American University of Armenia

College of Science and Engineering

CS 213, Optimization

Classification Algorithms

Team:
Sona Bezirganyan
Hripsime Ghazaryan
Lusine Davtyan

Spring, 2021

Abstract

The aim of the project is to discuss the problem of Classification and one of the approaches to solve it. Since the project is done in the scope of the Optimization course, we have tried to go with an approach that is more connected with Optimization problems. Thus, our focus will be the Support Vector Machines. The project will give the theoretical description of the problem and the algorithm. Besides, it will include the implementation of the algorithm in Python and compare the results with that of scikit-learn library. [Ped+11]

Contents

1 Introduction **3**
1.1 The Structure of the Project Paper 3

2 Theoretical Background **4**
2.1 Machine Learning 4
2.2 Problem Setting and Description 4
2.3 Support Vector Classifiers 4

3 Experiments **9**
3.1 Dataset 9
3.2 Comparison 9

4 Conclusion **11**

5 Appendix **12**
5.1 Algorithm Implementation in Python 12

1 Introduction

1.1 The Structure of the Project Paper

The project will start by giving a theoretical background to the problem. It will discuss the motivation of solving this problem and the way it is used in Machine Learning to do predictions. After having the big picture, it will zoom in to the mathematical formulation of the problem and discuss one of the approaches to solve it. Finally, it will include an implementation of the solution in Python. We will compare the results of our implementation and that of the scikit-learn library on the same dataset and make conclusions.

2 Theoretical Background

2.1 Machine Learning

Machine learning is programming computers to optimize a performance criterion using example data or past experience. [**alpaydin2014introduction**] There are three main categories of Machine Learning: supervised, unsupervised and reinforcement learning.
In case of supervised learning, we need to train the model to map from inputs x to outputs y given labeled observations. [Mur12] Each observation is a vector of features and each label is the "correct answer" for the given observation. Our aim is to train the model using current data in a way that given a new observation, it will be able to predict its label. In case the labels are real-valued scalars, the problem is considered a Regression problem, whereas when the labels are categorical or nominal, we have a Classification problem. From now on, our focus will be on Classification problem. This problem can further be divided into binary and multi-class classification problem. The primal one is the case when we have just 2 labels while for the latter the number of labels is greater than 2. The way we can generalize this problem into multi-class classification problem depends on the algorithm solving the binary classification problem. This stage of the problem is not tightly connected with Optimization problem, hence, our focus will be on binary classification problem. For convenience, we will assume that the labels are -1 and 1.

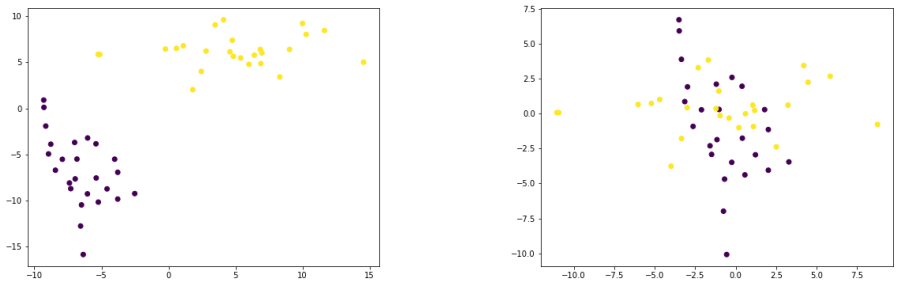
2.2 Problem Setting and Description

The problem is to do binary classification given a dataset. As we assume our observations to be labeled, we know which part of the training set has label 1 and which part has label -1. Thus, one of the approaches is to bring the problem to constructing a function which will be positive for the observations with label 1 and negative for the observations with label -1. As a result, when having a new observation, we will predict the label 1 if the function is positive and label -1 if the function is negative.

