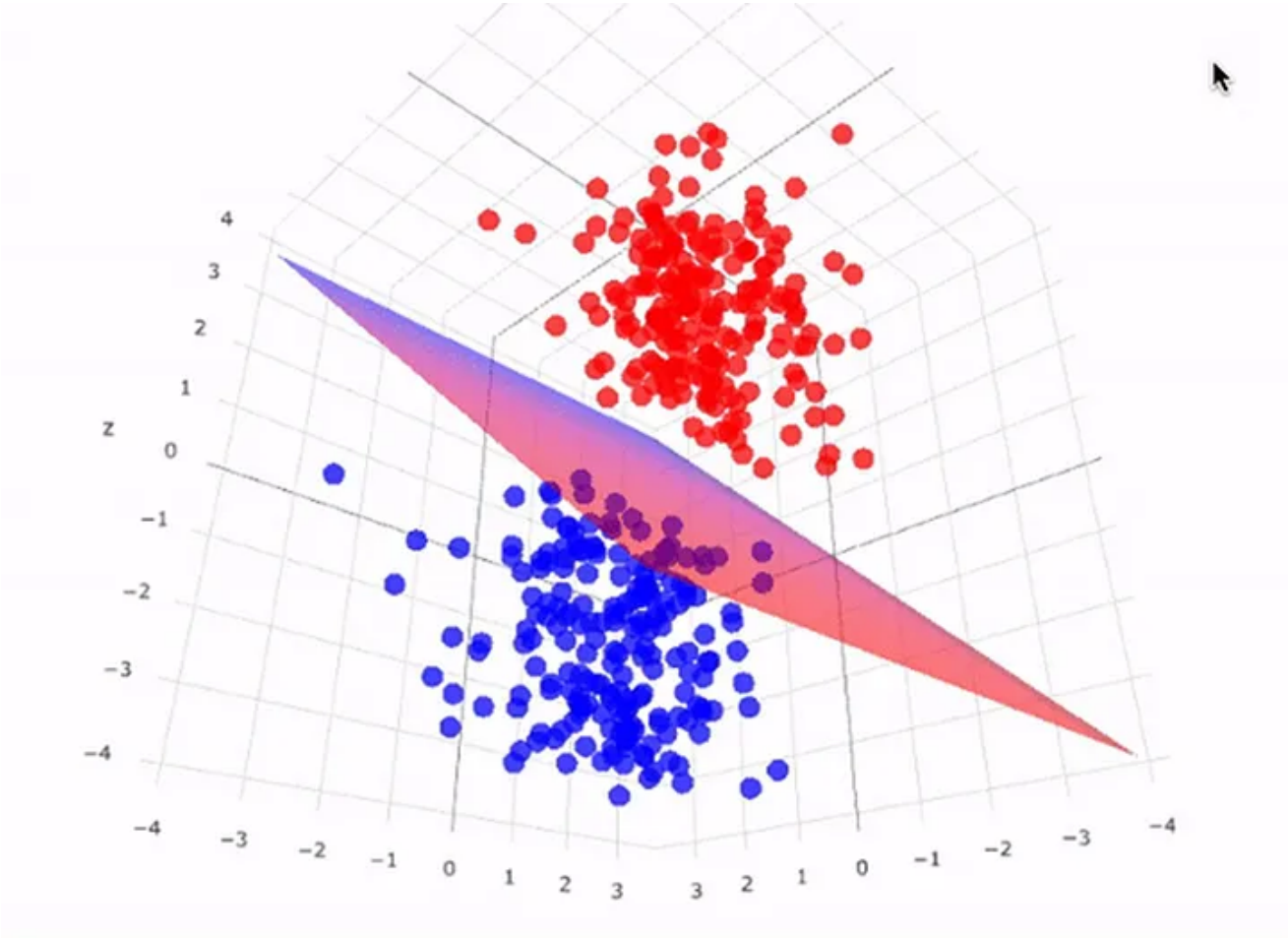
DATA SCIENCE THEORY | ML ALGORITHMS | KNIME ANALYTICS PLATFORM

Support Vector Machines (SVM): An Intuitive Explanation

Everything you always wanted to know about this powerful supervised ML algorithm

Tasmay Pankaj Tibrewal · Follow
Published in Low Code for Data Science
17 min read · Jul 1, 2023

Support Vector Machines (SVMs) are a type of supervised machine learning algorithm used for classification and regression tasks. They are widely used in various fields, including pattern recognition, image analysis, and natural language processing.



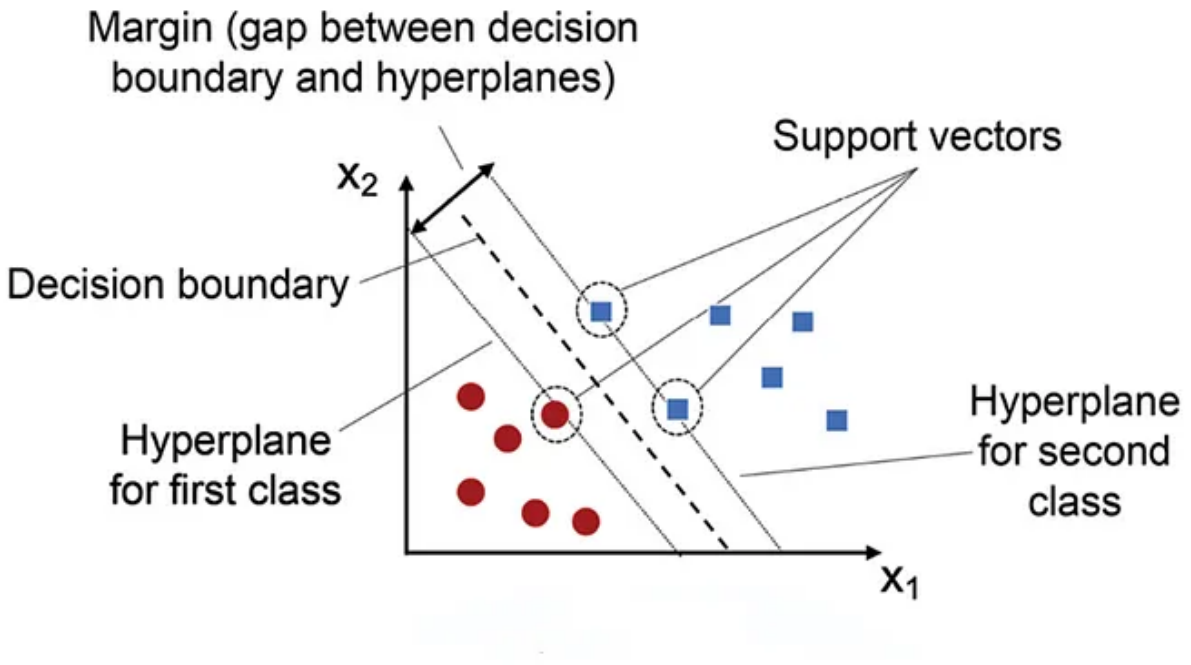
SVMs work by finding the optimal hyperplane that separates data points into different classes.

Key Terms

Hyperplane

A **hyperplane** is a decision boundary that separates data points into different classes in a high-dimensional space. In two-dimensional space, a hyperplane is simply a line that separates the data points into two classes. In three-dimensional space, a hyperplane is a plane that separates the data points into two classes. Similarly, in **N-dimensional space**, a hyperplane has **(N-1)-dimensions**.

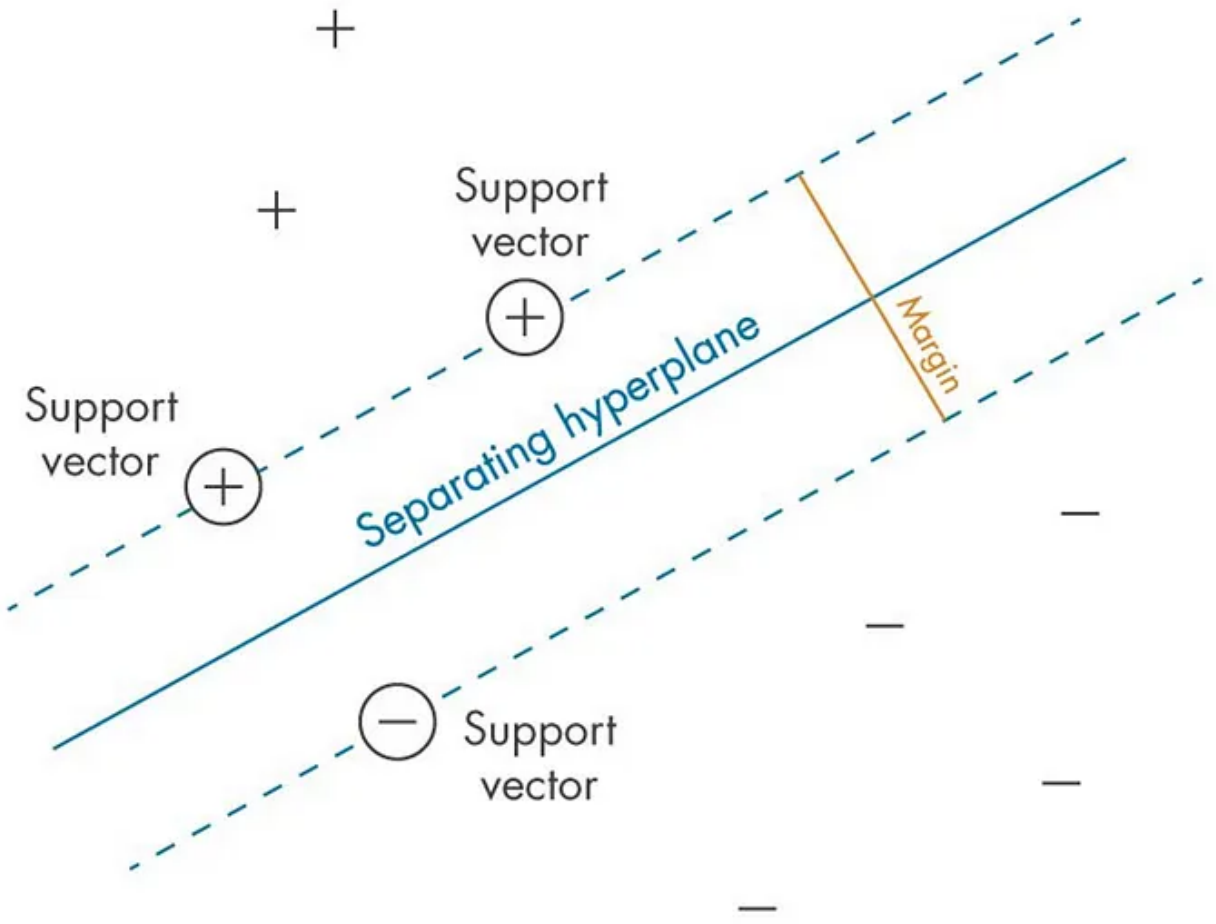
It can be used to make predictions on new data points by evaluating which side of the hyperplane they fall on. Data points on one side of the hyperplane are classified as belonging to one class, while data points on the other side of the hyperplane are classified as belonging to another class.



Margin

A **margin** is the distance between the decision boundary (hyperplane) and the closest data points from each class. The goal of SVMs is to maximize this margin while minimizing classification errors. A larger margin indicates a greater degree of confidence in the classification, as it means that there is a larger gap between the decision boundary and the closest data points from each class. The margin is a

measure of how well-separated the classes are in feature space. SVMs are designed to find the hyperplane that maximizes this margin, which is why they are sometimes referred to as maximum-margin classifiers.



Support Vectors

They are the data points that lie closest to the decision boundary (hyperplane) in a Support Vector Machine (SVM). These data points are important because they determine the position and orientation of the hyperplane, and thus have a significant impact on the classification accuracy of the SVM. In fact, SVMs are named after these support vectors because they "***support***" or define the decision boundary. The support vectors are used to calculate the margin, which is the distance between the hyperplane and the closest data points from each class. The goal of SVMs is to maximize this margin while minimizing classification errors.

