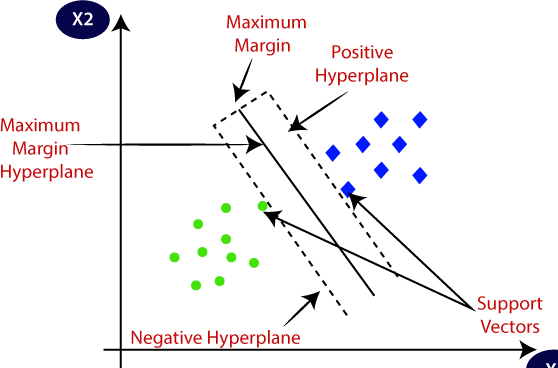
|  |  |  |  |
| --- | --- | --- | --- |
| **Topic: Support Vector Machine** | **Theory** | **Mathematics** | **Numerical** |
|  |  |  |

**Theory questions**

**23. What is Support Vector Machine?**

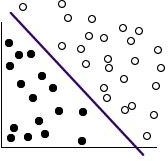
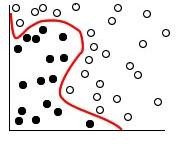
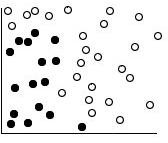
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms,

which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable.

**24. How does the SVM work?**

|  |  |  |
| --- | --- | --- |
| Original dataset | Data with separator added | Transformed data |



A separator between the categories is found, and then the data are transformed in such a way that the separator could be drawn as a hyperplane. Following this, characteristics of new

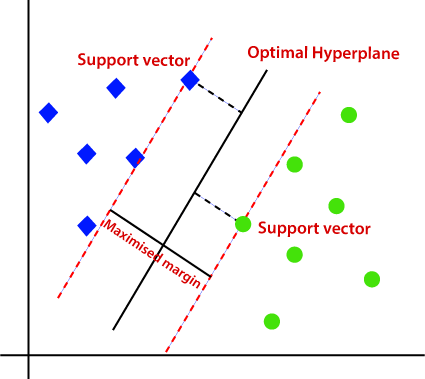
data can be used to predict the group to which a new record should belong. For example, consider the following figure, in which the data points fall into two different categories. The two categories can be separated with a curve, as shown in the figure. After the transformation, the boundary between the two categories can be defined by a hyperplane, as shown in the following figure.

The mathematical function used for the transformation is known as the kernel function. Following are the popular functions.