2.3 Support Vector Classifiers

First, let us note that we can have two types of settings; the case when the dataset is linearly separable and when it is not. You can see the illustration of the linearly seperable cases in Figures 1 and 2 respectively.
We assume that our datapoints are linearly separable. In this case, there exists a hyperplane dividing our dataset into two portions; datapoints with label 1 and datapoints with label -1. Thus, we need to find that hyperplane to classify the observations correctly. We can notice that there are infinitely many hyperplanes separating the dataset but we have to find the one which will allow us to do a better prediction. In other words, we need to find a hyperplane which will maximize the distance between two set of observations with different classes, so that we will predict the label of the unknown observation with higher confidence. To make it clearer, let us define the concept of margin. Assume we have a hyperplane separating our dataset into 2 classes. We move that hyperplane parallelly to the left and to the right until we hit the first datapoints. At that moment we stop and define those first datapoints to be Supporting Vectors and we define the 2 hyperplanes passing through those points to be Supporting Hyperplanes. We want to maximize the distance between the two Supporting Hyperplanes. Here, it is important to note that there are infinitely many parallel hyperplanes maximizing this distance. So we will agree to consider as Seperating Hyperplane the one that has equal distances from both of the Supporting Hyperplanes. Finally, we define the

Margin to be the distance between the Seperating Hyperplane and any of the Supporting Hyperplanes. In the linearly separable case, we assume there are no datapoints between the Supporting Hyperplanes.



(a) Linearly Separable Case

(b) Linearly Nonseparable Case

Figure 1: Linear separability

Now, after getting the intuition, let us find a formula for the margin. So if we assume that the Seperating Hyperplane is given by the following equation

$$\mathbf{w}^T \cdot \mathbf{x} + b = 0,$$

then the Supporting Hyperplanes will be represented by

$$\mathbf{w}^T \cdot \mathbf{x} + b = \pm c,$$

where c is a constant. Note that we can assume $c = 1$, because we can always get the same hyperplane by multiplying its coefficients with the same nonzero constant. As a result, if the observation has label 1, we assume that the datapoint is on or above the Supporting Hyperplane having positive label Support vector on it. Hence, we will have

$$\mathbf{w}^T \cdot \mathbf{x}_i + b \geq 1$$

for the observation $\mathbf{x}_i$ with label 1. In the same way, the observation with label -1 will be on or below the Supporting Hyperplane having negative label Support vector on it, and we will have

$$\mathbf{w}^T \cdot \mathbf{x}_i + b \leq -1$$

for the observation $\mathbf{x}_i$ with label -1. Now we are taking two Supporting Vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ with positive and negative labels respectively. By definition, the Margin M of a separating hyperplane is given by

$$M = \max_{\mathbf{w},b:y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 0} \min_{i\in[m]} \frac{|\mathbf{w}\cdot\mathbf{x}_i + b|}{\|\mathbf{w}\|},$$

where m is the size of the sample. Now, since we assume to have the linearly separable case, the pair $(\mathbf{w}, b)$, $y_i(\mathbf{w} \cdot \mathbf{x}_i + b)$ needs to be non-negative for all $i \in [m]$, so we will have

$$M = \max_{\mathbf{w},b} \min_{i \in [m]} \frac{y_i(\mathbf{w} \cdot \mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

Recall that we can safely multiply this expression by a scalar number, so we can assume that $\min_{i \in [m]} y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$, leading to

$$M = \max_{\mathbf{w},b:\min_{i \in [m]} y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1} \frac{1}{\|\mathbf{w}\|} = \max_{\mathbf{w},b:\forall i \in [m], y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1} \frac{1}{\|\mathbf{w}\|} \tag{1}$$

The second part of the above equation comes from the fact that if $(\mathbf{w}, b)$ is a maximizer, then $\min_{i \in [m]} y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$.
Finally, note that maximizing $\frac{1}{\|\mathbf{w}\|}$ is the same as minimizing $\frac{1}{2} \|\mathbf{w}\|^2$, and thus, we arrive to the following convex optimization problem:

$$\begin{aligned} \text{minimize} \quad & \frac{\|\mathbf{w}\|^2}{2} \\ \text{subject to} \quad & y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) \geq 1 \text{ for any } i. \end{aligned}$$

It turns out that it is much easier to solve the dual problem of this one, which we will derive in the coming steps. First, let us form the Lagrangian function to apply the KKT conditions.

