Support Vector Machines for binary classification

Maksym Bondarenko

May 6, 2023

1 Motivation

Support Vector Machines (SVMs) are not just a powerful and widely used machine learning algorithm, but they also have a fascinating mathematical background. SVMs were first introduced by Vladimir Vapnik and his colleagues in the 1990s, and they quickly became popular due to their ability to deal with high-dimensional and complex data. Their unique features and mathematical foundations make SVMs a compelling choice for various binary classification tasks, including those encountered in everyday life.

One example of such a decision-making problem in everyday life is choosing between using public transportation or driving one's own car for commuting. This decision depends on various factors, such as the travel time for each option and the cost of parking at the destination. An efficient binary classifier can effectively analyze these factors and provide a recommendation that balances convenience and cost for the user.

SVMs offer a powerful approach to binary classification tasks, as they focus on finding the optimal decision boundary (or hyperplane) that maximizes the margin between the two classes. The margin is the distance between the decision boundary and the closest data points from each class. By maximizing the margin, the SVM algorithm ensures that the decision boundary is as robust as possible, making it less likely to be affected by noise or outliers in the data.

One of the unique features of SVMs is their use of kernel functions. Kernel functions are a way of mapping data into a higher-dimensional space, where it becomes easier to find a decision boundary that separates different classes. The idea of using kernel functions to solve complex problems goes back to the 1960s, but it was not until the introduction of SVMs that kernel methods became popular in machine learning.

SVMs are particularly good at handling non-linearly separable data, where the decision boundary is not a straight line. By mapping the data into a higher-dimensional space using a kernel function, SVMs can find decision boundaries that are non-linear in the original space. This has made SVMs a popular choice for many challenging machine learning problems, such as image and speech recognition, natural language processing, and bioinformatics.

In addition to these applications, SVMs are widely used in finance, where they are used for credit scoring, fraud detection, and portfolio optimization. In medical research, SVMs have been used to analyze gene expression data and diagnose diseases. In fact, SVMs have been used in so many different fields that it is hard to find an area of research where they have not been applied!

In this paper, we will study the theory and application of SVMs for binary classification tasks, using the public transportation vs. driving example as a motivating case study. We will delve into the mathematical foundations of SVMs, discuss the selection of appropriate kernel functions, and explore various optimization techniques for training the classifier. Ultimately, we aim to provide a comprehensive understanding of SVMs and demonstrate their efficacy in tackling real-world binary classification problems.

2 Statistical learning theory definitions

Hopefully, the motivation section convinced you that classification is something we should care about. First, let's informally state what we want from classification learning.

We start with a finite amount of labeled data generated by the environment, and given this data we want to find some function that will be able to predict unseen before labeled data well.

Now let's formalize this.

Definition
Domain set $\mathcal{X}$ is the set of all features based on which we perform classification. For the purposes of this paper, it's a subset of $\mathbb{R}^n$.

Definition
Label set $\mathcal{Y}$ is the set of labels we want to assign to the points in the domain set. For the purposes of this paper, the label set is $\{-1, 1\}$

Definition
True labeling f is a function from $\mathcal{X}$ to $\mathcal{Y}$ that we take to be ground truth labeling of the data.

Definition
Environment is domain set $\mathcal{X}$, with some probability distribution $\mathcal{D}$ over the domain set and an associated true labeling function f.

Note: probability distribution is defined in the standard way as a statistical function that describes all the possible values and likelihoods that a random variable can take within a given domain.

Definition
Example is any pair (x, y) generated by the environment so that $x \sim \mathcal{D}$ and $y = f(x)$ where $\mathcal{D}$, f describe the chosen environment. We will use y and $f(x)$ interchangeably to denote the correct labels.

In the car vs. public transportation example, we have some (unknown) distribution that produces parking prices and commute times and some unknown true labeling functions that produce satisfactory or dissatisfactory commute experiences.

When performing classification, we have access to some finite sample of examples generated by the environment. We call this sample a training set.