* Linear
* Polynomial
* Radial basis function (RBF)
* Sigmoid

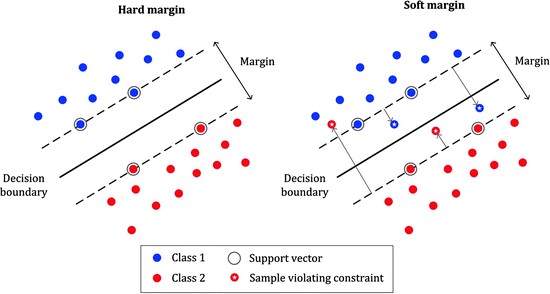
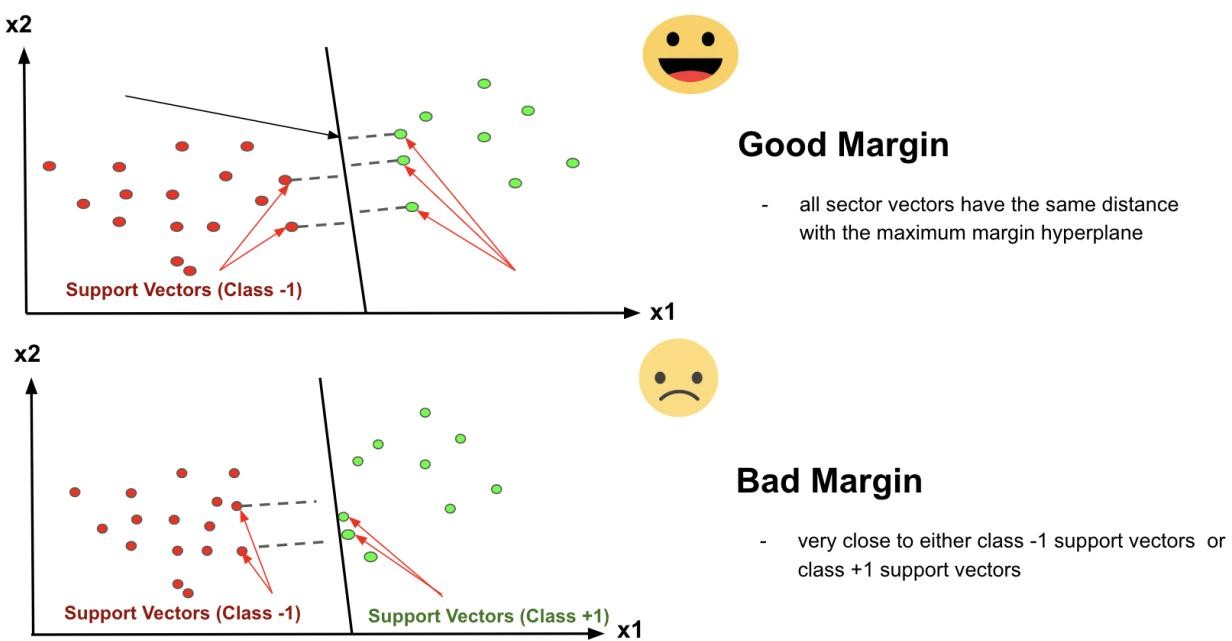
A linear kernel function is recommended when linear separation of the data is straightforward. In other cases, one of the other functions should be used. You will need to experiment with the different functions to obtain the best model in each case, as they each use different algorithms and parameters.

1. **Explain Hyperplanes and Support Vectors. Hyperplane:** There can be multiple lines/decision boundaries to segregate the classes in n- dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM. The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane. We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

**Support Vectors:** The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

1. **Explain Hard and soft margins with the help of sketch** The distance of the vectors from the hyperplane is called the **margin** which is a separation of a line to the closest class points. We would like to choose a hyperplane that maximizes the margin between classes. The graph below shows what good margins and bad margins are.

Again Margin can be sub-divided into,

* **Soft Margin** – As most of the real-world data are not fully linearly separable, we will allow some margin violation to occur which is called soft margin classification. It is better to have a large margin, even though some constraints are violated. Margin violation means choosing a hyperplane, which can allow some data points to stay on either the incorrect side of the hyperplane and between the margin and correct side of the hyperplane.
* **2. Hard Margin** – If the training data is linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible.

**27. Explain Support Vector Machine terminology.**

Support Vector Machines are part of the supervised learning model with an associated

learning algorithm. It is the most powerful and flexible algorithm used for classification, regression, and detection of outliers. It is used in case of high dimension spaces; where each data item is plotted as a point in n-dimension space such that each feature value corresponds to the value of specific coordinate. The classification is made on the basis of a hyperplane/line as wide as possible, which distinguishes between two categories more clearly. Basically, support vectors are the observational points of each individual, whereas the support vector machine is the boundary that differentiates one class from another class.

Some significant terminology of SVM is given below:

* **Support Vectors:** These are the data point or the feature vectors lying nearby to the hyperplane. These help in defining the separating line.
* **Hyperplane:** It is a subspace whose dimension is one less than that of a decision plane. It is used to separate different objects into their distinct categories. The best hyperplane is the one with the maximum separation distance between the two classes.
* **Margins:** It is defined as the distance (perpendicular) from the data point to the decision boundary. There are two types of margins: good margins and margins. **Good margins** are the one with huge margins and the **bad margins** in which the margin is minor.

The main goal of SVM is to find the maximum marginal hyperplane, so as to segregate the dataset into distinct classes. It undergoes the following steps:

* Firstly the SVM will produce the hyperplanes repeatedly, which will separate out the class in the best suitable way.
* Then we will look for the best option that will help in correct segregation.

