Support Vector Machines - Group 32

Kartik.S, Tanmay.G, Gautham.B, Dishank.J, Aayush.P

Indian Institute Of Technology, Hyderabad

Introduction to SVM's

Support Vector Machine (SVM) is one of the most popular Machine Learning Classifiers. It falls under the category of Supervised learning algorithms and uses the concept of a Margin to classify between classes. In our discussion today, we will focus on how an SVM solves the problem of Binary Classification of a set of data points.

Empirical Risk Minimization

The core idea of any (Supervised) Machine learning model is to predict a certain output y using an input x correctly, given a certain amount of training data. How will I answer the following Questions

1. How do I concretely define my "ML model"?
2. How Expressive is my ML model?
3. Is my ML model doing "good"?

To answer (2) we need to define some quantity that measures the strength of a certain classification model
To answer (3) we need to define a measure of some sort of "error"

Empirical Risk Minimization

What is an ML model?

Any machine F tries to learn a certain hypothesis with certain parameters λ so that it fits the data "well". There are several types of complicated Hypothesis that a particular machine can learn. A linear SVM will try to learn parameters $\vec{W}$ and b and create a linear classifier of the form :-

$$F_{W,b}(\vec{X}) = \text{sgn}(\vec{W}^T\vec{X} + b) \in \mathcal{H} = \text{Linear classifiers}$$

Every machine learning Model aims at picking a Certain **Hypothesis function** f_λ with parameters $\lambda \in \Lambda$. The set of possible hypothesis functions is called the Hypothesis Space and is denoted by $\mathcal{H}$

Empirical Risk Minimization

The Training error

Say My machine has chosen a certain $f_\lambda \in \mathcal{H}$ and I wish to calculate the training error of a set of m Data points $(\vec{x_i}, y_i)$ then the error will be defined by the average of absolute errors of all the data points.

$$R^{train}(\lambda) = \frac{1}{m} \sum_{i=1}^{i=m} |y_i - f_\lambda(\vec{x_i})| \tag{1}$$

This Model assumes that the set of data points are basically samples selected from a set of Independent and identically distributed Random variables(i.i.d).This allows us to define the **Expected Error** on any single data point. we define the expected error (or the **probability of misclassification**) as:

$$R^{test}(\lambda) = \mathbb{E}\left[\frac{1}{2}|y - f_\lambda(\vec{x})|\right] = \iint_{\vec{x},y} \frac{1}{2}|y - f_\lambda(\vec{x})|P(\vec{x}, y)\, d\vec{x}\, dy \qquad (2)$$

Empirical Risk Minimization

Unanswered Questions

1. We have defined the two types of error. But is there a relationship between them ?
2. ML models usually minimize the first ($R^{train}(\lambda)$) but does that necessarily imply minimization of the second?

The second question can be answered by the law of large Numbers, and the first using VC dimensions. The law of Large Numbers guarantees(generally) that as the sample size grows large:-

$$\mathbb{E}\left[|y - f_\lambda(\vec{x})|\right] \approx R^{train}(\lambda) = \frac{1}{m}\sum_{i=1}^{i=m} |y_i - f_\lambda(\vec{x_i})| \tag{3}$$

Framing a Linear Discriminator

Existence of a Linear Classifier

1. We are given a set of n-tuples $(\vec{x_i}, y_i)$, i=1,...,n, where, $x_i \in R^n$ and $y_i \in \{-1, 1\}$.
2. We can theoretically find a separating hyperplane by solving the following feasibility problem:

$$\max 0$$
$$y_i(w^T x_i + b_i) > 0$$

3. A strict inequality constraint is not practically implementable in CVXPY or another solvers as a result the solver converts a strict inequality to a non-strict one like this:

$$\max 0$$
$$y_i(w^T x_i + b_i) \geq 0$$

4. The problem with setting the following constraints is that the values of $w, b = 0$ satisfy the constraints and the solver will return these values to us. **This will yield w , b = 0!**