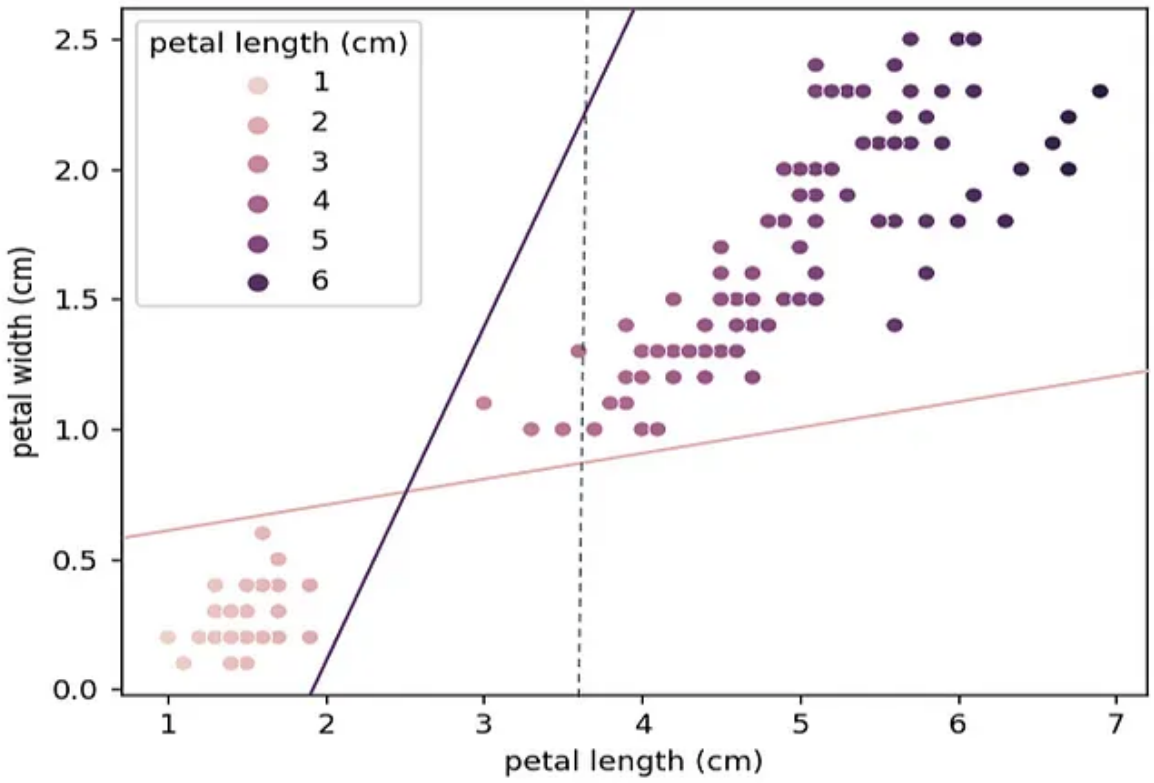
Understanding SVM with an example dataset

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Iris Dataset from the Scikit-learn library in Python.

We have a famous dataset called '**Iris**'. There are four features *(columns or independent variables)* in this dataset but for simplicity purposes, we shall on look at two which are: '*Petal length*' and '*Petal Width*'. These points are then plotted on a 2D plane.



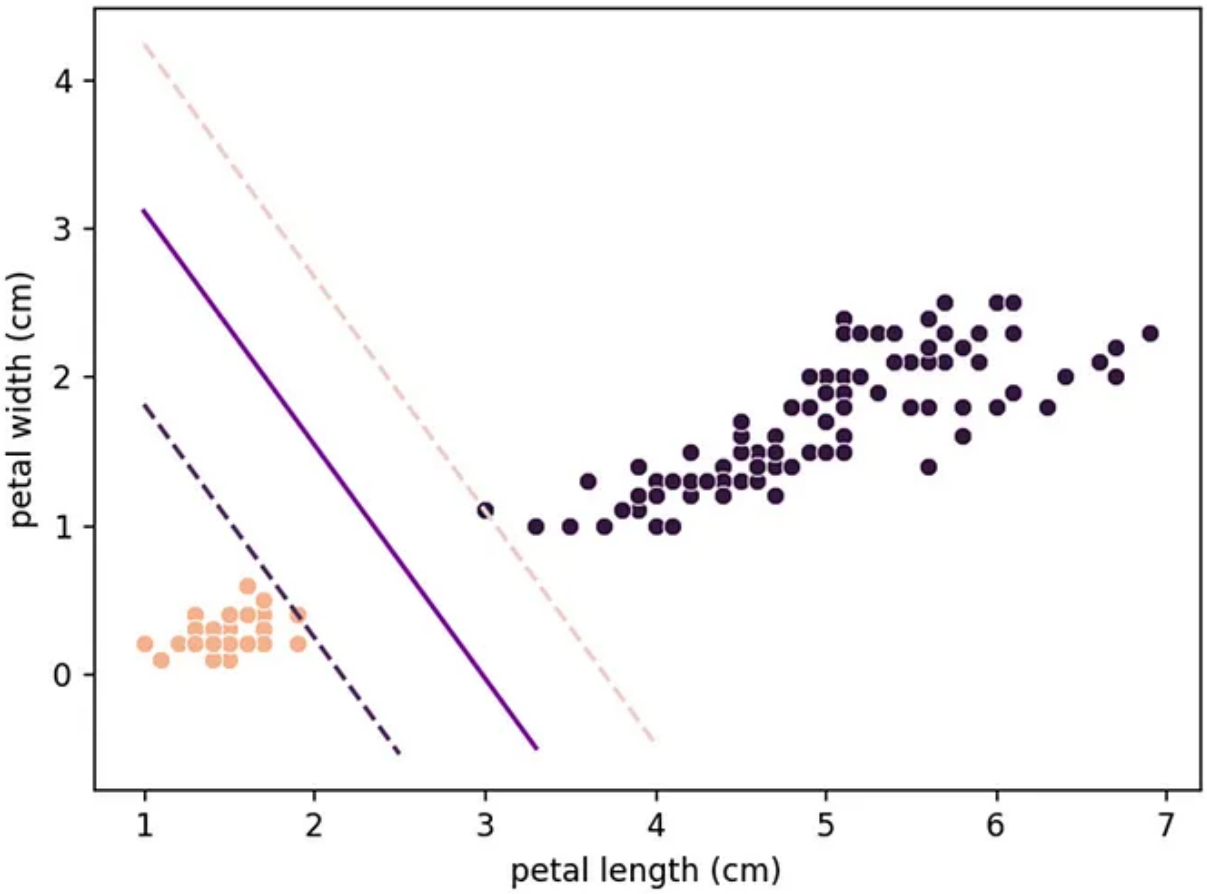
Iris dataset separated by three lines (linear classifiers) represent by dark, light and dotted lines.

Lighter points represent the species '*Iris Setosa*' and darker ones represent '*Iris versicolor*'.

We can simply classify this by plotting lines, using linear classifiers.

The dark and light lines accurately classify the test data set but may fail on new data due to the closeness of the boundary from the respective classes. Whereas, the dotted line classifier is entirely trash and misclassified many points.

What we want is the best classifier. A classifier which stays farthest from the overall class, that is where SVM comes in.



Iris dataset separated by a hyperplane obtained by an SVM model.

We can think of SVM as fitting the widest possible path *(represented by parallel dashed lines)* between the classes.

This is termed "**Large Margin Classification**".

> ***Note.*** *In theory, the hyperplane is exactly between the support vectors. But here it is slightly closer to the dark class. Why? This will be discussed later in the regularization part.*

Understanding by an Analogy (You can skip if you understood :)

You can think of SVM as a construction company. The 2D plane is a map and the 2 classes are 2 cities. The data points are analogous to buildings. You are the government and your goal is to create the best highway to minimise traffic which passes through both the cities, but you are constrained by the area available to you.

We are considering the road to be "straight" for now. (We will explore non-linear models later in the article)

You give the contract to SVM construction company. What SVM does to minimise traffic is it wants to maximise the width of the road. It looks at the widest stretch of land between the 2 cities. The buildings at the end of the road are called “Support Vectors” since they are constraining or “Supporting” the model. The highway is angled such that there is equal space for the cities to expand along it.

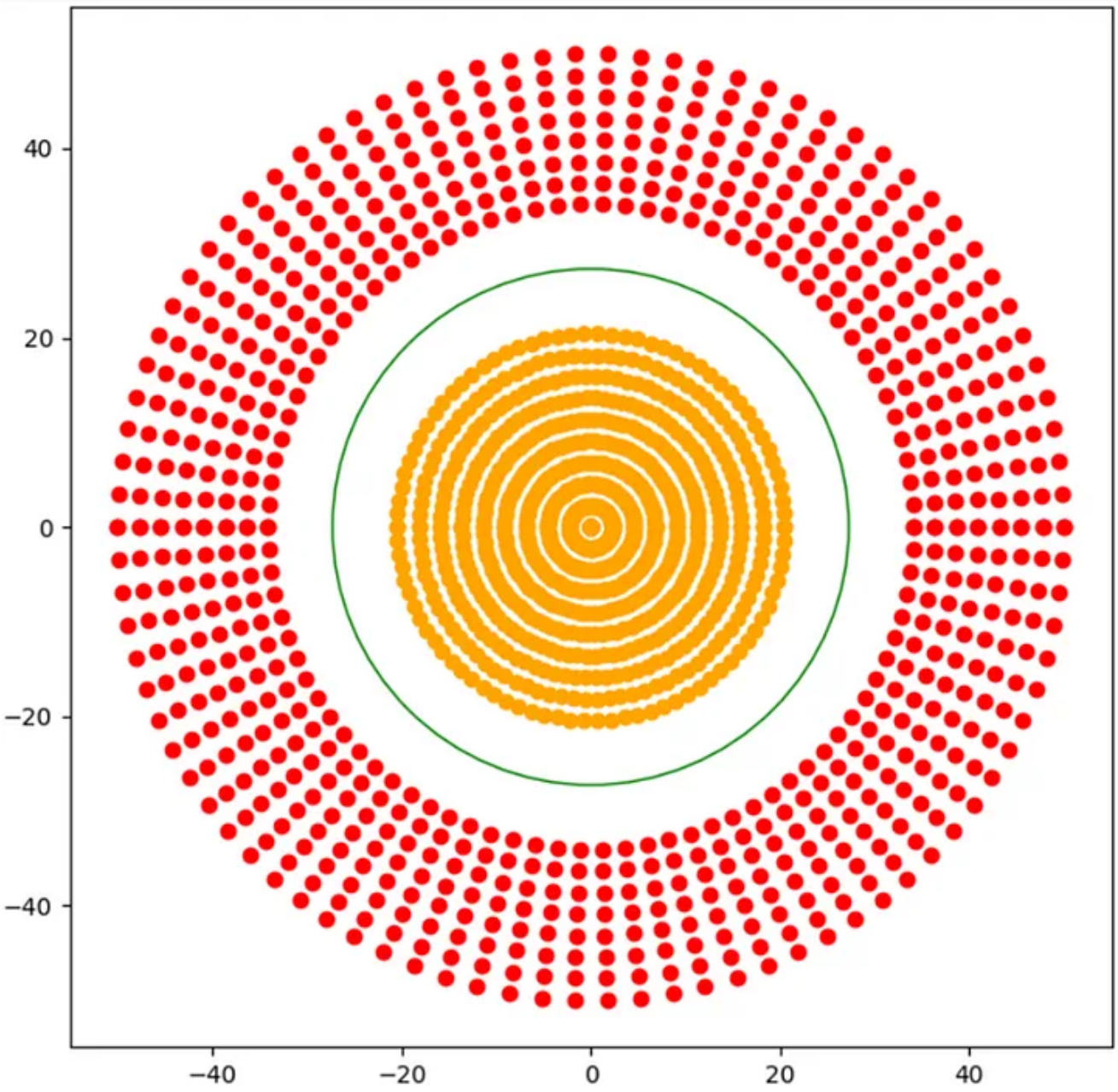
This central line dividing the highway represents the Decision Boundary (Hyperplane) and the edges represent Hyperplanes for the respective Classes. The width of this highway is the Margin.

What Happens if the data is not linearly classifiable?

When a linear hyperplane is not possible, the input data is transformed into a higher-dimensional feature space, where it may be easier to find a linear decision boundary that separates the classes.

What do I mean by that 😕 ?

Let’s look at an example:



A dataset which is not linearly separable.

In the above figure, a 2-D hyperplane was not possible and hence transformation was required *(remember I told you the case if the highway wasn't straight)*.

What is transformation or addition of a new feature?