Definition
Training set - finite set of examples generated by the environment available for classification learning. We usually write it as $(x_1, y_1), ...(x_n, y_n)$ or denote with a letter S

Based only on this sample, our goal is to find some predictor rule that minimizes true error - the probability that we get things wrong for all points in this distribution.

Definition
Predictor of hypothesis h(x) is a function from domain set to label set that we try to learn

Definition
True error - Probability that $h(x) \neq f(x)$ for some environment $\mathcal{D}, f$

$$L_{D,f}(h) = \mathbb{P}_{x \sim D} h(x) \neq f(x)$$

Since it is a probability, $0 \leqslant L_{D,f} \leqslant 1$

This is the thing we truly care about. We want our predictor to make as few mistakes as possible.

Up to this point, every definition is simply a formalization of our stated goal. A different notion of "performing well" and environment could be defined, but for our purposes, this is perfectly fine.

Unfortunately, we have no way of knowing the true error for any predictor h. We cannot directly access either the true distribution $\mathcal{D}$ or the true labeling function f.

In our example with the commute, we neither have a way to precisely describe the types of responses generated nor do we know the nor we know the true labeling function.

From here, the only possible thing we could do is make use of the sample available and minimize the true error equivalent over the training set.

Definition
Training error - percentage of incorrectly predicted labels for our training set S of n examples by our hypothesis h. Vertical brackets denote the size of a set.

$$L_S(h) = \frac{|(x_i, y_i)\text{s. t.} h(x_i) \neq y_i|}{n}$$

From the definition, it is evident that $0 \leqslant L_s \leqslant 1$.

However, just because the function achieves a training error of 0 does not mean it will achieve a low true error. For example, a simple piece-wise function that labels each point equal to the point from the training set that we encountered with label 1 label 1, and all other inputs label -1.

$$h(x) = \begin{cases} 1 & \text{if } x = x_i \text{ for } i \text{ such that } y_i = 1 \\ -1 & \text{else} \end{cases} \tag{1}$$

This function by definition would achieve 0 training error but is extremely unlikely to achieve an acceptable true error. Classification learning which results in finding $h(x)$ that achieves 0 training error but a high true error is called over-fitting. The precise definition of "high error" depends on the use case (this is not a formal definition).

In our commute analogy, this is like deciding that only the 3 buses that we liked are good buses for choosing public transport, and in no other ways can public transport be good.

To avoid over-fitting, we introduce two assumptions into our learning process by

1. Assuming that not all functions are sensible predictors and sensible predictors can be limited to some class $\mathcal{H}$.

2. Assuming that some $h^* \in \mathcal{H}$ we choose can achieve 0 true error.

Formally:

Definition
Predictor class $\mathcal{H}$ is a set of all functions h our learning process is allowed to consider as hypotheses.

Definition
Realizability assumption states that there exists $h^* \in \mathcal{H}$ such that $L_{D,f}h^* = 0$

As we will see later, in practice we want to make H as restricted as possible while still satisfying the realizability assumption.

After picking some class $\mathcal{H}$ in which we assume there exists some (unknown) h^* which achieves 0 true error, we now want some predictor $h \in \mathcal{H}$ which performs enough" - achieves sufficiently low true error.

We can never guarantee to find the perfect predictor based on limited data. However, given enough samples and a realizable hypothesis class $\mathcal{H}$, we want to be able to guarantee with some confidence that we will be able to find some predictor that passes some threshold of a true error. We will formalize the notion of confidence and accuracy below.

Definition
Confidence δ – a probability that we didn't get a bad sample.

Definition
Accuracy ϵ - the maximum $L_{D,f}$ that we want to guarantee.