The Linearly Separable Case: Defining the Variables

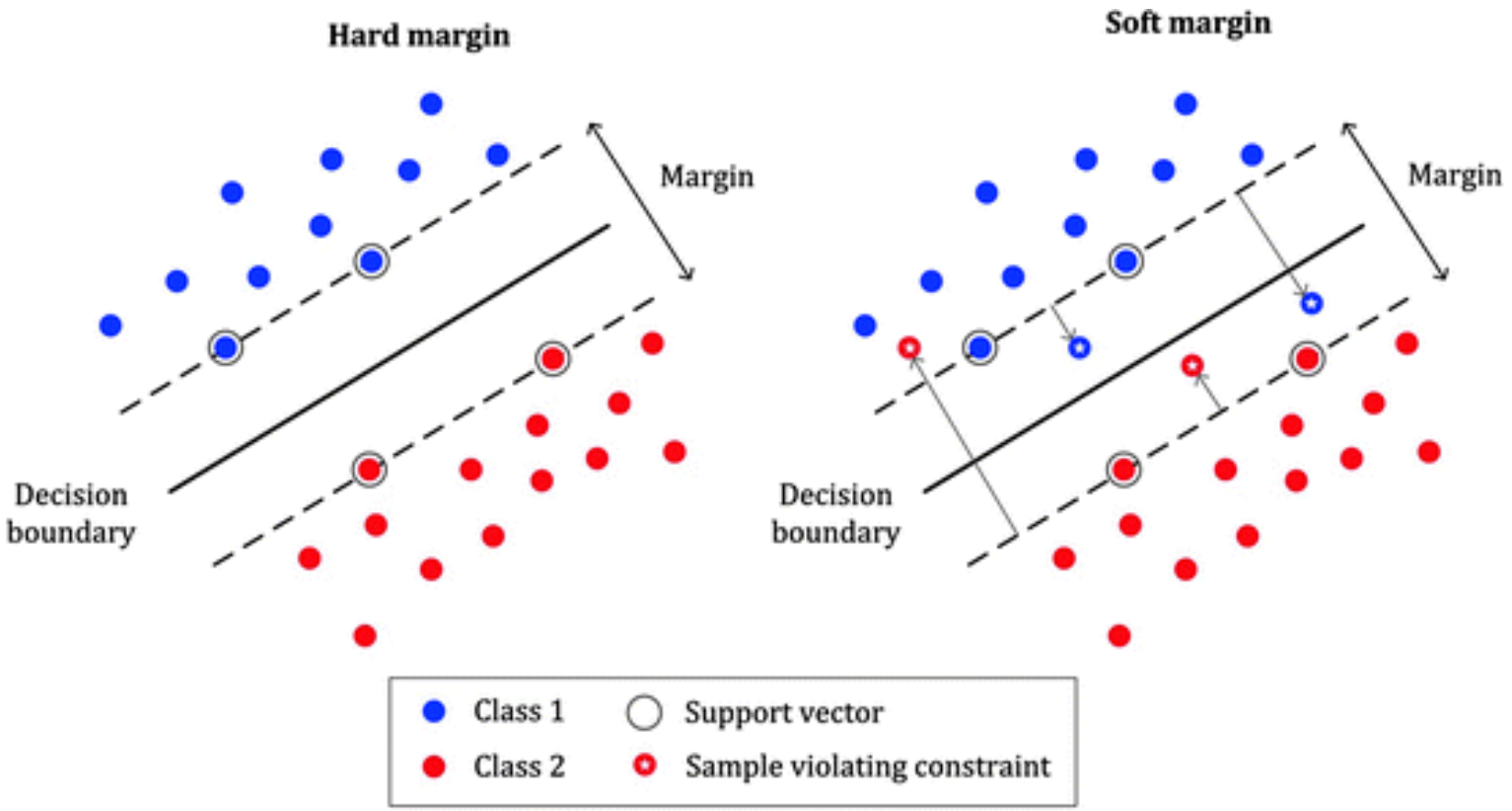
1. We begin our discussion by understanding how a SVM can classify linearly separable data classified into two classes.
2. The goal is to find ($\vec{w}$, b) such that:

$$\vec{w}.\vec{x_i} + b >= 1, \forall i \; y_i = 1$$
$$\vec{w}.\vec{x_i} + b <= -1, \forall i \; y_i = -1$$

3. Since the data is linearly separable some ($\vec{w}$, b) have to exist which satisfy the above conditions. In fact, if we can find one such pair, we can infinite other pair by multiplying them with a positive constant. To remove this redundancy, lets add one normalisation constraint:

$$\min_{i=1...n} |\vec{w}.\vec{x_i} + b| = 1$$

The Linearly Separable Case



The Linearly Separable Case: The Maximal Margin

1. Continuing from the above normalisation, we will get two hyperplanes such that:

$$\vec{w}.\vec{x_i} + b = 1$$
$$\vec{w}.\vec{x_j} + b = -1$$
For some i, j

2. These are two parallel hyperplanes and the distance between which we will henceforth refer to as maximal margin is $= \frac{2}{\|\vec{w}\|}$.
3. Now we frame our optimisation problem under the hypothesis that a "good" classifier will maximise the "margin" between the two classes of data.

The Linearly Separable Case: Framing the optimisation problem

1. Formulation(Note the combined constraint):

$$\max_{\vec{w},b} \frac{2}{\|\vec{w}\|}$$
$$\text{s.t. } y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1 \; \forall i = 1....n$$

2. Reformulating into an equivalent convex optimisation problem:

$$\min_{\vec{w},b} \frac{\|\vec{w}\|^2}{2}$$
$$\text{s.t. } y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1 \; \forall i = 1....n$$

3. This is a standard quadratic programming problem with linear constraints.

The Linearly Separable Case: Using Lagrangian Multipliers

1. Lagrangian of the above the optimisation problem.

$$L(\vec{w}, b, \vec{\lambda}) = \frac{\|\vec{w}\|^2}{2} - \sum_{i=1}^{n} \lambda_i [y_i(\vec{w} \cdot \vec{x_i} + b) - 1] \tag{4}$$

where $\vec{\lambda}$ is the vector of non negative Lagrange multipliers.

2. Differentiating the Lagrangian w.r.t the parameters $\vec{w}$, b and equating them to 0(for saddle point), we get:

$$\frac{\partial L(\vec{w}, b, \vec{\lambda})}{\partial \vec{w}} = \vec{w} - \sum_{i=1}^{n} \lambda_i y_i \vec{x_i} = 0 \tag{5}$$

$$\frac{\partial L(\vec{w}, b, \vec{\lambda})}{\partial b} = \sum_{i=1}^{n} \lambda_i y_i = 0 \tag{6}$$

Framing the dual problem using KKT conditions

I will **Impose the condition** $\vec{\lambda} \cdot \vec{y} = 0$ and $\lambda \geq 0$ in my solver. Plugging in the value of w , we get the final dual problem:

$$\max_{\lambda} \quad \vec{\lambda} \cdot \vec{1} - \frac{1}{2} \vec{\lambda}^T D \vec{\lambda} \tag{7}$$

$$\vec{\lambda} \cdot \vec{y} = 0 \tag{8}$$

$$\vec{\lambda} \geq 0 \tag{9}$$

where D is a matrix such that $D_{ij} = y_i y_j \vec{x_i}^T \vec{x_j}$

Meaning of the Complementary slackness

Support Vectors

The complementary slackness conditions are as follows :-

$$\lambda_i(y_i(\vec{w}^T x_i + b_i) - 1) = 0$$

This shows the data points corresponding to $\lambda_i > 0$ are forced to be the at minimum distance form the hyperplane! Moreover, the other data points are **irrelevant** to the calculation of $\vec{w}$

$$\vec{w}^* = \sum_{i|\lambda_i>0} \lambda_i y_i \vec{x_i}$$

$$b^* = y_i - \vec{w}^* \cdot \vec{x_i}$$

These vectors $\vec{x_i}$ are called **Support vectors**. Fun fact: The values of w and b depend only on support vectors.