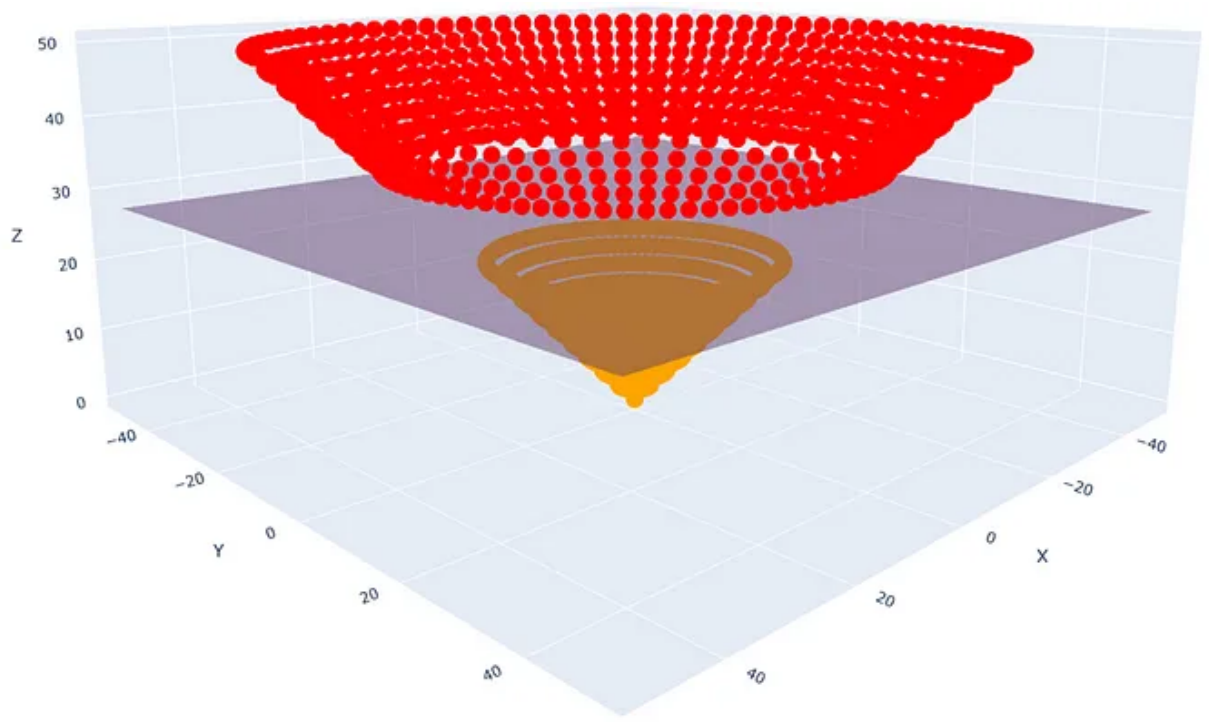
We have two features X and Y, and the data which not linearly classifiable. What we need to do is add another dimension in which if the data is plotted it becomes linearly separable.

Values of a point in the dimensions are nothing but column values of the point. To add another dimension we have to create another column *(or feature)*.

Here we have two features X and Y, a third feature is needed which will be a function of the original features i.e. X and Y, which will be enough to classify the data linearly in three dimensions.

We take the third feature $Z = f(x,y)$; f representing a function on X and Y. Here the Radial Basis Function(RBF) (measuring Euclidean distance) from the origin is enough.

> $Z = (X^2 + Y^2)^{(1/2)}$



The points, plotted in 3-D with a hyperplane dividing them

Here the hyperplane was as simple as creating a plane parallel to the X-Y plane at a certain height.

Problems with this method

The main problem here is the heavy load of the calculations to be performed.

Here we took points centred on origin in a concentric manner. Suppose the points were not concentric but could be separated by the RBF. So we would need to take each point in the dataset as a reference each time and find the distance of all other points with respect to that point.

So we would need to calculate **n*(n-1)/2** distances. *(n-1 other points with respect to each n points, but once the distance of 1–2 is calculated the distance of 2–1 does need to be calculated)*

Time Complexity

The time complexity of a square root is ***O(log(n))*** and for power, addition is ***O(1)***. Thus to do **n*(n-1)/2** total calculations we would need ***O(n²log(n))*** time complexity.

But as our goal is to separate the classes and not find the distance, we can do away with the square root. *($Z = X^2 + Y^2$)*

In that case, we would get a time complexity of ***$O(n^2)$***.

But this was not even the problem, the problem starts now

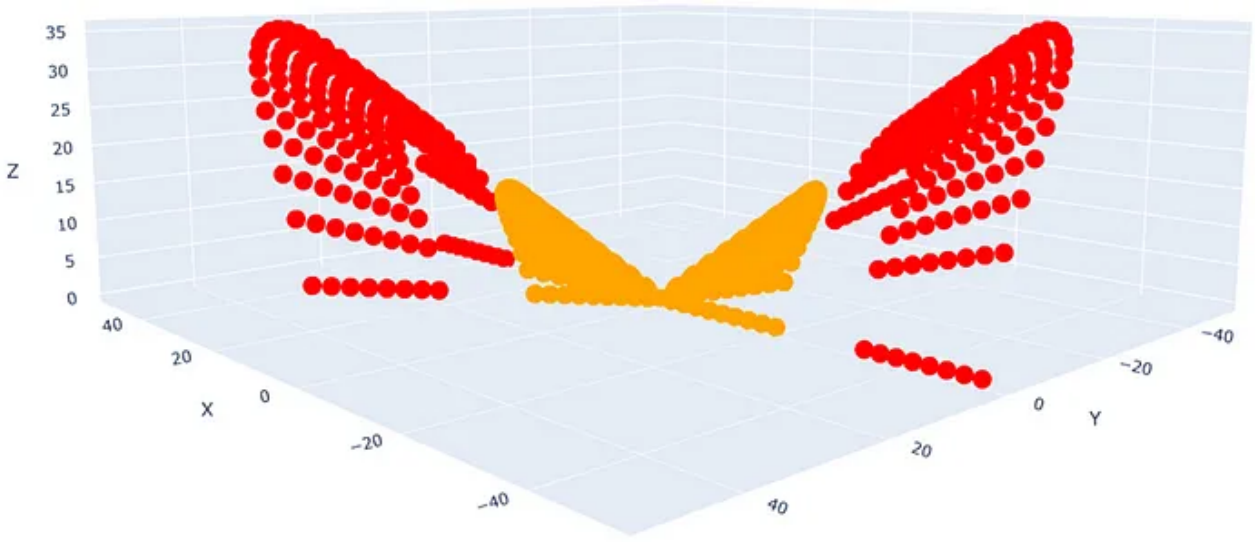
Here we knew which function was to be used. But there could be many functions with just the degree limited to 2 *(X, Y, XY, X^2 and Y^2)*.

We can use these 5 as 3 dimensions in ${}^5C_3$ ways = 10 ways. Not to mention, the infinite possibility of their linear combinations *($Z = 4X^2 + 7XY + 13Y^2$, $Z = 8XY + 17X^2$, and so on…)*.

And this was only for 2-degree polynomials. If we started using 3-degree polynomials then X^3, Y^3, X^2Y, XY^2 will also come in the picture.

Not all of these are good enough to be our additional feature.

For example, I started with (X vs Y vs XY as the features):

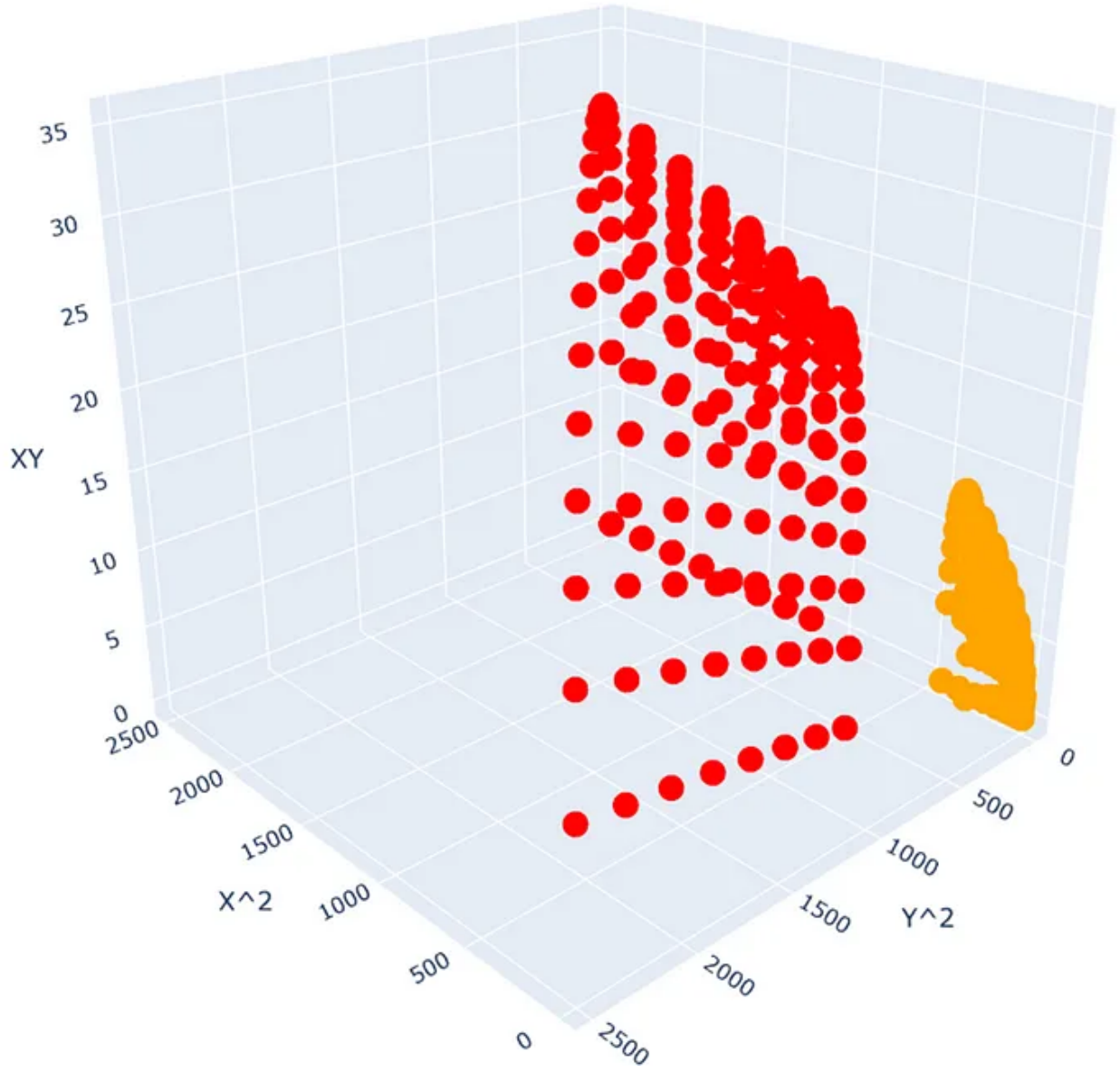


X vs Y vs XY plot of the same data, this plot is not linearly classifiable [Doesn't the figure look like two birds that have touched their beaks].

All the calculations and computations that went into this plot were in vain.

Now we have to use another function as the feature and try again.

Say, I use (X^2+vs Y^2 vs XY as the features, yes I replaced X and Y):



X² vs Y² vs XY plot of the same data, this plot is linearly classifiable [Doesn't the figure look like a bird with its beak].

I saw that earlier data and noticed that it wasn't linearly separable since yellow was in between the red points.

Since the two yellow beaks met at the centre and one of them was going in the negative X and negative Y direction, I decided to square X and Y so that a new set of values begin from 0 forming only a single separation region between the beak and the bird's face as compared to two earlier.

This plot is linearly separable, in this way, we can reuse the XY calculations and plot smartly to get the desired features to separate the data.

But even this has limitations, such as using only one or two feature datasets so we get a plot in 3-d and not more dimensions, also our brain's capacity to look for

patterns to identify the next set of features and what if the first plot had no pattern so we had to again take a guess for a feature and start from scratch.

If we got the desired feature set in only two steps as we got above, even then this method is slower than the one we actually use.

What we use is called the **Kernel Trick**.

The Kernel Trick

A **Kernel Trick** instead of adding another dimension/feature, finds the similarity of the points.

Instead of finding **f(x,y)** directly, it computes the similarity of the image of these points. In other words, instead of finding **f(x1,y1)** and **f(x2,y2)** we take the points *(x1,y1)* and *(x2,y2)* and compute how similar would their outputs be using a function **f(x,y)**; where f can be any function on *x,y*.

Thus we don't need to find a suitable set of features here. We find similarity in such a way that it is valid for all sets of features.

To calculate the similarity we use the **Gaussian** function.