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\mu}) = \frac{\|\mathbf{w}\|^2}{2} - \sum_{i=1}^{m} \mu_i (y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) - 1) \tag{2}$$

According to the KKT Theorem,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{m} \mu_i y_i \mathbf{x}_i = 0 \quad \Longrightarrow \quad \mathbf{w} = \sum_{i=1}^{m} \mu_i y_i \mathbf{x}_i \tag{3}$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{m} \mu_i y_i \quad \Longrightarrow \quad \sum_{i=1}^{m} \mu_i y_i = 0 \tag{4}$$

$$\forall i, \quad \mu[y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) - 1] = 0 \quad \Longrightarrow \quad \mu_i = 0 \quad \text{or} \quad y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) = 1 \tag{5}$$

Plugging 3 into the Lagrangian, we get

$$\mathcal{L} = \frac{1}{2} \left\| \sum_{i=1}^{m} \mu_i y_i \mathbf{x}_i \right\|^2 - \sum_{i,j=1}^{m} \mu_i \mu_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) - \sum_{i=1}^{m} \mu_i y_i b + \sum_{i=1}^{m} \mu_i,$$

which, using 4, becomes

$$\mathcal{L} = \sum_{i=1}^{m} \mu_i - \frac{1}{2} \sum_{i,j=1}^{m} \mu_i \mu_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j).$$

So, we arrived to the dual problem which is:

$$\text{maximize} \quad \sum_{i=1}^{m} \mu_i - \frac{1}{2} \sum_{i,j=1}^{m} \mu_i \mu_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{subject to} \quad \mu_i \geq 0 \quad \text{and} \quad \sum_{i=1}^{m} \mu_i y_i = 0, \forall i \in m.$$

Note that here we do not have **w**-s anymore, and, following the Lagrangian duality, we maximize $\mathcal{L}$ over μ-s, instead of minimizing over **w**-s and b, as in the original problem. Since our constraints are affine, we have strong duality. Thus, the dual problem we obtained is equivalent to the primal one we had.
Also, using 5, we can conclude that if $\mu_i \neq 0$, then $y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) = 1 \implies \mathbf{x}_i$ is a support vector. Note that since $y_i \in \{-1, 1\}$, $\frac{1}{y_i} = y_i$. It remains to plug this in $y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) = 1$ to find

$$b = y_i - \mathbf{w}^T \cdot \mathbf{x}_i.$$

Finally, using 3, we get

$$b = y_i - \sum_{j=1}^{m} \mu_j \cdot y_j \cdot \mathbf{x}_j^T \cdot \mathbf{x}_i.$$

There are different approaches to maximize $\mathcal{L}$ over μ-s. If we had an unconstrained optimization problem, we could simply use, say, the Gradient Methods. So, for maximizing $\mathcal{L}$ over μ-s, at each step we could go in the direction of the gradient of $\mathcal{L}$. However, for this problem, we should keep in mind the constraints we have. So, in order to use the ideas we are already familiar with, we can use a modification of the Gradient Descent algorithm for solving our problem. Gradient Methods tell us to go in the direction of the gradient of $\mathcal{L}$, but considering our problem, the next approximation might not belong to our constraint set. For this reason, we should consider two cases: when the next approximation belongs to our constraint set and when it does not. If the approximation does belong to the constraint set, we are happy with the approximation and can proceed with that. Otherwise, we are projecting it to "bring it back" to our constraint set. After getting the idea, let us go to the mathematical formulation of the algorithm.
Let us consider the optimization problem

$$\text{minimize } f(\mathbf{x})$$

$$\text{subject to } \mathbf{x} \in \Omega$$

First, let us define projection of **x** onto Ω to be the point $\prod[\mathbf{x}]$. $\prod[\mathbf{x}]$ is the "closest" point in Ω to **x**. Now we can $\prod$ operator to reformulate the above algorithm:

$$\mathbf{x}^{(k+1)} = \prod[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}]$$

And the algorithm above is called a *projected algorithm*. In general, the projection of **x** onto Ω is defined as:

$$\prod[\mathbf{x}] = \arg\min_{\mathbf{x} \in \Omega} \|\mathbf{z} - \mathbf{x}\|$$

It is defined in a way that $\prod[\mathbf{x}]$ is the "closest" point in Ω to $\mathbf{x}$ as we wanted. However, the operator $\prod$ is well-defined only for particular types of constraint sets. In case it is well-defined, we can apply the projected algorithm.

$$\mathbf{x}^{(k+1)} = \prod[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}]$$

The computation of the projection $\prod[\mathbf{x}]$ for a given $\mathbf{x}$ is not an easy task even if in some cases $\prod$ is well-defined. Projection $\prod[\mathbf{x}]$ is usually being computed numerically. On the other hand, the numerical computation of $\prod[\mathbf{x}]$ yields another optimization problem. And this optimization problem may be as challenging as the original one.