Code Snippets

Hard Margin Classifier(Dual)

The data for the hard margin classifier has been taken from Lab Tutorial 11. All features of the data set was used to train and 2 features were used to visualise the trained model

```
lambd = cp.Variable(shape=(100,))
M = np.concatenate((X[[i1, i2], :], -Y[[i1, i2], :]), axis=1)
D = M.T @ M + np.eye(M.shape[1]) * 1e-5
assert(np.all(np.linalg.eigvals(D) >= 0))

objective = cp.Minimize(-cp.sum(lambd) + 0.5*cp.quad_form(lambd, D))
constraints= [lambd >= 0, cp.sum(lambd[0:50] - lambd[50:100]) == 0]
problem = cp.Problem(objective, constraints)
problem.solve(solver=cp.ECOS, verbose=False)

w = np.sum(np.multiply(lambd.value,M), axis=1)
index = np.argmax(lambd.value)
b = 1 - (w.T@M[:,index])[0,0]
```

✓ 0.3s

Figure: Code Snippet

Plot for test data

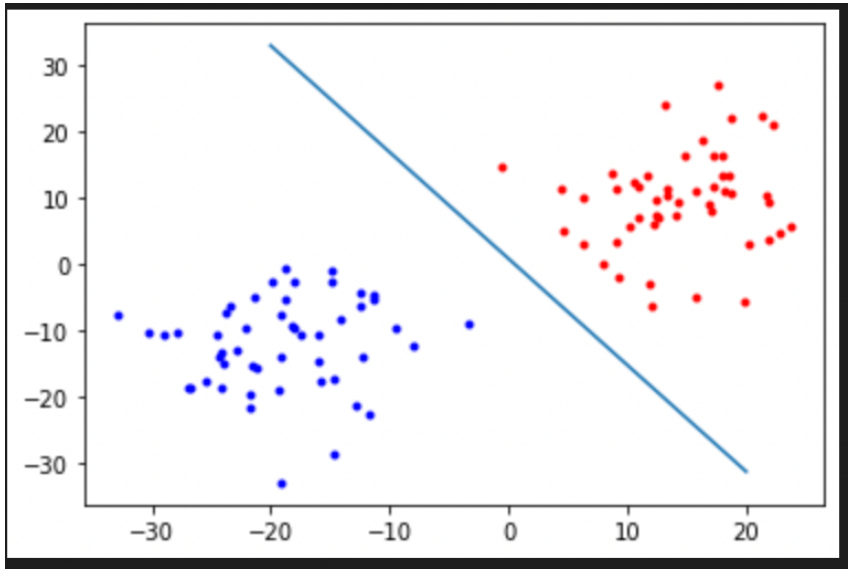


Figure: Visualized Plot

Definition

A set S of examples is shattered by a set of functions $\mathcal{H}$ if for every partition of the examples in S into positive and negative examples there is a function in the Hypothesis space $\mathcal{H}$ that gives exactly these labels to the examples

Let us understand Shattering with an example. Let us take a 2 class problem with N points that can be labelled as positive and negative.
If, for all possible labelings, a function can be found in $\mathcal{H}$ consistent with the labeling, we can say that $\mathcal{H}$ can *shatter* the set

Let us take 3 points that are not on a line and without loss of generality let us take some labellings as an example

This is how we can use a straight line to classify these three labelled points into 2 areas each of which are in their respective spaces

Definiton

The VC-dimension of a hypothesis class $\mathcal{H}$, denoted $VCdim(\mathcal{H})$, is the maximal size of a set $C \subset X$ (X is the instance space) that can be shattered by $\mathcal{H}$. If $\mathcal{H}$ can shatter sets of arbitrarily large size we say that $\mathcal{H}$ has infinite VC-dimension.
An unbiased hypothesis space shatters the entire instance space.