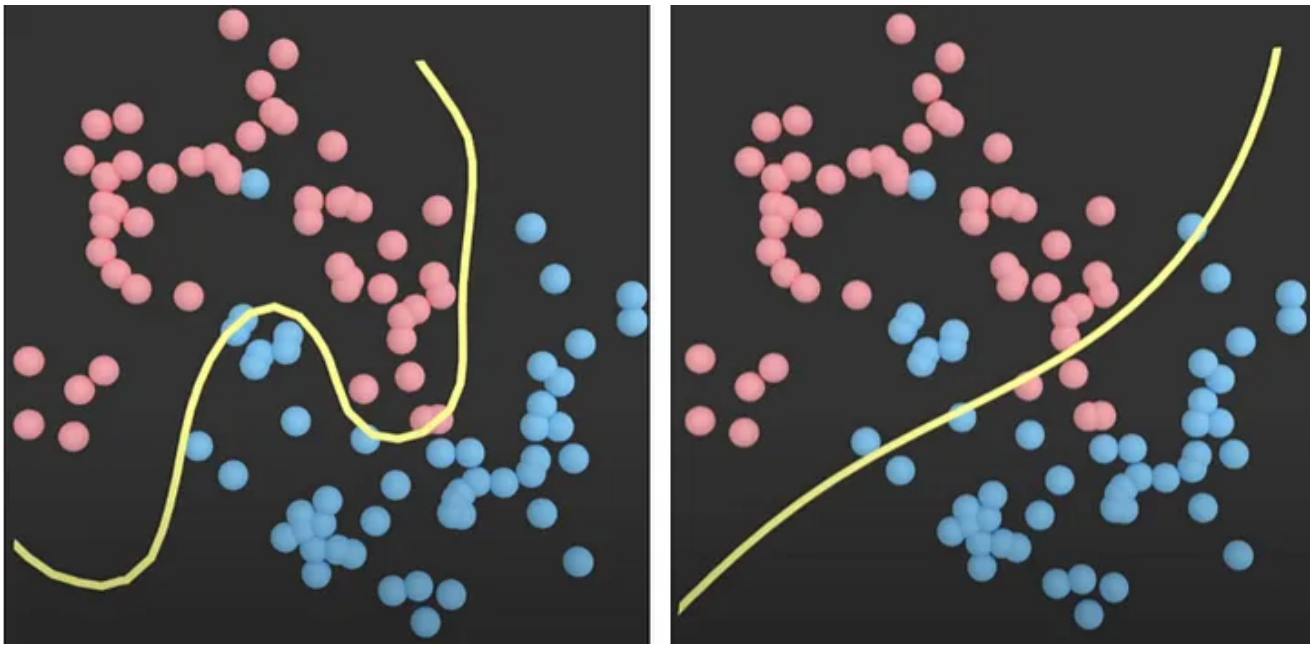
> $f(x) = ae^{(-(x-b)^2/2c^2)}$
>
> a : represents the height of the peak of the curve
>
> b : represents the position of the centre of the peak
>
> c : is the standard deviation of the data

For RBF Kernel we use:

> $K(X,X') = e^{-\gamma(|X-X'|^2)} = (1/ e^{|X-X'|^2})^{\gamma}$
>
> γ : is a hyperparameter which represents the linearlity of the model ($\gamma \propto 1/c^2$)
>
> X,X' :represents the position vectors of the two points

A small γ *(tending to 0)* means a more linear model and a large γ means a more non-linear model.

Here we have 2 models (*left with γ = 1 and right with γ = 0.01*, much more linear in nature).

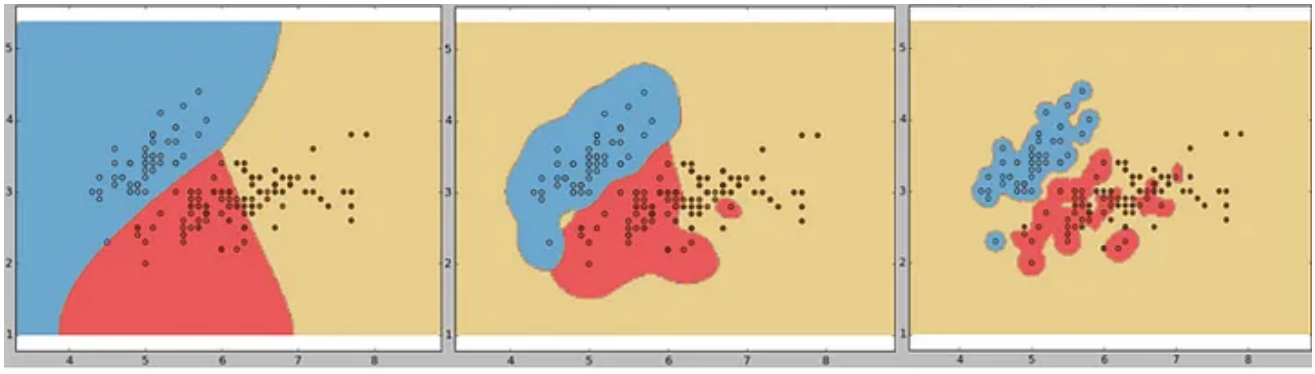


Two Models with Different Gamma Values.

So why not let gamma be a very high value? What is the use of a low γ?

Large values of gamma may lead to overfitting as well and thus we need to find appropriate gamma values.

Figure with 3 models, from left *γ = 0.1, γ = 10, γ = 100*. (The left one is accurately fitted, the middle is overfitted and the right one is extremely overfitted)



Three Models with Different Gamma Values.

Time Complexity

Since we need to find the similarity of each point with respect to all other points we need a total of **n*(n-1)/2** calculations.

Exponent has a time complexity of ***O(1)*** and thus we get a total time complexity of ***O(n²)***.

We do not need to plot points, check feature sets, take combinations, etc. This makes this method way more efficient.

What we do have here is the usage of different **Kernels** for this.

We have *Kernels* such as:

> *Polynomial Kernel*
>
> *Gaussian Kernel*
>
> *Gaussian RBF Kernel*
>
> *Laplace RBF Kernel*
>
> *Hyperbolic Tangent Kernel*
>
> *Sigmoid Kernel*
>
> *Bessel function of first kind Kernel*
>
> *ANOVA radial basis Kernel*
>
> *Linear Splines Kernel*

Whichever one we use, we get results in lesser computations than the original method.

To further optimise our calculations we use the "***Gram Matrix***".

The Gram Matrix is a matrix which can be easily stored and manipulated in the memory and is highly efficient to use.

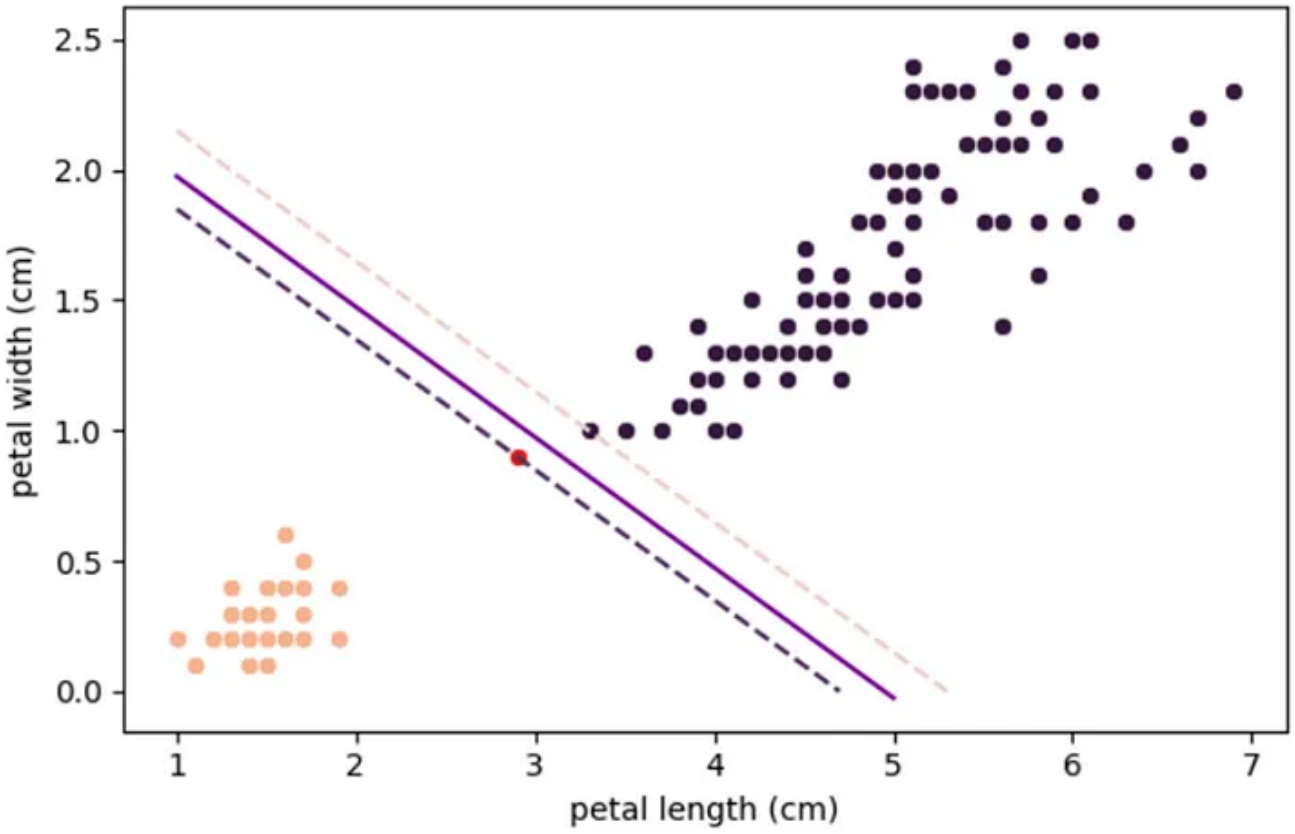
Regularization and Soft Margin SVM

Finally, Onto a new topic 😮💨*(phew)*.

If we strictly impose that all points must be off the street on the correct side then this is called **Hard Margin Classification** *(remember the first SVM model figure that I showed)*.

There are two issues with this method. First, it would only work with linearly separable data and not for non-linearly separable data *(which may be linearly classifiable for the most part).*

Second, is its sensitivity to outliers. In the figure below the **Red Point** is introduced as an outlier in the left class and it significantly changes the decision boundary, this may result in misclassifications of non-outlier data of the second class while testing the model.

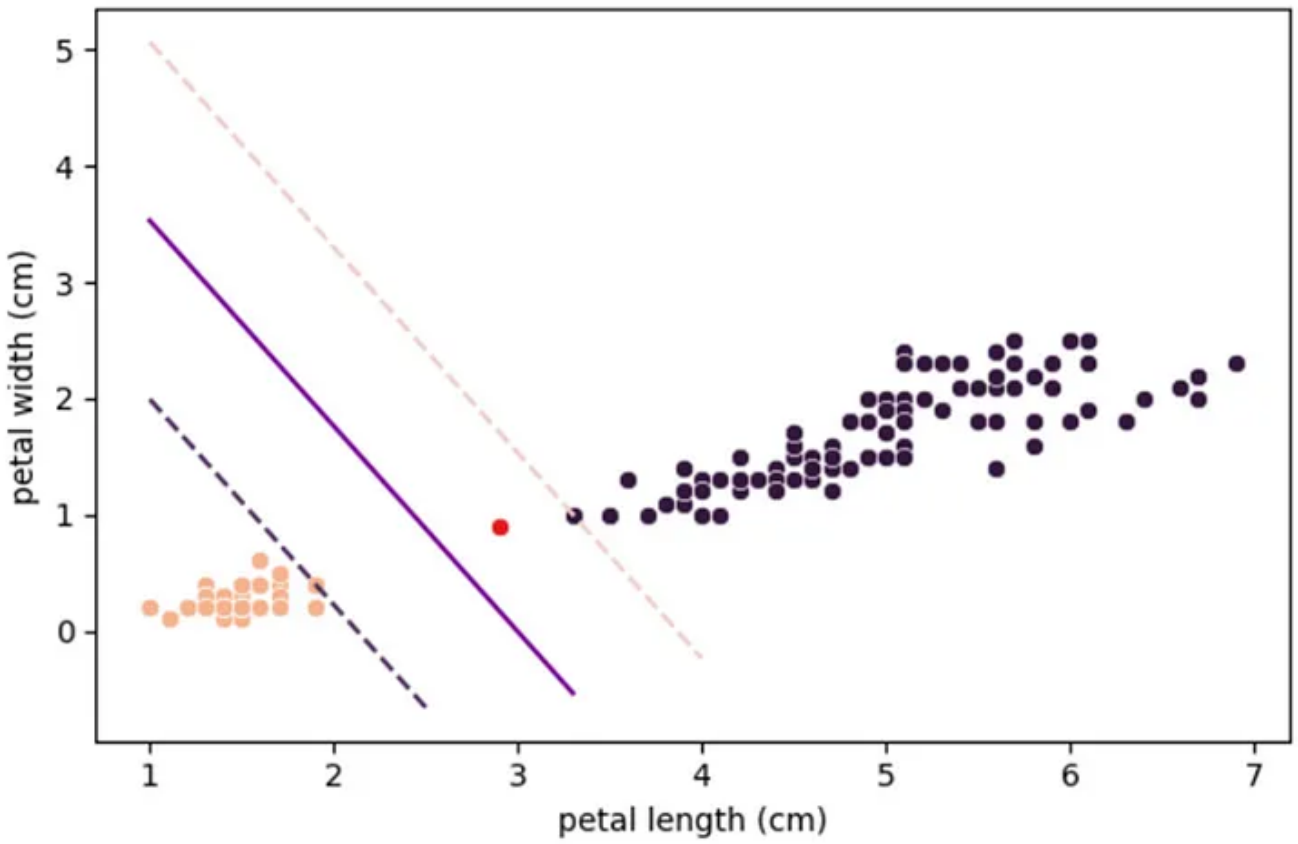


The “Iris” dataset with an outlier completely changes the decision hyperplane.