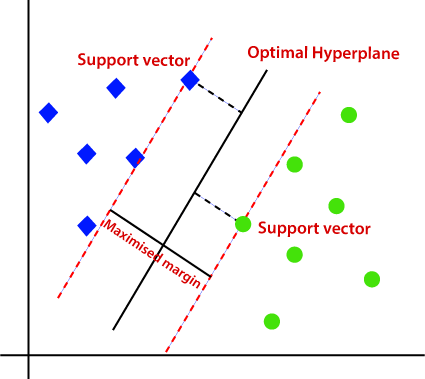
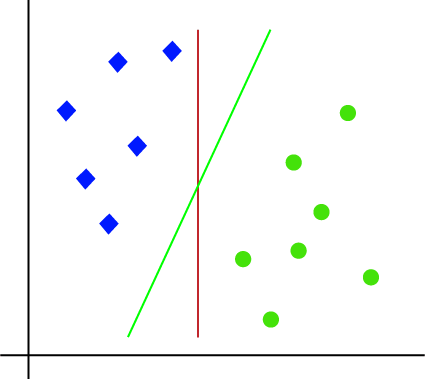
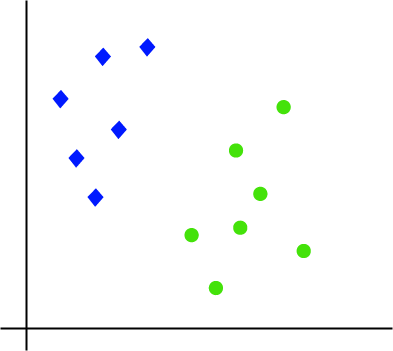
**28. Explain Linear SVM, non-linear SVM.**

#### Linear SVM:

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image: So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image. Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the

hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The

**hyperplane** with maximum margin is called the **optimal hyperplane**.



#### Non-Linear SVM:

If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:

So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

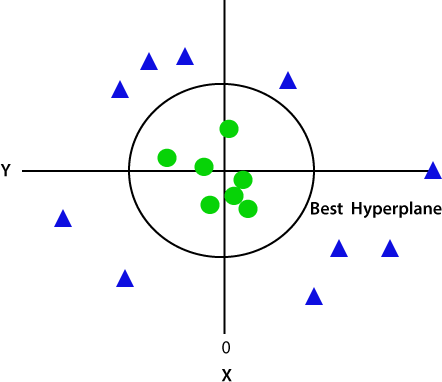
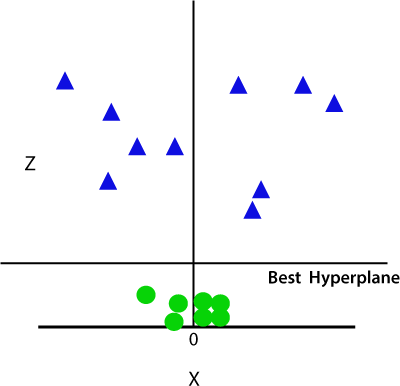
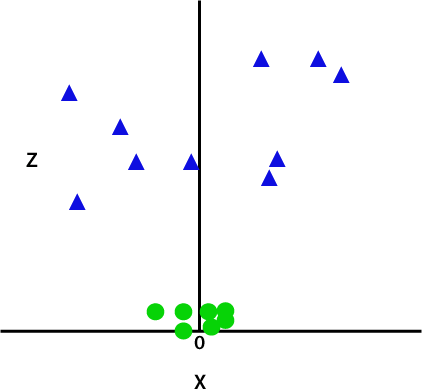
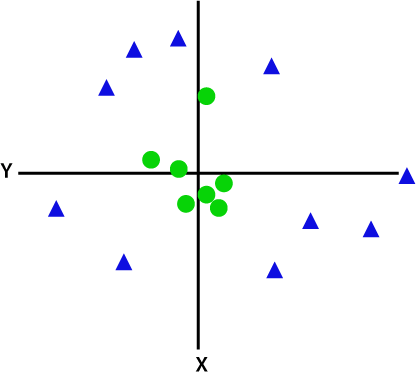
z=x2 +y2

By adding the third dimension, the sample space will become as below image:

So now, SVM will divide the datasets into classes in the following way. Consider the below image:

Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:

Hence we get a circumference of radius 1 in case of non-linear data.