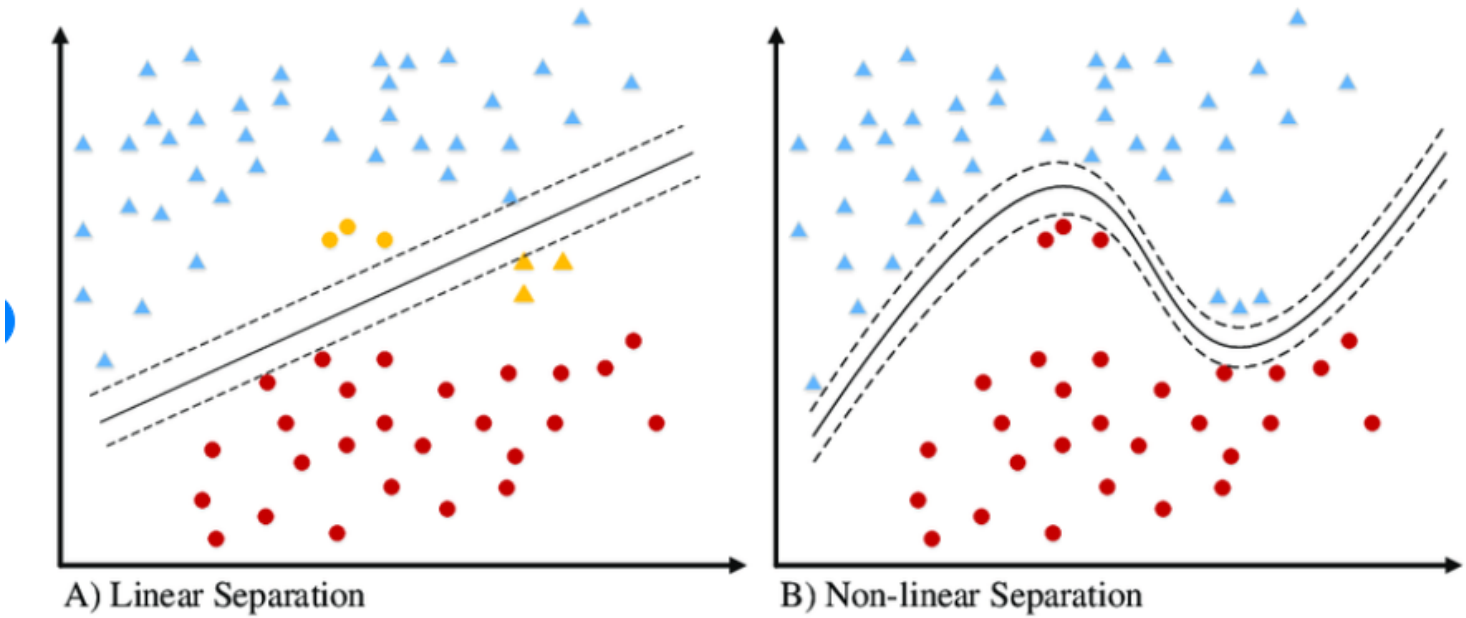
Meaning

- It characterises the expressive power of a **hypothesis class**
- It is measured by hypothesis class **shattering** n points

Structural Risk Minimization

VC dimension - Intuition

To understand what we mean by the "strength" of a certain ML model. Lets look at the example below:-



The second model uses a more complex - non linear model to classify the data.

The first model is more expressive and is more prone to mis -classification(High Bias) , but the advantage of this model is that it will generalize well to new data (The Hypothesis function has low variance over this dataset)

The second model is more expressive and is less prone to mis -classification(Low Bias) , but the downside of this model is that it may not generalize well to new data (The Hypothesis function has high variance over this dataset)

So to Summarize - Models with more expressive capability tend to have a greater variance and therefore higher test set - error

The generalization error bounds the difference between **true risk** and **empirical risk** with a certain **probability** $= 1 - \eta$

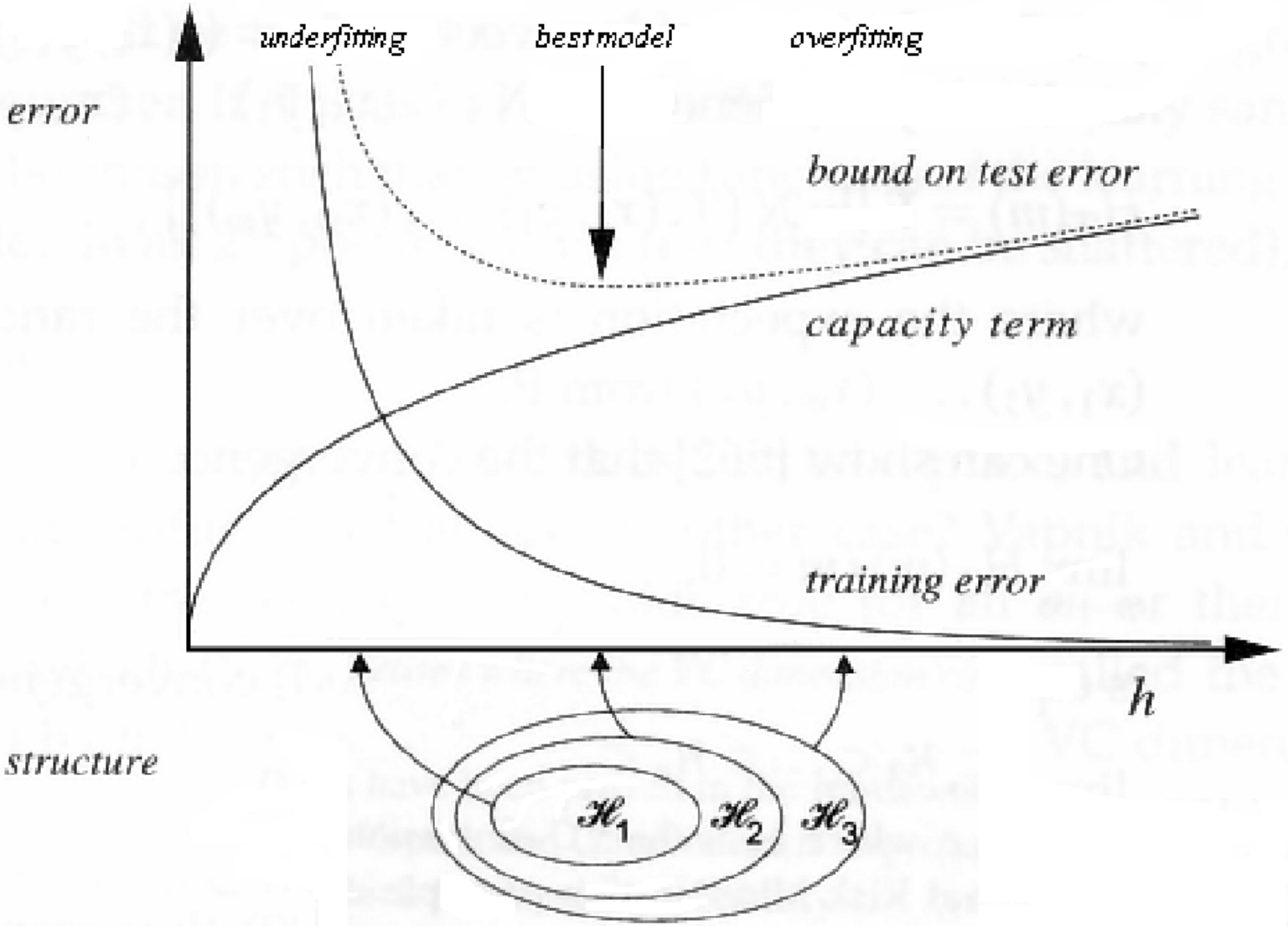
$$R^{test}(\lambda) \leq R^{train}(\lambda) + \sqrt{\frac{h(\ln \frac{2l}{h} + 1) - \ln \frac{\eta}{4}}{l}} \tag{10}$$