Although this model has not misclassified any of the points it is not a very good model and will give higher errors during testing.

To avoid this, we use **Soft Margin Classification**.

A **soft margin** is a type of margin that allows for some misclassification errors in the training data.



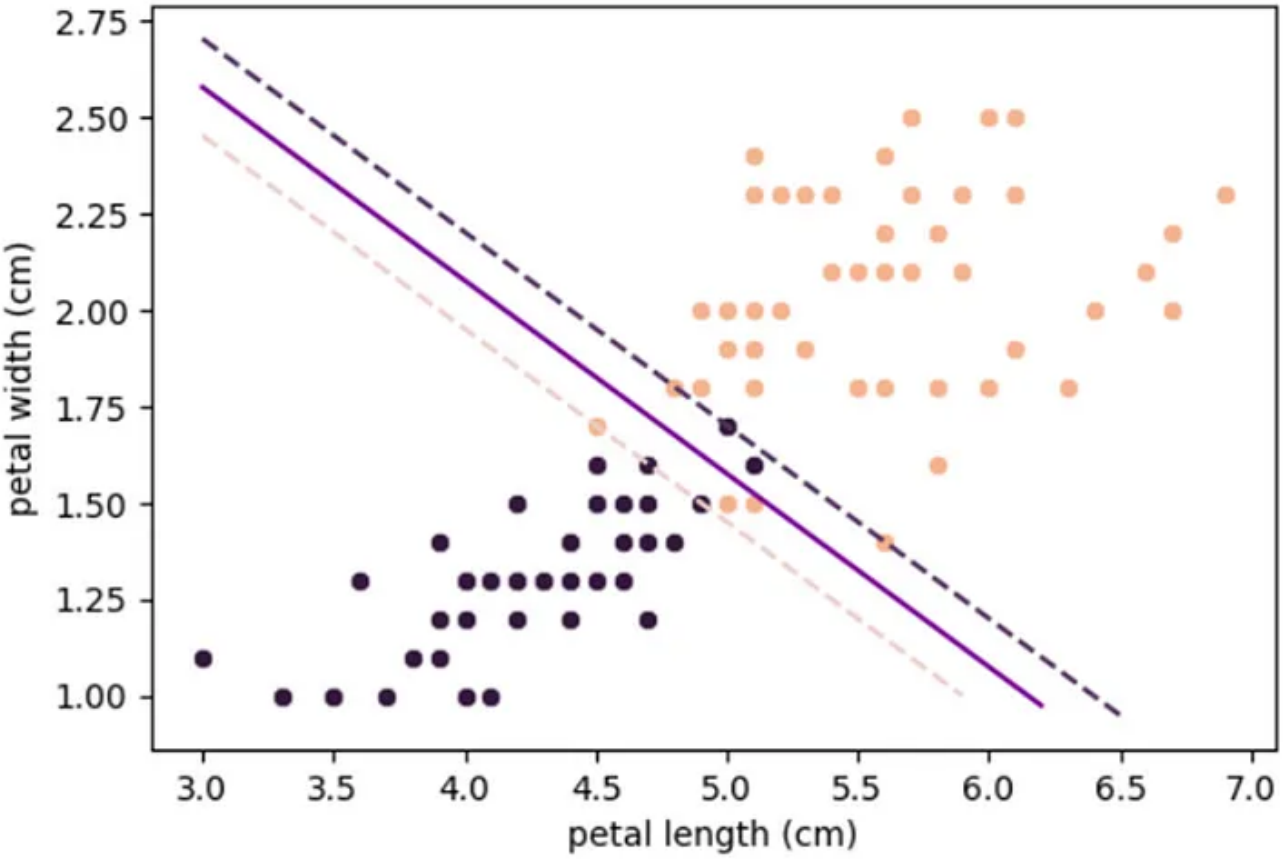
"Iris" dataset with the outlier but with Soft Margin Classification.

Here, a soft margin allows for some misclassification errors by allowing some data points to be on the wrong side of the decision boundary.

Even though there is a misclassification in the training data set and worse performance with respect to the previous model, the general performance would be much better during testing, as a result of how far it is from both classes.

But we can solve the problem of outliers by **removing** them using *data preprocessing* and *data cleaning* right? Then why Soft Margins?

They are used **mainly** when the data is *not linearly separable*, meaning that it is not possible to find a hyperplane that perfectly separates the classes without any errors and to avoid outliers ***(Hard Margin Classification** is not possible)*. Example :



"Iris" dataset (Right Class is Iris Virginica, Left Class is Iris Versicolor), is non-linearly separable but when Soft Margin Classification is used, we get a model with minimum misclassification. (0 minor misclassifications with respect to hyperplanes of respective classes) [5 major misclassifications, wrong side of decision boundary] (Value of C = 100).

How are Soft Margins working?

Soft margins are implemented by introducing a *slack variable* for each data point, which allows the SVM to tolerate some *degree of misclassification* error. The amount of tolerance is controlled by a parameter called the **regularization hyperparameter C**, which determines how much weight should be given to minimizing classification errors versus maximizing the margin.

It controls how much *tolerance* is allowed for misclassification errors, with larger values of C leading to a harder margin *(less tolerance for errors)* and smaller values of C leading to a softer margin *(more tolerance for errors)*.

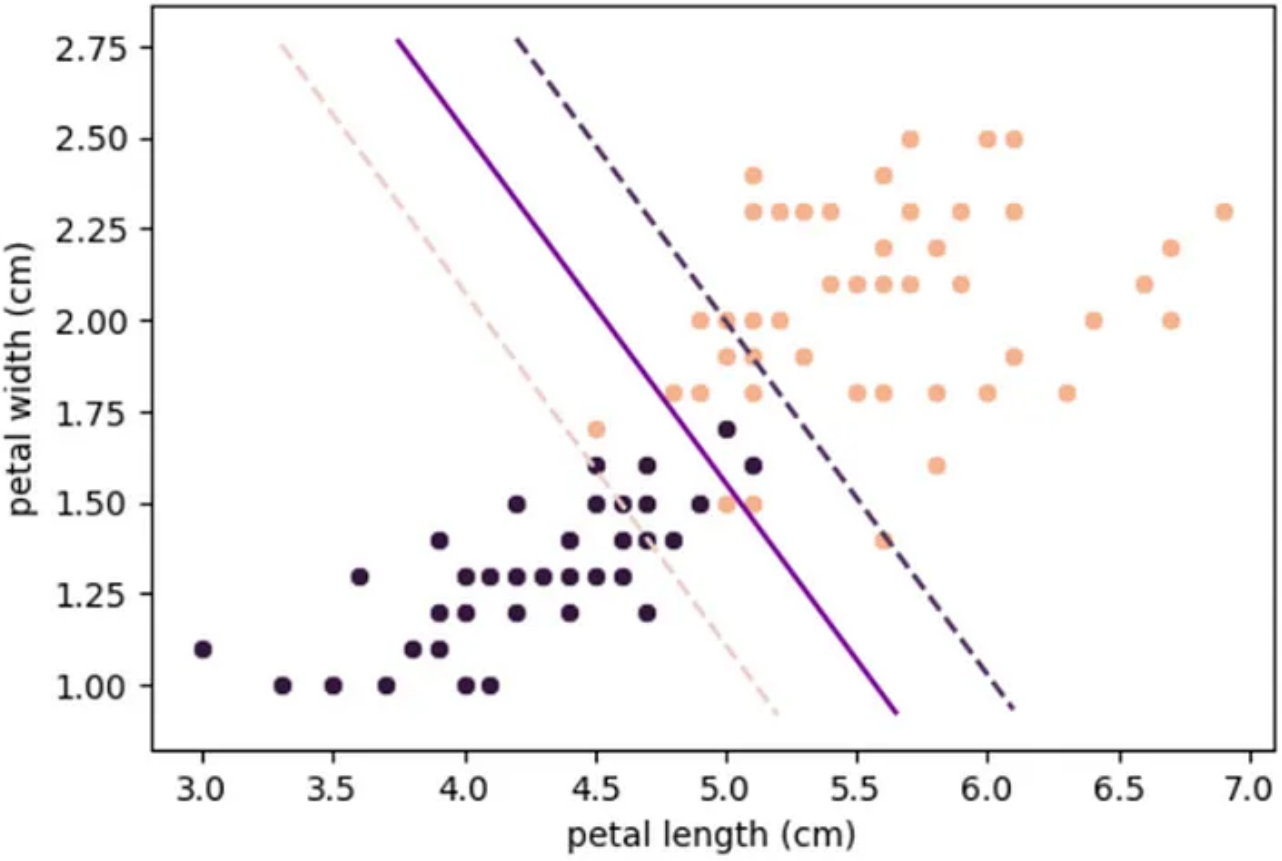
Basically through our analogy, instead of creating a very small road *(for the outlier case)* when it was not possible to create a large road through the middle of the two cities we create a larger road by moving out some people.

It may be bad for the people moving out *(the outlier getting misclassified)* but overall the highway *(our model)* would be way larger *(more accurate)* and better.

In the case above where no road could be created, we ask some people to move out and create a narrow road. A wider road though better for transport would cause problems for a large number of people *(many points getting misclassified)*.

The regularization hyperparameter 'C' controls how many people can be moved out *(how many points can be misclassified or tolerance)* for the construction of the project.

A high value of C means the model is harder in nature (less tolerant to misclassifications). Whereas a low value of C means that the model is softer in nature (more tolerant to misclassifications).



Same Model with the value of C = 1 (12 minor misclassifications with respect to hyperplane of the classes) [4 major misclassifications, wrong side of decision boundary].

A lower C value for the previous model *(1 with respect to 100)* tolerates more misclassifications, allowing more people to move out and thus building a wider street).

> ***Note.*** *Lower value of C does not necessarily mean more major misclassifications always, sometimes it may mean way more minor classifications.*

In this case and for most general cases, low values of C tend to give trash models misclassifying multiple points and reducing accuracy.

> ***Note.*** *A low C is not simply just widening the original path until the required tolerance level is met. It means creating a new widest path by misclassifying the maximum number of points such that it is below the tolerance threshold.*

C controls the Bias/Variance trade-off. A low bias means that the model has low or no assumptions about the data. A high variance means that the model will change depending on what we take as training data.