**29. What are advantages and limitations of the Support Vector Machine**

#### Advantages

* SVM‘s are very good when we have no idea on the data.
* Works well with even unstructured and semi structured data like text, Images and trees.
* The kernel trick is real strength of SVM. With an appropriate kernel function, we can solve any complex problem.
* Unlike in neural networks, SVM is not solved for local optima.
* It scales relatively well to high dimensional data. SVM is more effective in high dimensional spaces.
* SVM models have generalization in practice; the risk of over-fitting is less in SVM.
* SVM works relatively well when there is a clear margin of separation between classes.
* SVM is effective in cases where the number of dimensions is greater than the number of samples.
* SVM is relatively memory efficient

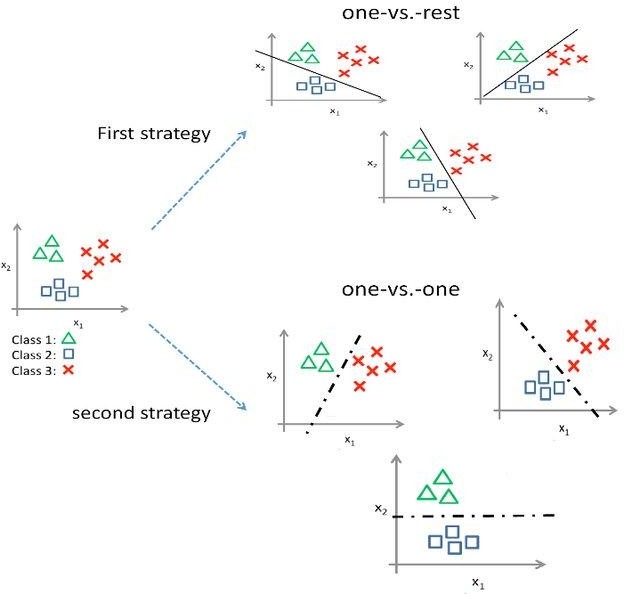
#### Disadvantages

* Choosing a ―good‖ kernel function is not easy.
* SVM algorithm is not suitable for large data sets.
* Long training time for large datasets.
* Difficult to understand and interpret the final model, variable weights and individual impact.
* Since the final model is not so easy to see, we cannot do small calibrations to the model hence it‘s tough to incorporate our business logic.
* The SVM hyper parameters are Cost -C and gamma. It is not that easy to fine-tune these hyper-parameters. It is hard to visualize their impact
* SVM does not perform very well when the data set has more noise i.e. target classes are overlapping.
* In cases where the number of features for each data point exceeds the number of training data samples, the SVM will underperform.
* As the support vector classifier works by putting data points, above and below the classifying hyperplane there is no probabilistic explanation for the classification.

**30. Explain multi class classification methods**

Two different examples of this approach are the One-vs-Rest and One-vs-One strategies.

* Binary classification models like logistic regression and SVM do not support multi-class classification natively and require meta-strategies.
* The One-vs-Rest strategy splits a multi-class classification into one binary classification problem per class.
* The One-vs-One strategy splits a multi-class classification into one binary classification problem per each pair of classes.



#### Explain hyper parameters of SVM.

Hyper parameters of SVM are considered as Kernel, Regularization, Gamma and Margin. **Kernel:** The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role.

For **linear kernel** the equation for prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

f(x) = B(0) + sum(ai \* (x,xi))

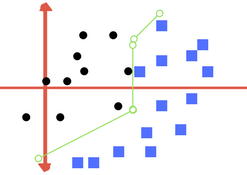
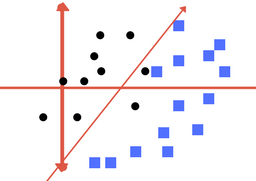
This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients B0 and ai (for each input) must be estimated from the training data by the learning algorithm.