With these definitions set, we can finally introduce PAC learnability, which is a property of hypothesis classes by the Shai Shalev-Shwartz and Shai Ben-David [Shai, Shai]:

Definition (PAC Learnability)
A hypothesis class $\mathcal{H}$ is PAC learnable if there exists a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm with the following property:

- For every $\delta \in (0,1)$, for every distribution $\mathcal{D}$ over $\mathcal{X}$, and for every labeling function $f : X \to \mathcal{Y}$, if the realizable assumption holds, then when running the learning algorithm on $m \geqslant m_H(\epsilon, \delta)$ examples generated by the environment, the algorithm returns a hypothesis h such that, with a probability of at least $1 - \delta$ over the training set S, we have that:

$$L_{D,f}(h) \leqslant \epsilon$$

Definition
Sample complexity is the function m_H for a PAC-learnable hypothesis class $\mathcal{H}$

The bounds defined by the sample complexity function are a key factor in comparing different algorithms. Assuming both algorithms have the same amount of data available, the one with lower sample complexity can learn better approximation. In the example of buses, if you need to try 5 buses to learn to determine good ones and your friend needs to try 100, you are a better learner.

We will now prove a result from statistical learning theory about sample complexity of finite hypotheses classes.

Theorem (Finite classes are PAC-learnable)
Any finite H is PAC learnable with

$$m_H(\epsilon, \delta) = \frac{\log(|H|/\delta)}{\epsilon}$$

Proof We aim to prove that finite hypothesis classes are PAC learnable. Let's present the proof in a structured manner.

1. Consider a finite hypothesis class $\mathcal{H}$ with $|\mathcal{H}| = k$, where $k \in \mathbb{N}$.

2. Given the empirical risk $L_S(h)$ and the true risk $L_{D,f}(h)$, we acknowledge that $L_{D,f}(h)$ depends on the training set S, which is randomly sampled. As a result, we cannot guarantee perfect label prediction, and we introduce the accuracy parameter ϵ and the confidence parameter $1 - \delta$.

3. We define the set of "bad" hypotheses as $H_B = \{h \in \mathcal{H} : L_{D,f}(h) > \epsilon\}$ and the set of misleading samples as $M = \{S|x : \exists h \in H_B, L_S(h) = 0\}$.

4. We aim to bound the probability of the event $L_{D,f}(h_S) > \epsilon$. Due to the realizability assumption, $L_S(h_S) = 0$, meaning that this event can only happen if our sample is in the set of misleading samples, M.

5. Applying the Union Bound, we can bound the probability of the event $L_{D,f}(h_S) > \epsilon$ as:

$$P_{D^m}\{S|x : L_{D,f}(h_S) > \epsilon\} \leqslant \sum_{h \in H_B} P_{D^m}\{S|x : L_S(h) = 0\}.$$

6. Since the examples in the training set are sampled i.i.d., we can further bound the probability for each bad hypothesis $h \in H_B$ as:

$$P_{D^m}\{S|x : L_S(h) = 0\} \leqslant (1 - \epsilon)^m \leqslant e^{-\epsilon m}.$$

7. Combining the previous bounds, we have:

$$P_{D^m}\{S|x : L_{D,f}(h_S) > \epsilon\} \leqslant |H_B|e^{-\epsilon m} \leqslant |H|e^{-\epsilon m}.$$

8. To ensure the probability of the event $L_{D,f}(h_S) > \epsilon$ is at most δ, we can set:

$$|H|e^{-\epsilon m} \leqslant \delta.$$

9. Solving for m, we get:

$$m \geqslant \frac{\log(|H|/\delta)}{\epsilon}.$$

10. Since the hypothesis class $\mathcal{H}$ is finite, this proof demonstrates that finite hypothesis classes are PAC learnable with sample complexity $m_H(\epsilon, \delta) = \frac{\log(|H|/\delta)}{\epsilon}$. QED.

This is a fairly weak result since most hypothesis classes of interest are infinite in size. Proving stronger results (for infinite hypothesis classes) would require introducing more complex machinery such as VC dimension or Rademacher complexity. I decided not to do so, since it is not the central focus of the paper. What we really want from this definition is the default sample complexity for learning affine functions, which we will then compare to the sample complexity achieved by SVMs to motivate their use. I will provide the strong version of the statement we need without proof, and I will use the finite classes learnability theorem and a certain assumption to prove a weaker result. The primary purpose of this weaker result is to act as a justification for the stronger result in lieu of the actual proof.