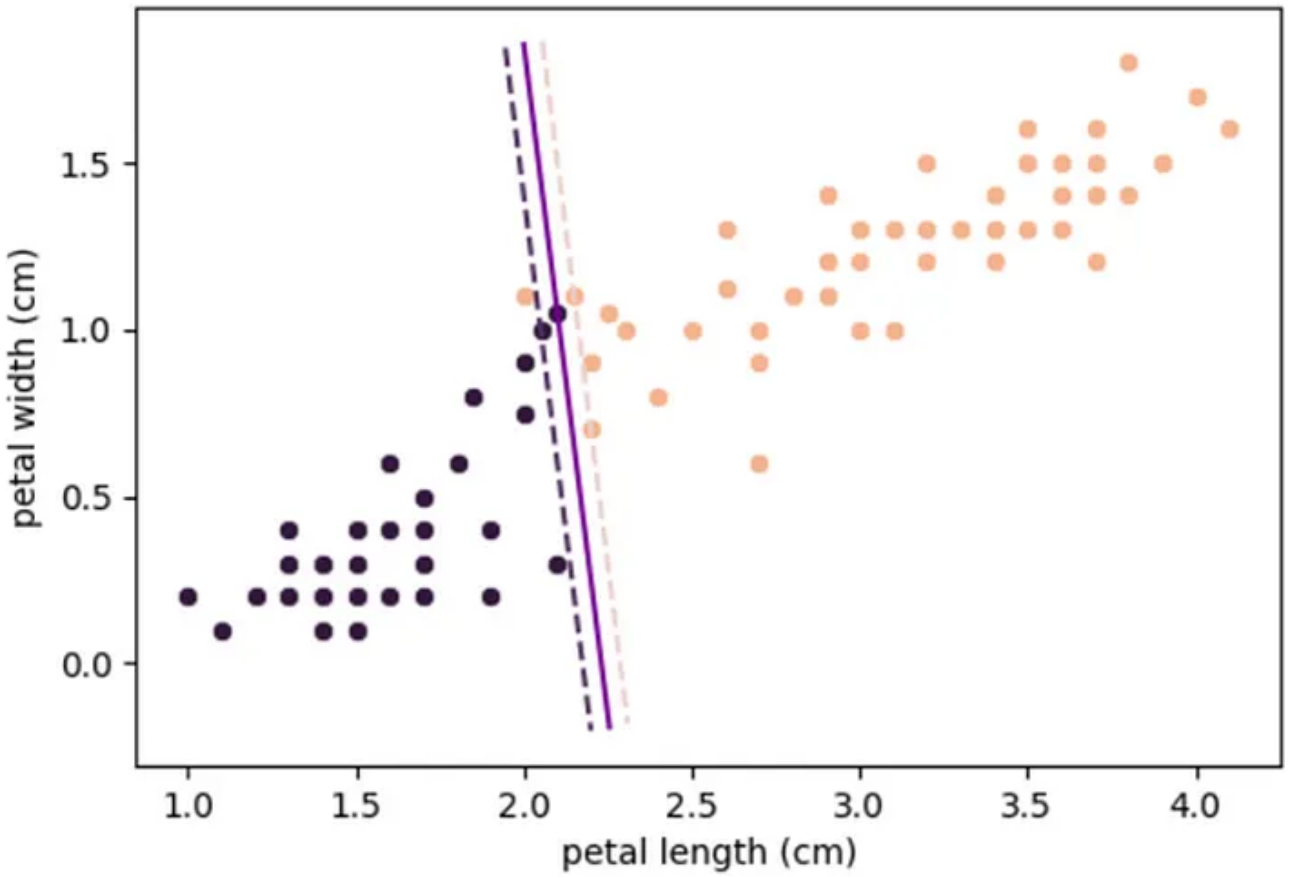
For Hard Margin Classification, the model changes significantly on changing data *(if new points were introduced between the hyperplanes)* so it has high variance, but it has no assumption about the data there is low bias.

Soft Margin Classification models have negligible changes *(due to tolerance to misclassify data)* thus they have a low variance. But it assumes we can misclassify some information and assumed that the particular model with a wider margin would lead to better results and thus have a high bias.

Why use low value of C then? Why an appropriate value needs to be chosen

This is a phenomenon similar to overfitting and underfitting which happens with very high values of C and very low values of C.

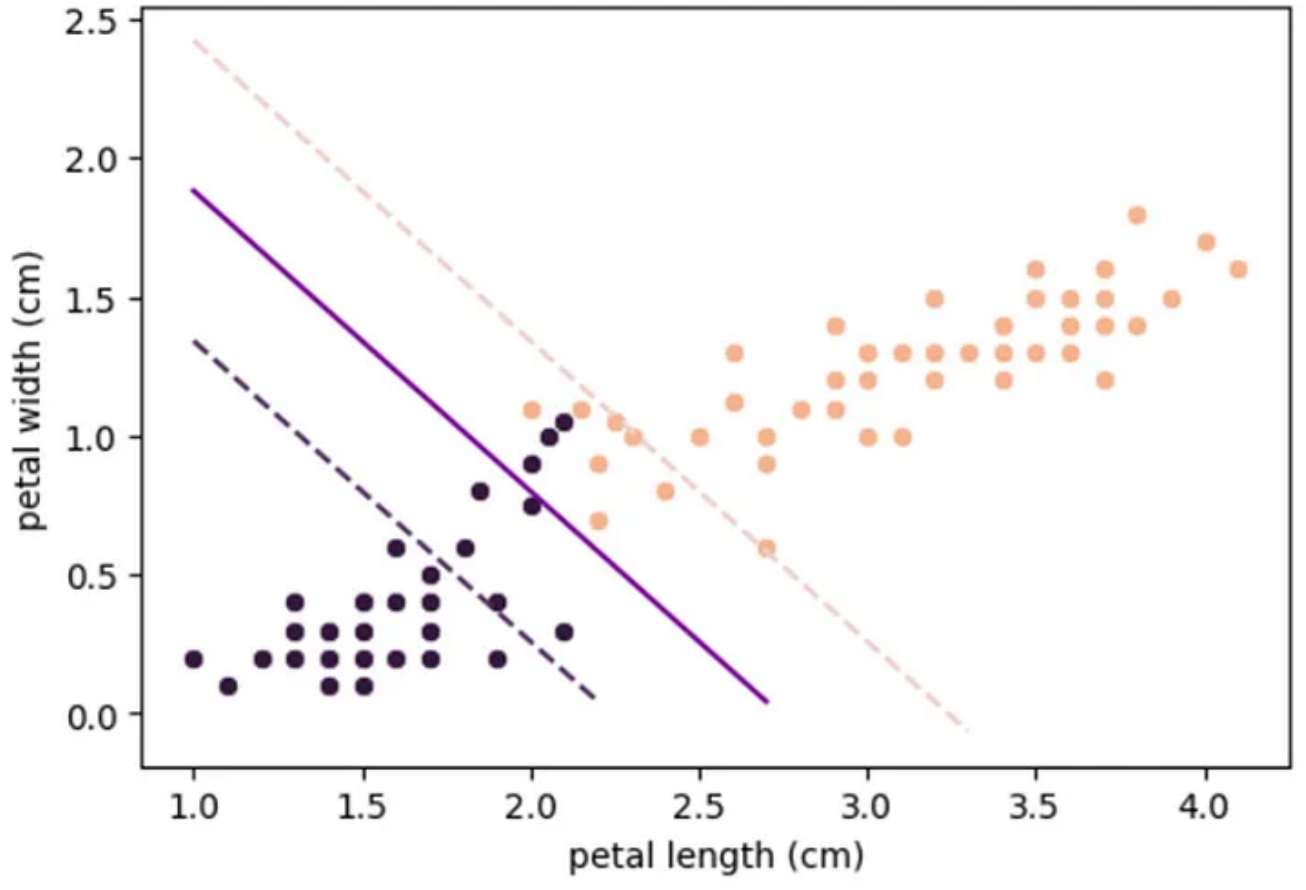
Very low values will give very poor results as seen *(similar to the case of underfitting)*.



Modified dataset to show the phenomenon.

Model with C = 1000, is unsuitable as it is too close to the left class at the bottom and too close to the right class at the top, with chances of misclassifying data *(here there is only 1 major misclassification and 1 minor misclassification hence, during training the model is good, but the model is not good for general decision making and may perform poorly during testing).*

Thus models with a very high value of C may also give poor results on testing (*similar to the case of overfitting).*



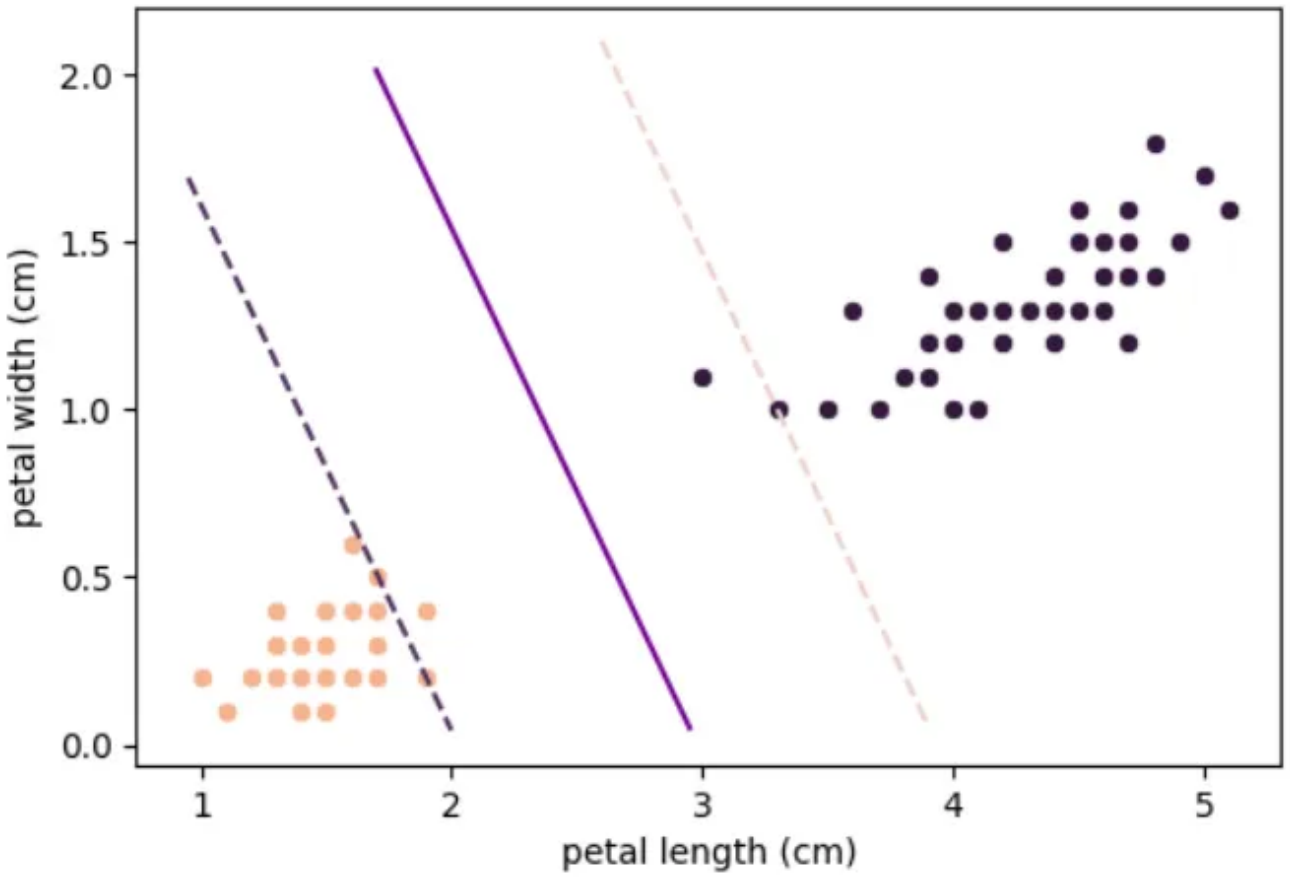
Modified “Iris” dataset used, but here with C = 1.

Model with C = 1, suitable and better-generalised model. (*Though there are 3 major misclassifications and about 12 minor misclassifications and thus a worse performance on training data but the model keeps in mind the bulk of the data and creates decision boundaries according to that, hence it has better performance during testing owing to its distance from both the classes*).

Note: *Minor misclassification is a term which I use to describe data not correctly classified by the class’ hyperplane. They do not lead to worse performance directly but give an indicator that the model* ***may be*** *worse. Hence in the above case despite 15 misclassifications performance is not 7.5 times worse, but only 3 times worse on training data due to 3 times more major misclassifications.*

Remember I said initially about how in theory the decision boundary shall be in between the support vectors, but it was slightly closer to the darker class. That, was due to regularization. It created a model with 2 minor misclassifications such that the overall model is a more accurate one.