Let us consider the projection method for the gradient algorithm. As we know the gradient methods for unconstrained optimization problem, have the form $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \nabla f(\mathbf{x}^{(k)})$, where α_k is the step size. We choose the step size α_k depending on the gradient algorithm we use. For instance, in the steepest descent algorithm we have, $\alpha_k = \arg\min_{\alpha \geq 0} f(\mathbf{x}^{(k)} - \alpha \nabla f(\mathbf{x}^{(k)}))$.
The following form is called the projected gradient algorithm.

$$\mathbf{x}^{(k+1)} = \prod[\mathbf{x}^{(k)} - \alpha_k \nabla f(\mathbf{x}^{(k)})] \tag{6}$$

Now let us focus on the case when we have optimization problems with equality constraints.

$$\text{minimize } f(\mathbf{x})$$
$$\text{subject to } \boldsymbol{A}x = \boldsymbol{b}$$

where $f : \mathbb{R}^n \to \mathbb{R}, \boldsymbol{A} \in \mathbb{R}^{m \times n}, m < n, \text{rank } \boldsymbol{A} = m, \boldsymbol{b} \in \mathbb{R}^m$. We assume $f \in \mathcal{C}^1$. In this problem, the constraint set is $\Omega = \mathbf{x} : \boldsymbol{A}\mathbf{x} = \boldsymbol{b}$. This particular structure of the constraint set makes it easier to compute the projection operator $\prod$ as, in this case, we are allowed to use the orthogonal projector. In this setting, we define $\prod[\mathbf{x}]$ with the orthogonal projector matrix $\boldsymbol{P}$ given by

$$\boldsymbol{P} = \boldsymbol{I}_n - \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{A}$$

Let us refer to 6, which is projected gradient algorithm. In our problem setting where we have linear constraints, instead of projection operator $\prod$ we use the matrix $\boldsymbol{P}$:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \boldsymbol{P} \nabla f(\mathbf{x}^{(k)})$$

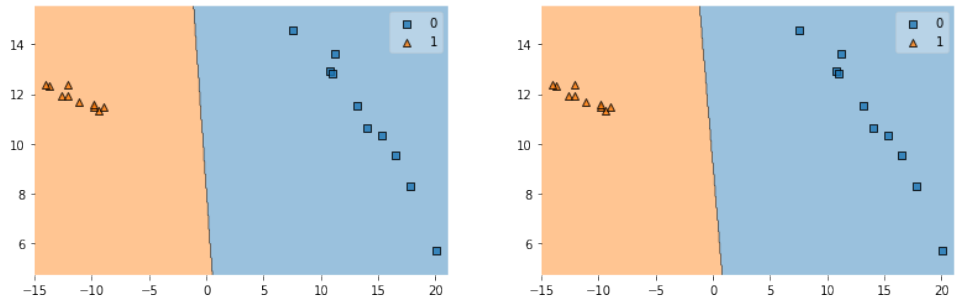
3 Experiments

3.1 Dataset

To test our implementation, we generated differen datasets, that consist of 2 classes and are linearly separable.

3.2 Comparison

We compared our implementation against that of scikit-learn library of Python, using 3 different samples. We have plotted the Separating Hyperplanes, as well as decision boundaries of the models. Figures 2, 3, and 4 show these plots, that are based on the results of running the algorithm using the same dataset as an input. The table below each of the Figures shows the comparison between the values of $\mathbf{w}$ and b.