Here h is the VC Dimension of f_λ. We can assume that n will mostly remain constant, so our entire relationship comes out as

$$R^{test}(\lambda) \leq R^{train}(\lambda) + \sqrt{\frac{complexity}{l}} \tag{11}$$

Structural Risk Minimization

The Plot of the Test Error



VC dimensions in SVM's

We need to make some sorted of a "nested" Hypothesis model for support vector machines..How do we do this? we constrain the Norm of the vector w , after constraining the minimum possible distance of a particular data - point.

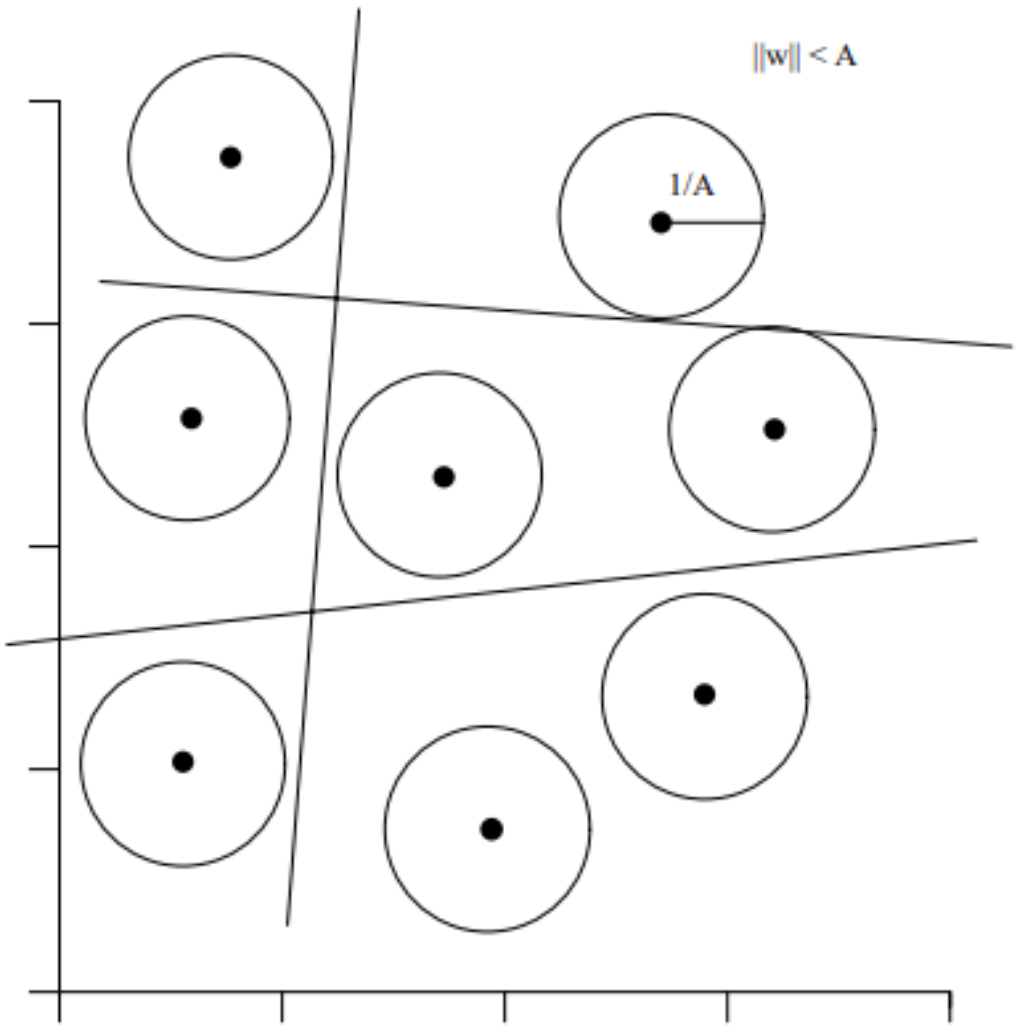
$$d_{min} = \frac{|w \cdot \vec{x_i} + b|_{min}}{\|w\|} = \zeta = 1$$

$$\mathcal{H}_A = \{f_{W,b}(x) = \text{sign}(w \cdot \vec{x} + b) \mid \|w\| \leq A\}$$

$$d_{min} = \frac{1}{||A||}$$

VC dimensions in SVM'S

Drawing the Canonical Hyperplane



VC dimensions in SVM

Bound on the VC dimension

We need to select a Hypothesis class $\mathcal{H}_A$ such that the V.C dimension is minimized (to reduce the generalization error $R_{test}(\lambda)$).From the diagram we can see why decreasing the Value of $\|A\|$ actually affects the nature of separation of my model. It can be proven for a certain arrangement of data points within a unit sphere :-

$$h \leq \min\{R^2A^2, N\} + 1$$

I will want A to be as small as possible for any input data set, i.e I will constrain my feasible w,b to minimum values of $\|w\|$. So that the function can belong to $\mathcal{H}_A$ with smallest A