And thus the model should have been represented like this:

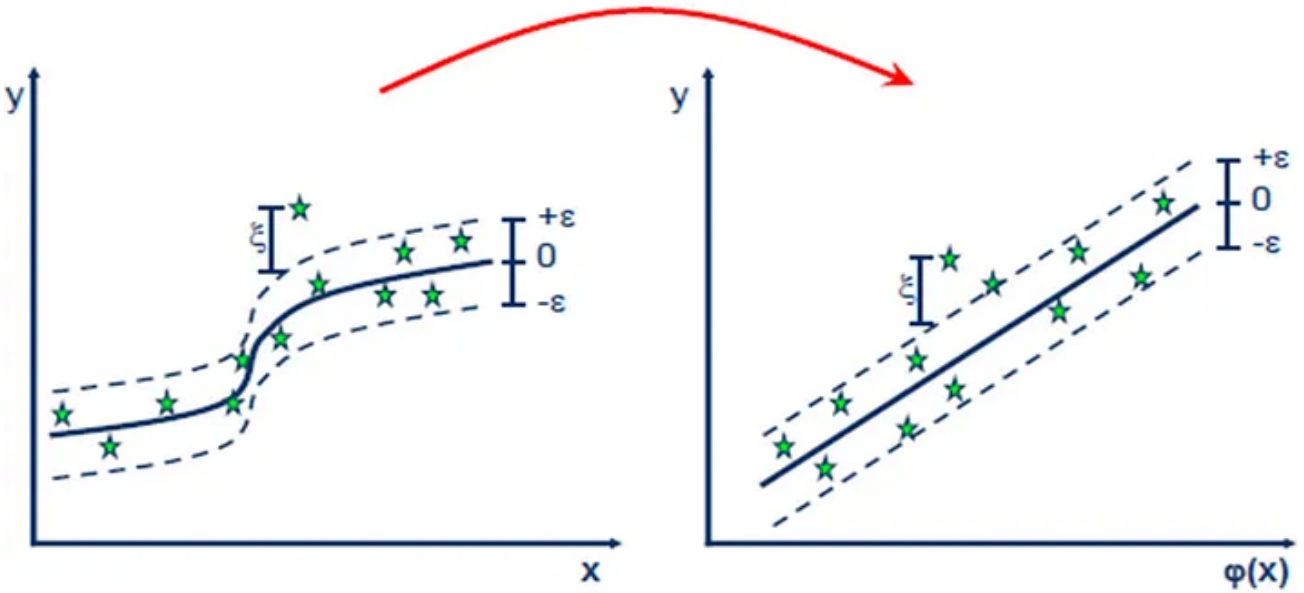


Corrected version of the first model, with 2 minor misclassifications (Decision Boundary is now equidistant from the support vectors).

Regression using SVM

SVMs, although generally used for classification can be used for both regression and classification. Support Vector Regression (SVR) is a machine learning algorithm used for regression analysis. It is different from traditional linear regression methods as it finds a hyperplane that best fits the data points in a continuous space, instead of fitting a line to the data points.

SVR in contrast to SVM tries to maximise the number of points in the street (margin), the width is controlled by a hyperparameter ε (epsilon).



Support Vector Regression (SVR) the decision hyperplane is used to predict the value.

An analogy of this could be passing a flyover or a bridge over buildings or houses where we want to give shade to the most number of houses keeping the bridge thinnest.

SVR wants to include the whole data into its reach while trying to minimise the margin, basically trying to encompass the points. Whereas linear regression wants to pass a line such that the sum of distances of the points from the line is minimum.

The advantages of SVR over normal regression are:

1. **Non-linearity:** SVR can capture non-linear relationships between input features and the target variable. It achieves this by using the kernel trick. In contrast, Linear Regression assumes a linear relationship between the input features and the target variable and Non-Linear Regression would require a lot of computation.

2. **Robustness to outliers:** SVR is more robust to outliers compared to Linear Regression. SVR aims to minimize the errors within a certain margin around the predicted values, known as the epsilon-insensitive zone. This characteristic makes SVR less influenced by outliers that fall outside the margin, leading to more stable predictions.

3. **Sparsity of support vectors:** SVR typically relies on a subset of training instances called support vectors to construct the regression model. These support vectors have the most significant impact on the model and represent

the critical data points for determining the decision boundary. This sparsity property allows SVR to be more memory-efficient and computationally faster than Linear Regression, especially for large datasets. Also, an advantage is that after the addition of new training points the model does not change if they lie in the margin.

4. **Control over model complexity:** SVR provides control over model complexity through hyperparameters such as the regularization parameter C and the kernel parameters. By adjusting these parameters, you can control the trade-off between model complexity and generalization ability this level of flexibility is not offered by linear regression.

Applications and Uses of SVM

Support Vector Machines (SVMs) have been successfully applied to various real-world problems across different domains. Here are some notable applications of SVMs:

1. **Image Classification:** SVMs have been widely used for image object recognition, handwritten digit recognition and optical character recognition (OCR). They have been employed in systems like filtering image-based spam and face detection systems used for security, surveillance, and biometric identification.

2. **Text Classification:** SVMs are effective for text categorization tasks, such as sentiment analysis, spam detection, and topic classification.

3. **Bioinformatics:** SVMs have been applied in bioinformatics for tasks such as protein structure prediction, gene expression analysis, and DNA classification.

4. **Financial Forecasting:** SVMs have been used in financial applications for tasks such as stock market prediction, credit scoring, and fraud detection.

5. **Medical Diagnosis:** SVMs have been utilized in medical diagnosis and decision-making systems. They can assist in diagnosing diseases, predicting patient outcomes, or identifying abnormal patterns in medical images.

SVMs have also been applied in other domains such as geosciences, marketing, computer vision, and more, showcasing their versatility and effectiveness in

various problem domains.

. . .

Finally, *Phew.*

Takes a breath of relief.

This was it for the blog. I hope you got an Intuitive understanding of SVM and the topics explained, as said in the title of the Blog.

I have skipped several topics such as deep dive into the mathematics of SVM, multi-class classifications (as shown in one of the figures) and the various Kernel functions, and I just touched upon Support Vector Regression.

But I hope this gave you a basic understanding of SVMs.

I have attached the Google Collab Notebook here: SVM Collab Notebook

I am, but human. When an algorithm can misclassify data, I can make mistakes, especially this being my first blog. If you spot one, feel free to point it out 😊.

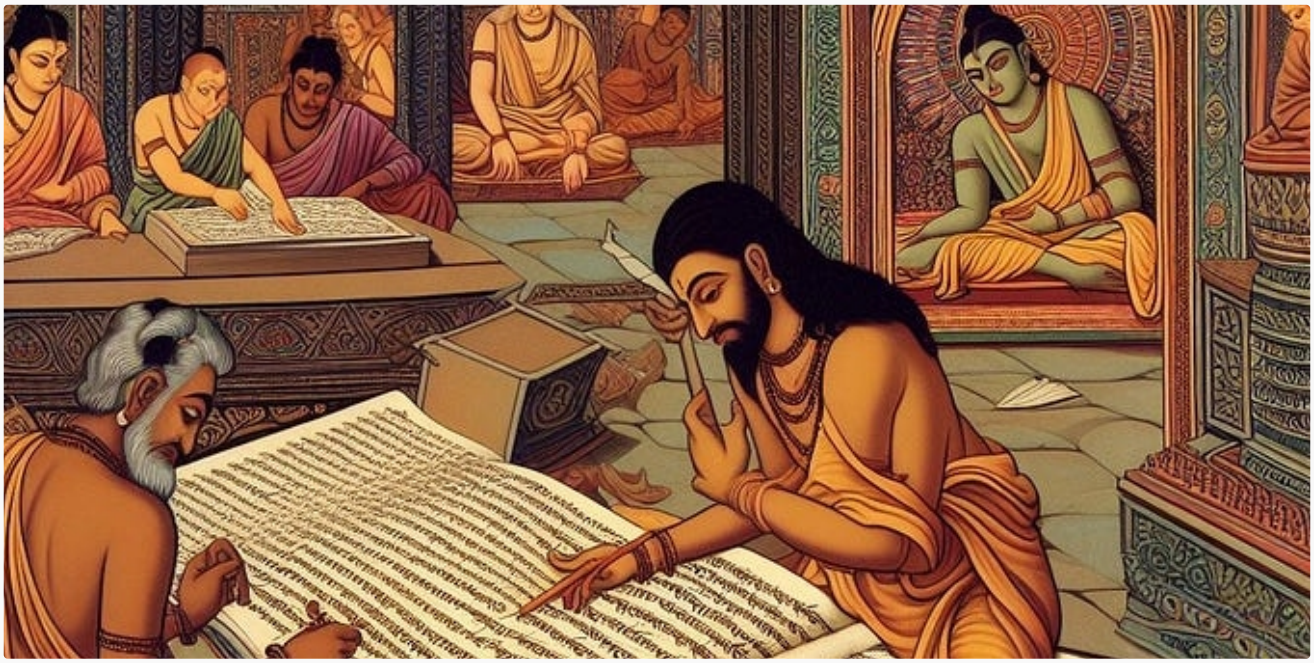
Approximately 4100 words later, I bid you adieu.

Support Vector Machine | Data Science | Machine Learning | Data Analysis | Data

Written by Tasmay Pankaj Tibrewal

87 Followers · Writer for Low Code for Data Science

More from Tasmay Pankaj Tibrewal and Low Code for Data Science



Tasmay Pankaj Tibrewal

An Introduction to Paninian Grammar

Preface: Panini and his work, was revolutionary in the field of linguistics. He not only gave an exhaustive description of Sanskrit but...

Oct 28, 2023 500

Rosaria Silipo in Low Code for Data Science

Is Data Science dead?

In the last six months I have heard this question thousands of time: “Is data science dead?”

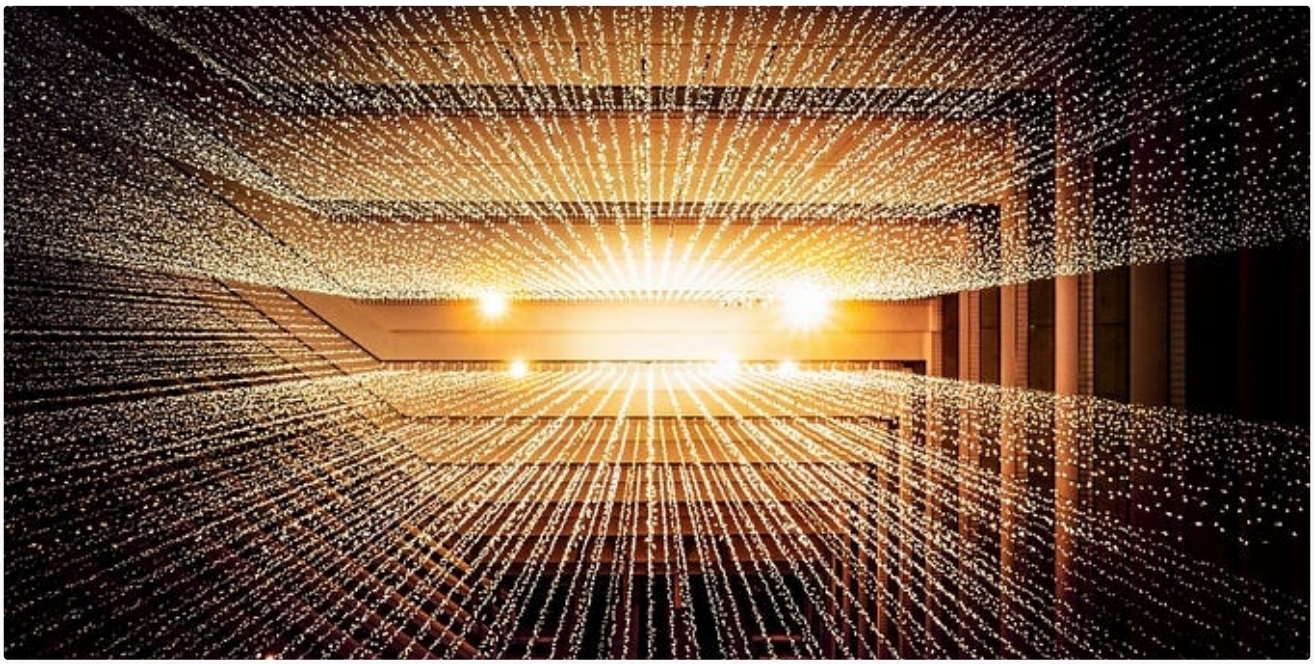
Mar 11 2.7K 66

Jamie Crossman-Smith in Low Code for Data Science

XGBoost Explained: A Beginner’s Guide

Understand how XGBoost works, when to use it, and its advantages over other algorithms