The **polynomial kernel** can be written as *K(x,xi) = 1 + sum(x \* xi)^d* and **exponential** as

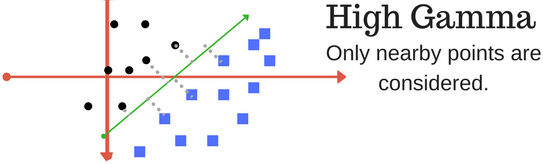
*K(x,xi) = exp(-gamma \* sum((x — xi²)).*

Polynomial and exponential kernels calculates separation line in higher dimension. This is called **kernel trick.**

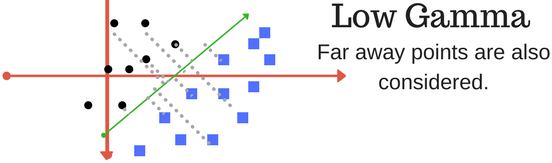
**Regularization:** The Regularization parameter (often termed as C parameter in python‘s sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger- margin separating hyperplane, even if that hyperplane misclassifies more points. The images below are example of two different regularization parameters. Left one has some misclassification due to lower regularization value. Higher value leads to results like right one.



Left: low regularization value, right: high regularization value

**Gamma:** The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‗far‘ and high values meaning ‗close‘. In other words, with low gamma, points far away from plausible separation line are considered in calculation for the separation line. Whereas high gamma means the points close to plausible line are considered in calculation.

High Gamma

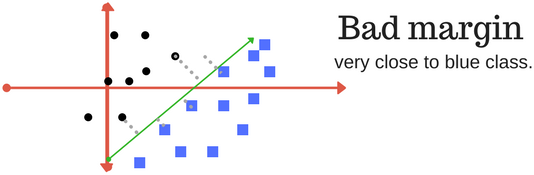
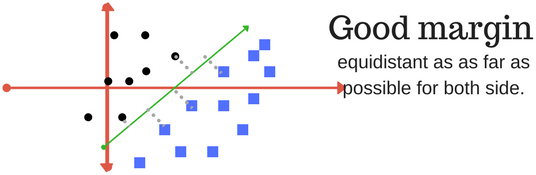


Low Gamma

**Margin:** And finally last but very important characteristic of SVM classifier. SVM to core tries to achieve a good margin.

***A margin is a separation of line to the closest class points.***

A ***good margin*** is one where this separation is larger for both the classes. Images below gives to visual example of good and bad margin. A good margin allows the points to be in their respective classes without crossing to other class.



#### Hyper parameters in detail

|  |  |
| --- | --- |
| **Sr.No** | **Parameter & Description** |
| 1 | ***C*** *− float, optional, default = 1.0*  It is the penalty parameter of the error term. |
| 2 | ***kernel*** *− string, optional, default = ‗rbf‘*  This parameter specifies the type of kernel to be used in the algorithm. we can choose any one among, **‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’**. The default value of kernel would be **‘rbf’**. |
| 3 | ***degree*** *− int, optional, default = 3*  It represents the degree of the ‗poly‘ kernel function and will be ignored by all other kernels. |
| 4 | ***gamma*** *− {‗scale‘, ‗auto‘} or float,*  It is the kernel coefficient for kernels ‗rbf‘, ‗poly‘ and ‗sigmoid‘. |
| 5 | ***optinal default*** *− = ‗scale‘*  If you choose default i.e. gamma = ‗scale‘ then the value of gamma to be used by SVC is 1/(𝑛\_𝑓𝑒𝑎𝑡𝑢𝑟𝑒𝑠∗𝑋.𝑣𝑎𝑟()).  On the other hand, if gamma= ‗auto‘, it uses 1/𝑛\_𝑓𝑒𝑎𝑡𝑢𝑟𝑒𝑠. |
| 6 | ***coef0*** *− float, optional, Default=0.0*  An independent term in kernel function which is only significant in ‗poly‘ and  ‗sigmoid‘. |