Soft Margin Classifiers

1. Now we consider the case when the data is not linearly separable. In such a case, one may want to separate the training set with a minimal number of errors.
2. To incorporate this notion, we introduce non-negative variables ξ_i, i=1,...,n, such that our new optimization problem is:

$$\min_{\vec{w},b,\vec{\xi}} \frac{||\vec{w}||^2}{2} + C(\sum_{i=1}^{I} \xi_i)^k$$
$$\text{s.t. } y_i(\vec{w} \cdot \vec{x} + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$

3. Thus we allow our classifier to commit ξ_i error for sample i, but we penalize the total error using the addition of the second term in the objective function. We can control C and k so as to control how we want to penalize the error.

Soft Margin Classifiers Contd.

1. The Lagrangian for the soft-margin problem is

$$L(\vec{w}, b, \vec{\lambda}, \vec{\xi}, \vec{\gamma}) = \frac{||\vec{w}||^2}{2} - \sum_{i=1}^{n} \lambda_i [y_i(\vec{w} \cdot \vec{x_i} + b) - 1 + \xi_i] - \sum_{i=1}^{n} \gamma_i \xi_i + C(\sum_{i=1}^{n} \xi_i)^k \quad (12)$$

2. We have to minimize this Lagrangian with respect to $\vec{w}, \vec{\xi}$ and b and maximize this Lagrangian with respect to $\vec{\lambda}$ and $\vec{\gamma}$.

Soft Margin Classifiers Contd.

1. Differentiating the Lagrangian and setting the partial derivatives to 0 gives

$$\frac{\partial L(\vec{w}, b, \vec{\lambda}, \vec{\xi}, \vec{\gamma})}{\partial \vec{w}} = \vec{w} - \sum_{i=1}^{n} \lambda_i y_i \vec{x}_i = 0 \tag{13}$$

$$\frac{\partial L(\vec{w}, b, \vec{\lambda}, \vec{\xi}, \vec{\gamma})}{\partial b} = \sum_{i=1}^{n} \lambda_i y_i = 0 \tag{14}$$

$$\frac{\partial L(\vec{w}, b, \vec{\lambda}, \vec{\xi}, \vec{\gamma})}{\partial \vec{\xi}} = \begin{cases} kC(\sum_{i=1}^{n} \xi_i)^{k-1} - \lambda_i - \gamma_i = 0 & k > 1 \\ C - \lambda_i - \gamma_i = 0 & k = 1 \end{cases} \tag{15}$$

Soft Margin Classifiers Contd.

1. When k > 1,

$$\sum_{i=1}^{n} \xi_i = \left(\frac{\delta}{Ck}\right)^{\frac{1}{k-1}}$$

   where $\delta - \lambda_i - \gamma_i = 0$

2. From equation (13), we get

$$\vec{w}^* = \sum_{i=1}^{n} \lambda_i^* y_i \vec{x}_i$$

3. Substituting in the Lagrangian, we get

$$L(\vec{\lambda}, \delta) = \sum_{i=1}^{n} \lambda_i - \sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \vec{x}_i \cdot \vec{x}_j - \frac{\delta^{\frac{k}{k-1}}}{(kC)^{\frac{1}{k-1}}}\left(1 - \frac{1}{k}\right) \quad (16)$$

Soft Margin Classifiers Contd.

1. Our new optimisation problem is

$$\max_{\vec{\lambda},\delta} \; L(\vec{\lambda},\delta) = \vec{\lambda}\cdot\vec{1} - \frac{1}{2}\vec{\lambda}\cdot D\vec{\lambda} - \frac{\delta^{\frac{k}{k-1}}}{(kC)^{\frac{1}{k-1}}}\left(1-\frac{1}{k}\right) \tag{17}$$

$$\text{s.t. } \vec{\lambda}\cdot\vec{y} = 0 \tag{18}$$

$$\vec{\lambda} \leq \delta \tag{19}$$

$$\vec{\lambda} \geq 0 \tag{20}$$

   where D is a symmetric nxn matrix with elements $D_{ij} = y_i y_j \vec{x_i}\cdot\vec{x_j}$

2. For the simple case of k $= 2$, the objective function becomes

$$L(\vec{\lambda},\delta) = \vec{\lambda}\cdot\vec{1} - \frac{1}{2}\vec{\lambda}\cdot D\vec{\lambda} - \frac{\delta^2}{4C} \tag{21}$$

   which is a quadratic programming problem!!