Mar 24 271 2



Rosaria Silipo in Low Code for Data Science

Is Data Science dead? The Rebuke

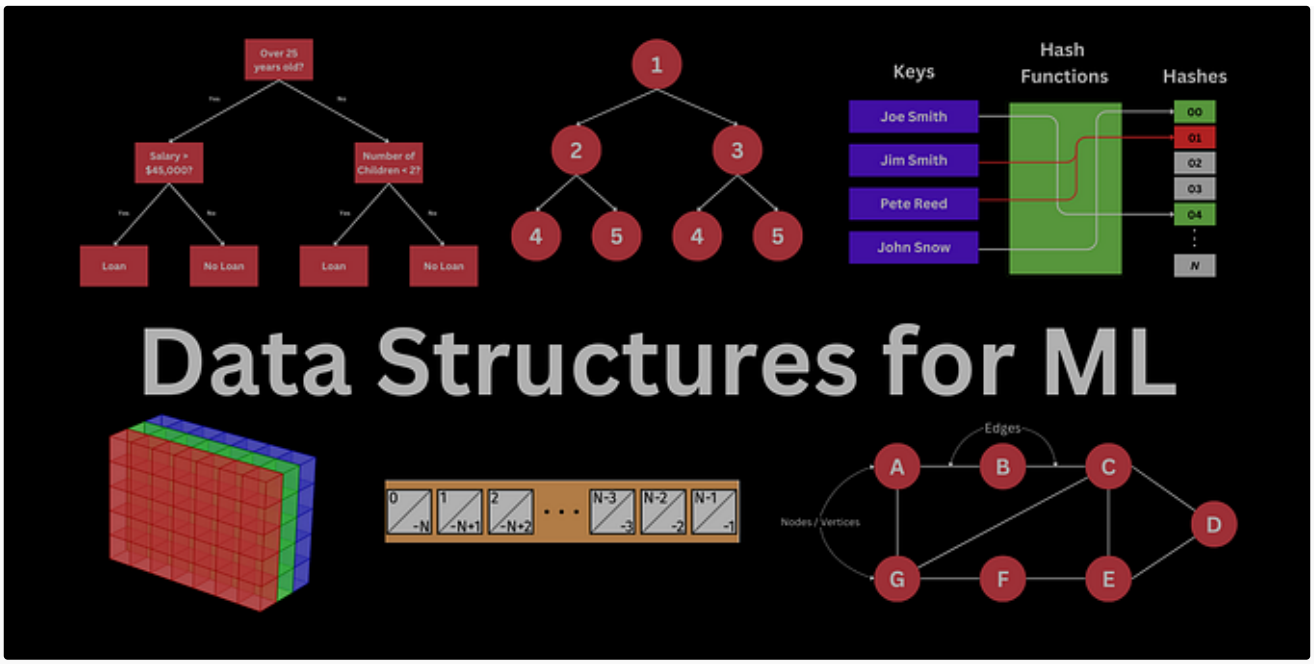
Addressing some of the hot topics in the reactions to “Is Data Science dead?”

Jun 3 218 2

See all from Tasmay Pankaj Tibrewal

See all from Low Code for Data Science

Recommended from Medium

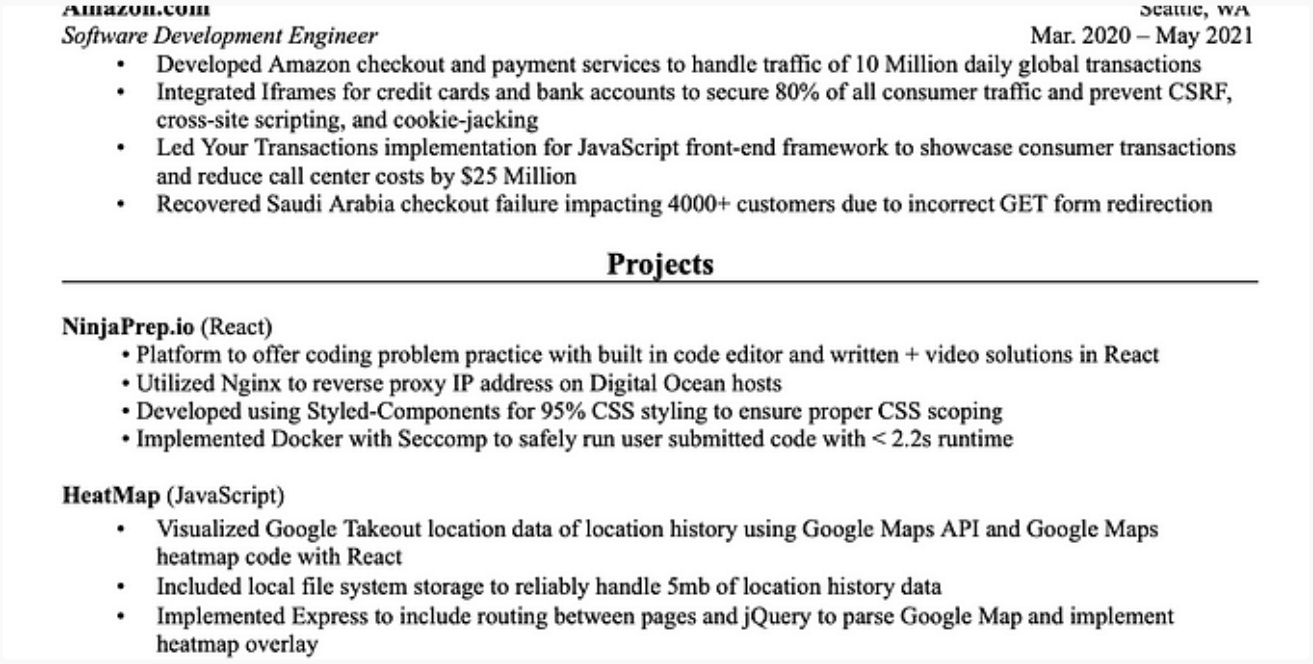


Joseph Robinson, Ph.D. in Towards AI

5 Data Structures That Machine Learning Engineers and Data Scientists Must Know

A Comprehensive Guide to Efficiency and Scalability

Sep 4 242 2



Amazon.com Seattle, WA

Software Development Engineer Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by $25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

Projects

NinjaPrep.io (React)

- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

HeatMap (JavaScript)

- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay

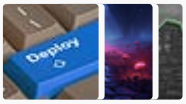
Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at Google.

1-page. Well-formatted.

Jun 1 20K 378

Lists



Predictive Modeling w/ Python

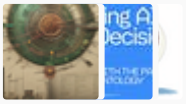
20 stories · 1503 saves

Practical Guides to Machine Learning

10 stories · 1832 saves

Natural Language Processing

1687 stories · 1258 saves



data science and AI

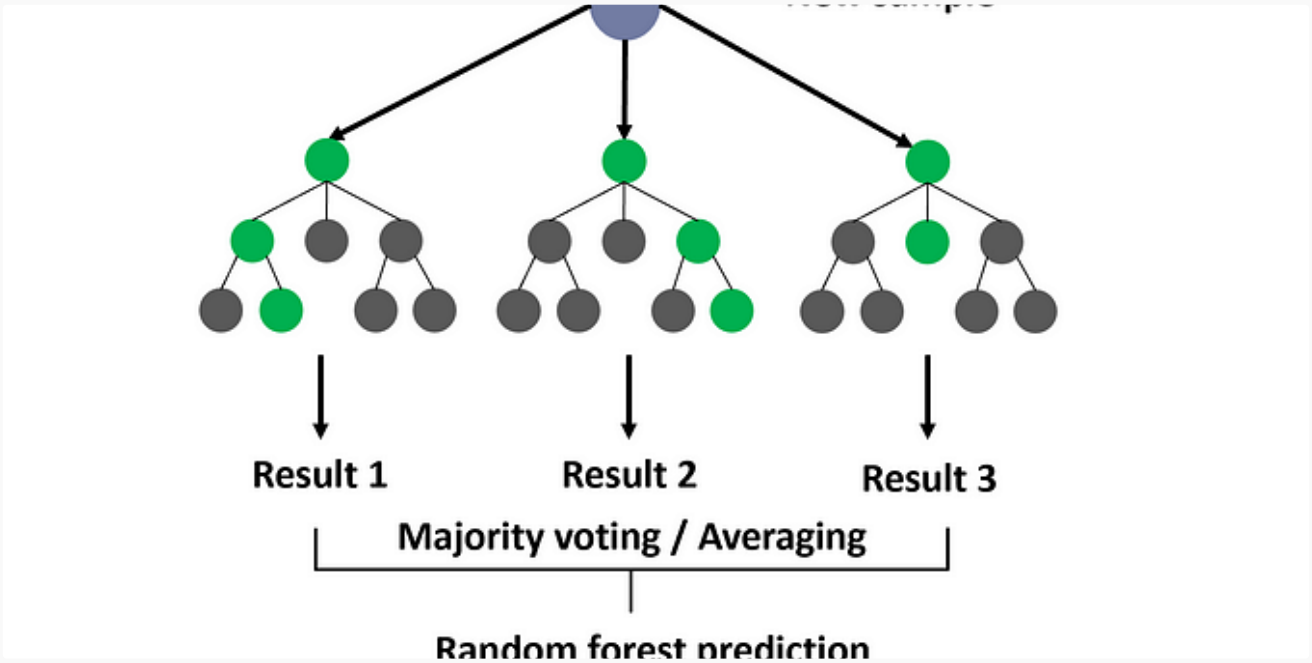
40 stories · 237 saves

Biraj karki

Understanding Support Vector Machines in Machine Learning

Welcome to the world of Support Vector Machines (SVM), a powerful algorithm that has become a cornerstone in the field of machine learning...

Mar 24 5

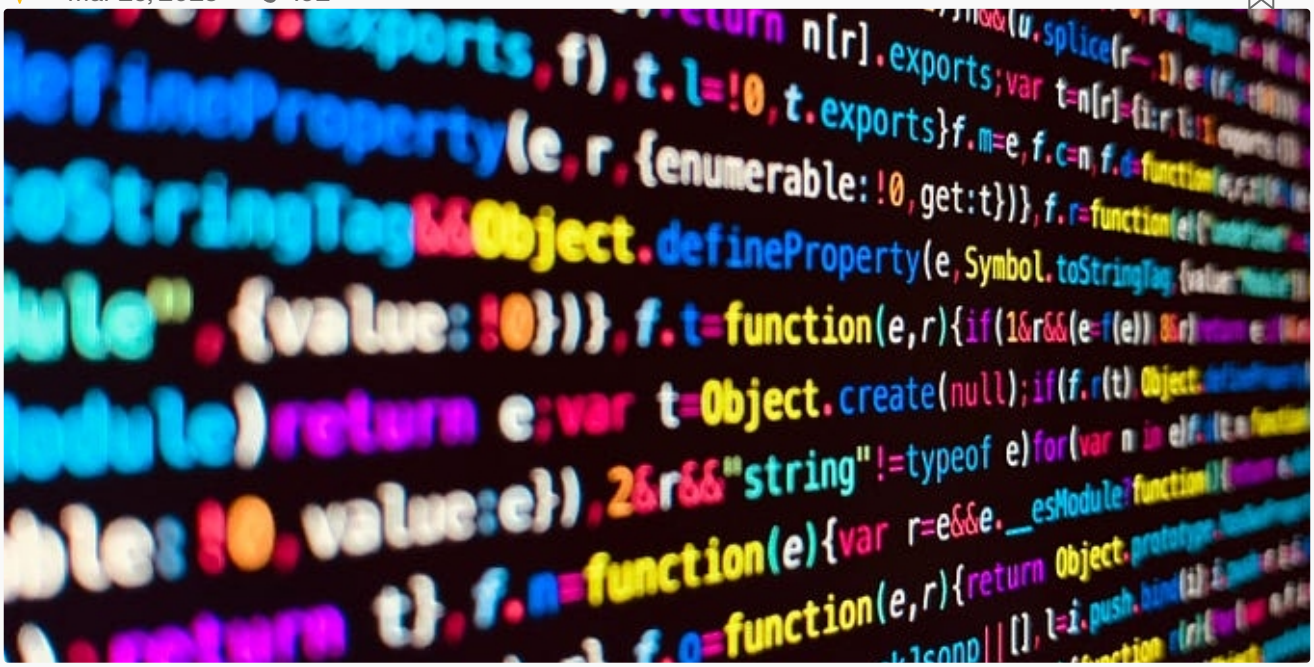


Dr. Roi Yehoshua

Random Forests

Random forests is a powerful machine learning model based on an ensemble of decision trees, where each tree is grown using a random subset...

Mar 25, 2023 192



Jamie Crossman-Smith in Low Code for Data Science

XGBoost Explained: A Beginner's Guide

Understand how XGBoost works, when to use it, and its advantages over other algorithms

Mar 24 271 2

 Zach Quinn in Pipeline: Your Data Engineering Resource

3 Data Science Projects That Got Me 12 Interviews. And 1 That Got Me in Trouble.

3 work samples that got my foot in the door, and 1 that almost got me tossed out.

Aug 29, 2022 4.95K 63

See more recommendations