Theorem (Sample complexity of linear hypothesis)
Any linear hypothesis class H with linear functions of dimension d is learnable with:

$$m_H = \Omega(\frac{d + \log(1/\delta)}{\epsilon})$$

Where Ω denotes the growth rate in the usual sense

This statement is true but provided without proof. Proof can be found in Understanding Machine Learning: From Theory to Algorithms by Shai Shalev-Shwartz and Shai Ben-David [Chapter 9].

Theorem (Weak sample complexity of linear hypothesis)
Any finite linear hypothesis H is learnable with:

$$m_H = \Omega(\frac{d + \log(1/\delta)}{\epsilon})$$

Proof In practice, the linear hypothesis that can be implemented by a computer is finite due to the discrete nature of the calculation. Say that a certain machine can do calculations up to k digits of precision, or, in other words, can only express n possible numbers. For a 64-bit machine n could be. Then, the total size of a possible linear hypothesis is limited by n^d where d is the dimensions of a linear hypothesis and thus its number of parameters. Then, since we now have a finite hypothesis class we can write:

$$m_H = \frac{\log(n^d/\delta)}{\epsilon})$$

$$m_H = \frac{d + \log(n/\delta)}{\epsilon})$$

Since n is a parameter of the machine rather than the algorithm, we can treat it as a constant, in which case we get:

$$m_H = \Omega(\frac{d + \log(n/\delta)}{\epsilon})$$

This is not the most satisfying proof, since saying "Well, actually the infinite class is finite in practice" cheats us out stating a proper mathematical property and may hinder our realizability assumption (can we really achieve 0 loss if the precision of our calculations is limited?).

I do want to provide it, not in the least part because Shai Shalev-Shwartz and Shai Ben-David [Chapter 3] did it to give an intuition for sample complexity bounds without having complicated statistical learning machinery, and I think providing a theorem with no justification entirely would have been worse.

In summary, so far, we have established that the learner consists of two things that are subject to choice:

- Decide which hypothesis class will be sufficient for the realizability assumption to hold. For example, one should not apply a linear classifier to a non-linear problem.
- Based on the type of data and hypothesis class, decide on which learner algorithm to use to achieve good bounds on sample complexity.

In this paper, we will take a look at one particular learning algorithm - support vector machines, and its generalization to a non-linear classification called the kernel methods. First, we will go through the mechanics of SVM, and then establish sample complexity bounds for SVM. We will then look at kernel methods mechanics and their computational complexity

3 SVM algorithm for linear classification

The learning algorithm I aim to analyze in this paper is the support vector machines. Informally, this is an algorithm that searches for the hyperplane with the largest margin between classes (assuming one exists). I will first introduce the learner and how it operates in the case of linearly separable data, and then extend it to the classification of any data.

We will assume the training set consists of m examples and the domain set is of dimensions d $\mathbb{R}^d$. We will also take the label set to be $\{+1, -1\}$ as stated in the previous section.

SVMs are applied when the linear separability assumption holds. Let's define that assumption.

Definition
A set of data S is *linearly separable* if exists $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ such that $y_i = \text{sign}(\langle w_i, x_i \rangle + b)$.

Informally, data is linearly separable if we can draw a hyperplane $H_{w,b} : wx + b = 0$ so that the points of two classes are on different sides. We call such hyperplane a decision boundary.

Definition
A decision boundary is a hyperplane $H_{w,b} : wx + b$ such that the points are on different sides or in other words $y_i = \text{sign}(\langle w_i, x_i \rangle + b)$

If we assume that our data is linearly separable, we basically assume realizability assumption for some class of functions $\mathcal{H}$ which consists of functions of the type $h(x) = \text{sign}(\langle w_i, x_i \rangle + b)$ for some parameters w, b.