Code Snippets

Soft Margin Classifier

The data for the soft-margin classifier has been taken from Lab Tutorial 11 and two features were found to make the data linearly inseparable.

```
k = 3
C = 0.1
objective = cp.Minimize( -cp.sum(lambd) + 0.5*cp.quad_form(lambd, D) + (lambd[0]+gamma[0])**(k/(k-1))/(k * C)**(k/(k-1)) * (1-1/k))

constraints= [lambd >= 0, cp.sum(lambd[0:50] - lambd[50:100]) == 0, lambd <= (lambd[0]+gamma[0]), lambd+gamma==C*np.ones(100)]
problem = cp.Problem(objective, constraints)
problem.solve(solver=cp.ECOS, verbose=False)
```

Figure: Code Snippet

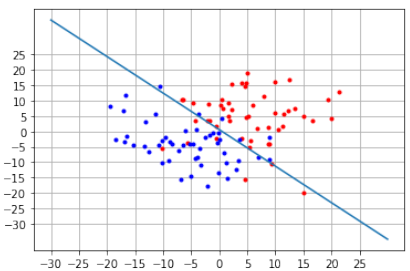


Figure: Visualized Plot

Kernels

So far we have talked about SVMs with linear decision surfaces, which might not be appropriate for certain tasks. We now transform our input variable $\boldsymbol{x}$ into higher dimensional feature space, and try to find a linear decision surface in that space.

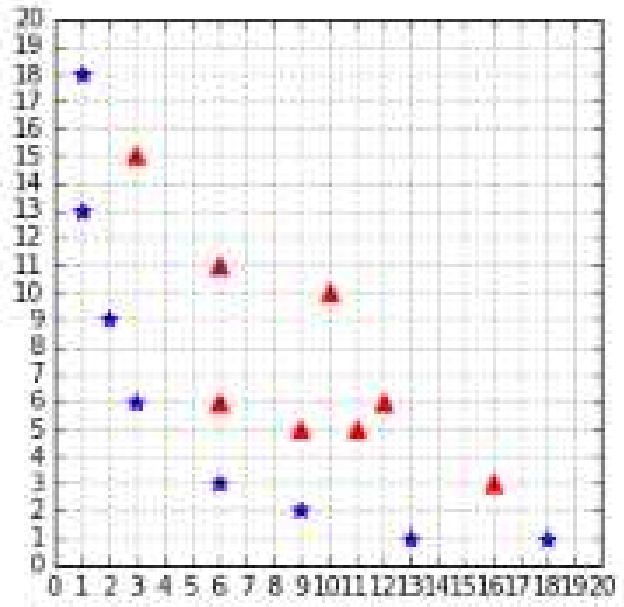


Figure: Linearly inseparable data

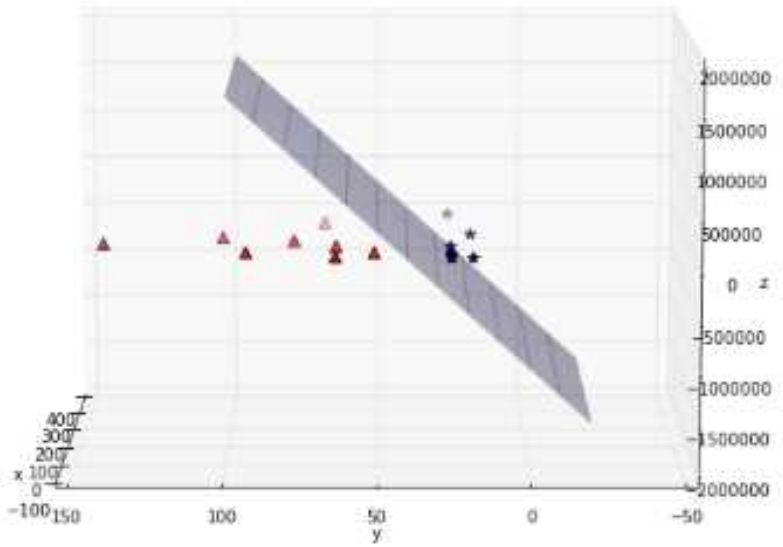


Figure: Data transformed into a higher dimension

Kernels

We map the input variable $\vec{x}$ to a vector of a higher dimension using the transformation:

$$\vec{x} \to \vec{\phi}(\vec{x}) = (a_1\phi_1(\vec{x}), a_2\phi_2(\vec{x}), \ldots, a_n\phi_n(\vec{x}), \ldots) \tag{22}$$

where $\{a_n\}_{n=1}^{\infty} \in \mathbb{R}$ and $\{\phi_n\}_{n=1}^{\infty}$ are some real functions. The Soft Margin Version of the SVM is then applied.

$$f(\vec{x}) = \text{sign}(\phi(\vec{x}) \cdot \vec{w}^* + b^*) \tag{23}$$

$$= \text{sign}(\sum_{i=1}^{l} y_i \lambda_i^* \phi(\vec{x}) \cdot \phi(\vec{x}_i) + b^*) \tag{24}$$

The Kernel function is defined as

$$K(\vec{x}, \vec{y}) = \phi(\vec{x}) \cdot \phi(\vec{y}) \tag{25}$$

Kernel Trick

The decision boundary in the higher dimensional feature space is given by

$$f(\vec{x}) = \text{sign}(\sum_{i=1}^{I} y_i \lambda_i^* K(\vec{x}_i, \vec{x}_j) + b^*) \tag{26}$$

The optimization problem (17 - 20) now becomes

$$\max_{\vec{\lambda},\delta} \; L(\vec{\lambda}, \delta) = \vec{\lambda} \cdot \vec{1} - \frac{1}{2}\vec{\lambda} \cdot D\vec{\lambda} - \frac{\delta^{\frac{k}{k-1}}}{(kC)^{\frac{1}{k-1}}}\left(1 - \frac{1}{k}\right) \tag{27}$$

$$\text{s.t. } \vec{\lambda} \cdot \vec{y} = 0 \tag{28}$$

$$\vec{\lambda} \leq \delta \tag{29}$$

$$\vec{\lambda} \geq 0 \tag{30}$$

where D is a symmetric nxn matrix with elements $D_{ij} = y_i y_j K(\vec{x}_i, \vec{x}_j)$

Kernel Examples

Let us take the transformation $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^6$ defined as

$$\phi(\vec{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2) \tag{31}$$

The Kernel Function for this transformation would be

$$\begin{aligned} K(\vec{x}, \vec{y}) &= \phi(\vec{x}) \cdot \phi(\vec{x}) && (32)\\ &= 1 + 2x_1y_1 + 2x_2y_2 + x_1^2y_1^2 + x_2^2y_2^2 + 2x_1y_1x_2y_2 && (33)\\ &= (1 + \vec{x} \cdot \vec{y})^2 && (34) \end{aligned}$$

This is called the polynomial kernel of the second degree. Note that we can compute the kernel function without explicitly transforming our data into higher dimension and feeding it to the SVM.

