model that has learned the underlying pattern in data without *memorizing* the noise, will generalize well on test data. A situation of high generalization error i.e. a model performing well on train data, but not on unseen data is called **overfitting**. When evaluating different settings for estimators, such as the regularization parameter $C$ for an SVM, there is a risk of overfitting on the test set. A very popular solution to this problem is a procedure called **$k$-fold Cross Validation**. A model is trained using $k-1$ folds or subsets of the training data. The resulting model is then validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy). The performance measure reported by $k$-fold Cross Validation is the the average of the values computed in the loop. The model with optimized parameters can finally be evaluated on unseen test data.

## 2.2 Scikit-learn and CVXOPT optimization

Briefly, the train and test data were read as Pandas dataframes using *read_csv* function. no missing entries were found in the dataset.The 0 values representing *negative* class were were replaced with $-1.0$. SVM classifier with linear kernel was employed with 'GridSearchCV' function of Scikit-learn to optimize regularization parameter $C$ with 5-fold Cross Validation. The optimal value of $C$ was found to be 0.001 with mean classification accuracy of 97.3%. The model with optimal $C$ was then fit on entire train set and used for predictions on test data. The parameters of the trained model, i.e. *weights* and *bias*, and the array of *support vectors* were stored as separate variables. The CVXOPT Quadratic program framework requires optimization functions to be of the form as shown in Figure 3A. The primal and dual formulations were accordingly manipulated. The variables vectors for primal and dual form are shown in Figures 3B, 3C respectively.

Figure 3: CVXOPT implementation of Primal and Dual

$$\begin{aligned} \min_{x} \quad & \tfrac{1}{2}x^\top P x + q^\top x \\ \text{subject to} \quad & Gx \preceq h \\ & Ax = b \end{aligned}$$

**A. CVXOPT format**

$$\begin{bmatrix} w_1 \\ \vdots \\ w_m \\ b \\ \xi_1 \\ \vdots \\ \xi_n \end{bmatrix}_{(m+n+1)\times 1}$$

**B. Primal variables**

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix}_{n\times 1}$$

**C. Dual variables**

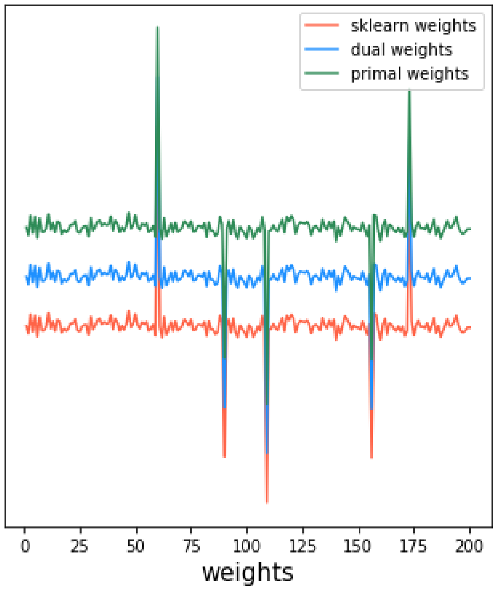
*$m$ - # of dimensions*

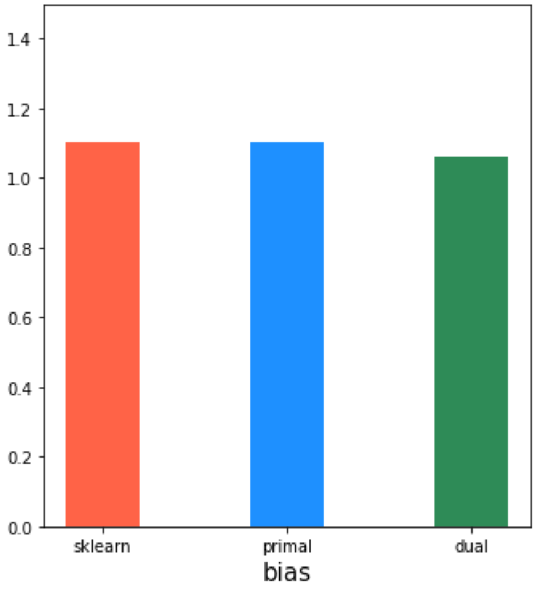
*$n$ - # of training examples*

Classification accuracy obtained on train and test sets with all three implementations are presented in Table 1, while Figure 4 shows the comparison of weights and bias terms. As expected, the classification accuracies and the parameters obtained by all three implementations are very similar.

| Software | Train accuracy(%) | Test Accuracy(%) |
|---|---|---|
| Scikit-learn | 97.6 | 97.4 |
| CVXOPT Primal | 97.6 | 97.4 |
| CXOPT Dual | 97.7 | 97.4 |

Figure 4: Comparisons of weights and bias parameters





## References

[1] Corrinna Cortes, Vladimir Vapnik. *Support-Vector Networks.* Machine learning, 20:273-297, 1995.

[2] *CVXOPT - Convex optimization*, https://cvxopt.org