Let's define this class.

Definition
Affine hypothesis class contains all functions of the type $h(x) = \text{sign}(\langle w_i, x_i \rangle + b)$

We now have a realizable hypothesis class $\mathcal{H}$, so we can try and pick some algorithm to find a predictor h in this class that we expect to have a low true error.

However, if we try and find a predictor h, looking at predictors for which $L_s(h) = 0$, we run into a problem. If the environment data is linearly separable, so is the training set (which is a subset of all possible environment data). Therefore there may be several choices w, b such that for all i in the training set $y_i = h_{w,b}(x_i) = \text{sign}(\langle w_i, x_i \rangle + b)$. This would mean that all these hypotheses have training error $L_S = 0$. Which one do we expect to give the lowest true error and how do we find it?

Intuitively, observe that each predictor $h(x) = \text{sign}(\langle w_i, x_i \rangle + b)$ corresponds to some decision boundary $H_{w,b}$. In fact, since both are fully defined by w, b we could go interchangeably between two perspectives. It is much easier to think about what type of decision boundary we want and expect to do well. Since we are dealing with a probabilistic process, the best decision boundary should allow for the largest "margin of error" while still classifying all the points correctly. If the margin defined by the is too small, it is likely that a randomly generated point from the same distribution may land on the other end up on the other sign of the line. We show in the next section that this intuition is sensible and yields an algorithm with some important advantages. For now, just trust the intuition.

So, let's try to formalize the idea of looking for a hyperplane with the largest margin of error.

Definition
Margin of separation or just margin - for a set of parameters w, b margin of separation is the distance from the corresponding hyperplane $H_{w,b}$ to the closes point x_i in the training set.

We will now cite a fact from the multivariable calculus/linear algebra for finding the distance between a point and a hyperplane.

Fact
The distance from a point x_i to a hyperplane $H_{w,b}$ is

$$\frac{|\langle w, x_i\rangle + b||}{\|w\|}$$

.

Therefore the margin-maximizing hyperplane, according to our definition, would be the one that maximizes the following:

$$\min_i \frac{|\langle w, x_i\rangle + b||}{\|w\|}$$

.

We can adjust w, b into w', b' for the any hyperplane $H_{w,b}$ so that $\min_x i|\langle w', x_i\rangle + b'| = 1$ and $H_{w',b'} = H_{w,b}$. Then we can rewrite the statement we want to maximize as:

$$\frac{1}{\|w\|}$$

.

This is equivalent to minimizing $|w|$ under the constraints that for all i

$$y_i = \text{sign}(\langle w_i, x_i\rangle + b)$$

Or equivalently

$$y_i(\langle w_i, x_i\rangle + b) \leqslant 1$$

So recap so far: we started with an assumption that our data is linearly separable, defined a hypothesis class that should be realizable for that data, and observed that intuitively, the members of that class with the best true errors should be the ones that correctly classify everything in training point and also correspond to a hyperplane with a maximum margin of separation. We finally arrived at a clear objective for finding a low-error hypothesis $h(x)$:

Minimize $||w||$ under the constraints that $\forall i \in |S|$:

$$y_i(\langle w_i, x_i\rangle + b) \leqslant 1$$

We will now use the method of lagrange multiplies to show that this simplifies to calculating a certain dot product between the vectors.