Kernel Examples

Some commonly used Kernels used include

Kernel Function, $K(\vec{x}, \vec{y})$	Type of Classifier
$\exp\left\{-\|\vec{x} - \vec{y}\|^2\right\}$	Gaussian RBF
$(1 + \vec{x} \cdot \vec{y})^d$	Polynomial of degree d
$\tanh(\vec{x} \cdot \vec{y} - \theta)$	Multi-Layer Perceptron

Code Snippets

Polynomial Kernel

The data for testing the polynomial kernel is a randomly generated dataset

```
1  def polynomial_kernel(M):
2      return np.array([1, 2**0.5*M[0], 2**0.5*M[1], 2**0.5*np.multiply(M[0],M[1]),M[0]**2, M[1]**2])
3
4  points_in_6d = np.apply_along_axis(polynomial_kernel, 0, points)
```

Figure: Code Snippet

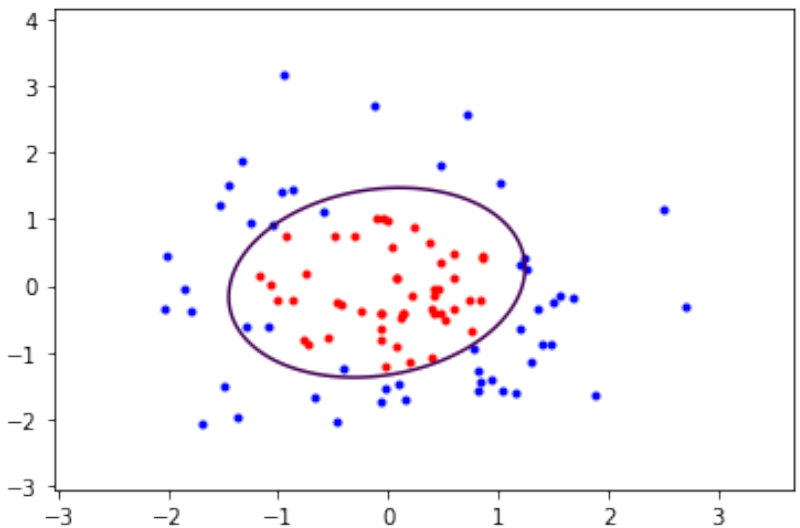


Figure: Final Plot

Code Snippets

Gaussian Kernel

The data for testing the Gaussian kernel is a randomly generated dataset

```
3  YiYj = labels[:, np.newaxis] @ labels[:, np.newaxis].T
4  #print(YiYj)
5  D = np.multiply(YiYj, np.exp(-1*norm_matrix/2))
```

Figure: Code Snippet

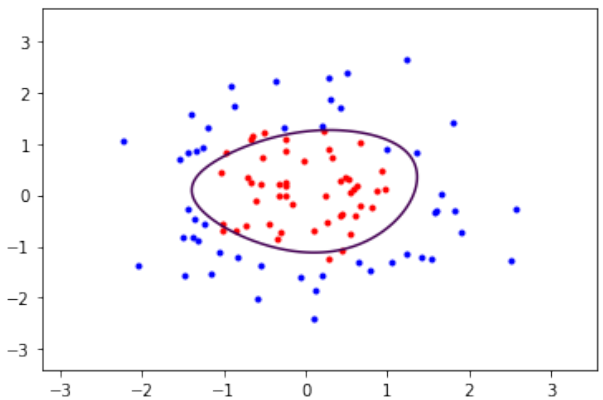


Figure: Final Plot

Spam Classification

Introduction

We are using Andrew Ng's Machine Learning Course Dataset for spam classification.

- We are doing a binary classification as spam and not spam
- We have a vocabulary list which contains top 1899 words that occur in generic spam emails
- Each word in the email is first pre-processed into tokens which means each word is given a specific number

```
1 aa
2 ab
3 abil
...
86 anyon
...
916 know
...
1898 zero
1899 zip
```

Figure: Vocabulary list

Spam Classification

Approach

- We extract features from the email, which is a vector of size 1899
- A 0 in the i^{th} position indicates that the word is not there in the email and 1 indicates otherwise
- After this, we train our SVM classifier on the training dataset provided in the course
- In order to test the accuracy of the model, we have used the test data provided in the course

Code Snippets

Soft Margin Classifier Primal

The data for the soft margin is taken from Andrew Ng Machine Learning Course for Spam Classification. The accuracy of test data has been shown below

```
objective_soft = cp.Minimize(0.5*cp.norm(w,2)**2 + C*(cp.sum(zeta))**k)

constraints_soft = [
    zeta >= 0,
    cp.multiply(y_label, (w.T@M + b)) >= 1 - zeta,
]

problem_soft = cp.Problem(objective_soft, constraints_soft)
problem_soft.solve(solver=cp.SCS)
```

Figure: Code Snippet

```
Accuracy on test  0.981
Mean of test label  0.308
Accuracy on Train  0.99975
```

Figure: Accuracy Plot