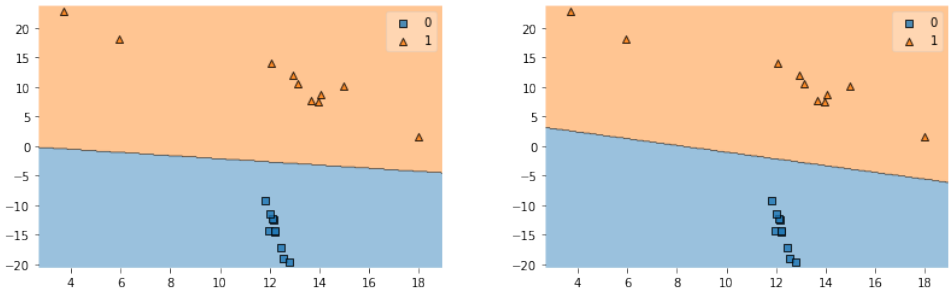


(a) Our implementation

(b) Scikit-learn implementation

Figure 2: Separating Hyperplanes and Decision Boundaries

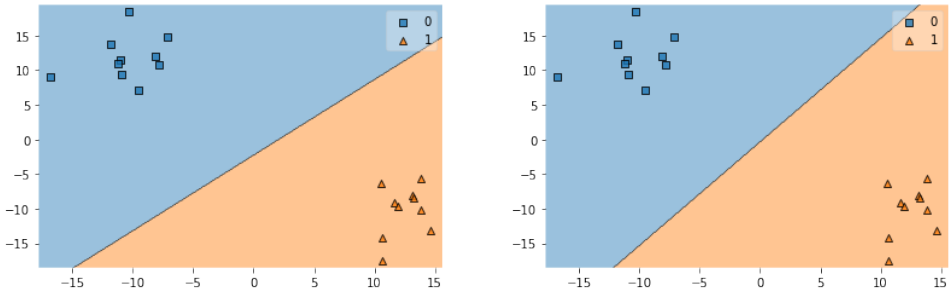
Figure 2	**w**	b
Our implementation	$[-0.1168152 \quad -0.01767305]$	0.14335702034898534
Scikit-learn implementation	$[-0.11656618 \quad -0.0214245]$	−0.19602326



(a) Our implementation (b) Scikit-learn implementation

Figure 3: Separating Hyperplanes and Decision Boundaries

Figure 3	**w**	b
Our implementation	[0.04055868 0.15194217]	−0.07836943669161078
Scikit-learn implementation	[0.07989694 0.13987615]	0.65379659



(a) Our implementation (b) Scikit-learn implementation

Figure 4: Separating Hyperplanes and Decision Boundaries

Figure 4	**w**	b
Our implementation	[0.05512101 − 0.05047589]	−0.1147240932691922
Scikit-learn implementation	[0.06875638 − 0.04599414]	0.01643352

4 Conclusion

The decision boundaries help us to visualize how SVC can help us to solve the initial problem we stated, i. e., predicting the label of a completely new observation. The idea is that if the new point will fall in the orange area, we will predict the label 1 for it. Otherwise, if it will be in the area colored in blue, the predicted label be -1. Now, comparing the result of our implementation with that of scikit-learn, we can see that whenever the new point will be far from the separating hyperplane, our predictions for the label of that new point will match. However, if the point is close enough to the Separating Hyperplane, it may happen that our model and scikit-learn's model will have different predictions.

5 Appendix

5.1 Algorithm Implementation in Python

The implementation of the algorithm, as well as the code used to do the experiments described in the paper, can be found in this[1] Github repository.
There are a few tricks used in the code that were not present in the theoretical part, which are worth mentioning here. The first one is that we transformed the dual problem, which was originally a maximization problem, to a minimization one, to apply the optimization algorithm more straightforwardly. Furthermore, we divided the dual problem, as well as the constraint $\sum_{i=1}^{m} \mu_i y_i = 0$ by the number of samples. This, without changing the problem, makes the code less prone to having overflows when working with large numbers. And the most important trick is related to calculating the intercept. In the theoretical part, we gave the formula to find the intercept using one of the support vectors. The formula we used when implementing the algorithm uses a similar idea but avoids the challenge of finding a support vector explicitly. Recall that if $\mathbf{x}_i$ has label -1, then

$$\mathbf{w}^T \cdot \mathbf{x}_i + b \leq -1.$$

Using this, we can claim that

$$max(\mathbf{w}^T \cdot \mathbf{x}_i + b) = -1.$$

Note that we are guaranteed to have at least one Support Vector satisfying to the equation above, because the idea of defining a Supporting Hyperplane was to move the Seperating Hyperplane parallelly, until reaching the first datapoint. Now, since b is not depending on $\mathbf{x}$, we can take it outside the *min* operator, and we will get the following formula for the intercept:

$$b = -1 - max(\mathbf{w}^T \cdot \mathbf{x}_i),$$

where $\mathbf{x}$ is a Support Vector with label -1.

[1] https://github.com/lussdavtyan/Optimization-Project/blob/master/Optimization%20Project%20-%20SVM.ipynb

References

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Ped+11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[Mur12] K.P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning series. MIT Press, 2012.

[Bui+13] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[CZ13] E.K.P. Chong and S.H. Zak. *An Introduction to Optimization*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2013.

[Alp14] Ethem Alpaydın. *Introduction to Machine Learning third edition*. The MIT Press, Cambridge, Massachusetts, London, England, 2014.

[MRT18] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2018.