When finding a minimum of a function under a constraint, we want the constraint to be tangent to the function itself (fact I cite from multi variable calculus/Lagrange optimization). Since the gradient is perpendicular to the graph function at every point, two graphs are tangent if the gradient is parallel. In general, if $f(x)$ is the funciton to optimize and $g(x)$ is the constraint we have that:

$$\nabla f(x) = \nabla \lambda g(x)$$

Using this fact we can write down a special function called the Lagrangian which combines both constraints into one neat package to give us a solution:

$$\min L(x, a) = f(x) - ag(x)$$

which can be found at point x, a where

$$\nabla L(x, a) = 0$$

where a is some parameter. Applying to our case with $f(x) = ||w||^2/2$, $g(x) = y_i(W\vec{x}_i + b) \geqslant 1$ we get:

$$\min L_p = ||w||^2/2 - \sum_i a_i[y_i(wx_i + b) - 1]$$

$$\min L_p = ||w||^2/2 - \sum_i a_i y_i(wx_i + b) + \sum a_i$$

Since the derivative at 0 must be equal to 0, we get:

$$\partial L_p/\partial w = w - \sum a_i y_i x_i = 0$$

$$\partial L_p/\partial b = \sum a_i y_i = 0$$

Therefore, we have successfully rewritten our constrained optimization problem in Lagrangian form. Because certain conditions called Karush–Kuhn–Tucker conditions hold (as per [R. Berwick]) we can rewrite this function as a property of strong duality: maximizing some sister function is equivalent to minimizing this function. For our case:

$$\min L_p = ||w||^2/2 - \sum_i a_i[y_i(wx_i + b) - 1]$$

Since the derivative at min must be 0 we get that with respect to w:

$$\partial L_p/\partial w = w - \sum a_i y_i x_i = 0$$

$$\partial L_p/\partial b = \sum a_i y_i = 0$$

We can plug them back in to obtain dual form:

$$\max L_d = 1/2 \sum a_i a_j y_i y_j (x_i \cdot x_j) - \sum a_i a_j y_i y_j (x_i \cdot x_j)/2 + \sum a_i$$

Now our goal is to maximize this function of 1 variable, and it will give us the resulting W and b we wanted.

Observe how this function boils down to computing dot product $(x_i\dot{x}_j)$. We can rewrite our initial hypothesis $h(x)$ using these new definitions of w and b as:

$$h(x) = \text{sign}\left(\sum_i \alpha_i y_i x_i \cdot x_j) + b\right)$$

This is a great result that combined with dimension-agnostic bounds on SVMs allows their use in Kernel Machines, which are powerful binary classifers which are unfortunately outside the scope of this paper.

4 Generalization bounds of SVM

In this chapter, we will shortly justify the choice to learn SVMs with maximum margin boundary instead using other learning methods.

As stated before, intuitively we expect that a large margin of separation will protect us from random perturbations in data.

Additionally, as we saw when working with the dual form of SVM optimization, not all vectors are important. Instead, our predictor is defined by a few key support vectors closest to the boundary.

From those two things, we should expect that the sample complexity of SVM scales more graciously than the default sample complexity for learnable linear hypothesis classes. We would further expect that sample complexity (how many samples we need for a good prediction) should somehow depend by how "separable" the points are. If the classes are far apart, we need fewer support vectors and thus lower sample complexity, if the vectors are close together we would require higher amount of samples.

I give all this intuition because the statement I am about to give is incredibly hard to prove and I did not find any proof online that would make it accessible without introducing another chapter of statistical learning theory. However, with the definitions already established it should be possible to understand precisely what this statement says and why it is important. Per Shai Shalev-Shwartz and Shai Ben-David [Shai, Shai]:

Definition
Let D be a distribution over $\mathbb{R}^d \times \pm 1$. We say that D is separable with a (γ, ρ)-margin if there exists (w,b) such that $|w| = 1$ and such that with probability 1 over the choice of $(x, y) \sim D$, we have that $y(h(w^T, x) + b) \geqslant \gamma$ and $|x| \leqslant \rho$. Similarly, we say that D is separable with a (γ, ρ)-margin using a homogenous halfspace if the preceding holds with a halfspace of the form $(w,0)$.

Theorem (Sample complexity for SVMs
Let D be a distribution over $\mathbb{R}^d \times \{\pm 1\}$ that satisfies the (γ, ρ)-separability with margin assumption using a homogenous halfspace. Then, for any given $0 - 1$ error bound ε and a confidence level $1 - \delta$, the required sample size m for the output of Hard-SVM is at least

$$m_H = \Omega(\frac{4(\rho/\gamma)^2 + 2\log(2/\delta)}{\varepsilon^2})$$

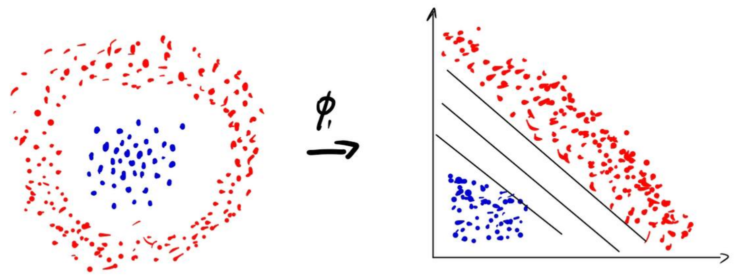
Here's the rephrased generalization bound theorem in terms of m (sample size):

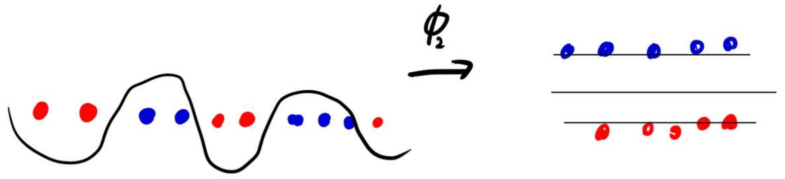
As a reminder the original bound on linear hypothesis classes was:

$$m_H = \Omega(\frac{d + \log(1/\delta)}{\epsilon})$$

This theorem states that the required sample size m for Hard-SVM depends on $(\rho/\gamma)^2$, instead of the dimensionality of the input. This is important, as it allows the algorithm to perform well if we transform data into some higher dimensional space, where the data may be linearly separable. With other algorithms for learning halfspaces, this would incur a need for more data/drop in accuracy, but SVMs we are free to do so and thus significantly increase the types of possible functions we can learn!

Bellow, I provide a visualization of how projecting data into a different space could yield linear separability.





5 References

Binary classification uses
https://machinelearningmastery.com/types-of-classification-in-machine-learning

Classification metrics
https://neptune.ai/blog/evaluation-metrics-binary-classification

MIT presentation (was very useful for me in terms of concept understanding, no proofs)
https://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf

Same MIT course examples
https://web.mit.edu/6.034/wwwbob/recitation9-fall11new.pdf https://web.mit.edu/6.034/wwwbob/recitation10-fall11.pdf

Support vector machines wikipedia
https://en.wikipedia.org/wiki/Support_vector_machine

Kernel method wikipedia
https://en.wikipedia.org/wiki/Kernel_method

Support vector machines wikipedia
https://en.wikipedia.org/wiki/Support_vector_machine

VC theory (explains generalization of SVM)
https://en.wikipedia.org/wiki/Vapnik%E2%80%93Chervonenkis_theory

Support vector machines wikipedia
https://en.wikipedia.org/wiki/Support_vector_machine

Utah presentation (haven't looked yet, but could be useful)
https://svivek.com/teaching/lectures/slides/svm/kernels.pdf

Berkeley SVMs
https://people.eecs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf

CUNY presentation on kernel trick
https://www.haralick.org/ML/kernel_trick.pdf

Multicalss classifcaiton using SVMs
https://www.analyticsvidhya.com/blog/2021/05/multiclass-classification-using-svm

Medium article on SVM
https://towardsdatascience.com/support-vector-machine-formulation-and-derivation-b146ce8

Gentle introduction to computational learning theory
https://machinelearningmastery.com/introduction-to-computational-learning-theory/

Generalization bounds on SVMs
https://stats.stackexchange.com/questions/259788/generalization-bounds-on-svm

Understanding Machine Learning: From Theory to Algorithms
https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-lear
pdf