|  |  |
| --- | --- |
| 7 | ***tol*** *− float, optional, default = 1.e-3*  This parameter represents the stopping criterion for iterations. |
| 8 | ***shrinking*** *− Boolean, optional, default = True*  This parameter represents that whether we want to use shrinking heuristic or not. |
| 9 | ***verbose*** *− Boolean, default: false*  It enables or disable verbose output. Its default value is false. |
| 10 | ***probability*** *− boolean, optional, default = true*  This parameter enables or disables probability estimates. The default value is false, but it must be enabled before we call fit. |
| 11 | ***max\_iter*** *− int, optional, default = -1*  As name suggest, it represents the maximum number of iterations within the solver. Value -1 means there is no limit on the number of iterations. |
| 12 | ***cache\_size*** *− float, optional*  This parameter will specify the size of the kernel cache. The value will be in MB(MegaBytes). |
| 13 | ***random\_state*** *− int, RandomState instance or None, optional, default = none*  This parameter represents the seed of the pseudo random number generated which is used while shuffling the data. Followings are the options −   * **int** − In this case, *random\_state* is the seed used by random number generator. * **RandomState instance** − In this case, random\_state is the random number generator. * **None** − In this case, the random number generator is the RandonState instance used by np.random. |
| 14 | ***class\_weight*** *− {dict, ‗balanced‘}, optional*  This parameter will set the parameter C of class j to 𝑐𝑙𝑎𝑠𝑠\_𝑤𝑒𝑖𝑔[𝑗]∗𝐶 for SVC. If we use the default option, it means all the classes are supposed to have weight one. On the other hand, if you choose ***class\_weight:balanced***, it will use the values of y to automatically adjust weights. |
| 15 | ***decision\_function\_shape*** *− ovo‘, ‗ovr‘, default = ‗ovr‘*  This parameter will decide whether the algorithm will return **‘ovr’** (one-vs-rest) decision function of shape as all other classifiers, or the original **ovo**(one-vs-one) decision function of libsvm. |
| 16 | ***break\_ties*** *− boolean, optional, default = false*  **True** − The predict will break ties according to the confidence values of decision\_function  **False** − The predict will return the first class among the tied classes. |

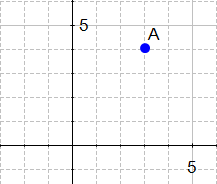
Thus hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.

#### https://miro.medium.com/max/1400/1*8pcHwrCmKbcQRWXflgv4gg.pngMathematics based questions 32. Explain following Kernel functions: linear, polynomial, rbf, sigmoid

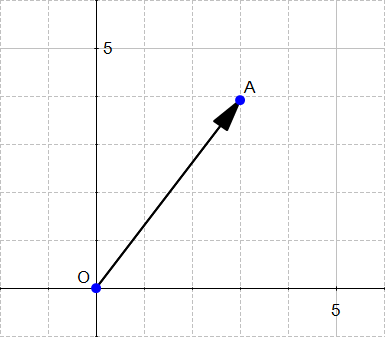
**Problems/Numerical 33. Problems based on calculating hyperplane and margin** In Support Vector Machine, there is the word **vector.** That means it is important to understand vector well and how to use them.

* + What is a vector?
    - its norm
    - its direction
  + How to add and subtract vectors?
  + What is the dot product?
  + How to project a vector onto another?
  + What is the equation of the hyperplane?
  + How to compute the margin?

#### What is a vector?

If we define a point *A*(3,4) in 2 we can plot it like this.

Definition: Any point *x*=(*x*1, *x*2),*x*≠0, in 2 specifies a vector in the plane, namely the vector starting at the origin and ending at x.

This definition means that there exists a vector between the origin and A.

If we say that the point at the origin is the point *O*(0,0) then the vector above is the vector

⃗𝑂⃗⃗⃗⃗𝐴⃗→. We could also give it an arbitrary name such as **u**. **Note**:

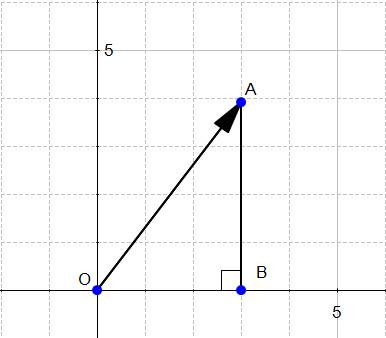
You can notice that we write vector either with an arrow on top of them, or in bold, in the rest of this text I will use the arrow when there is two letters like 𝑂⃗⃗⃗⃗⃗𝐴⃗→ and the bold notation otherwise.

Ok so now we know that there is a vector, but we still don't know what **IS** a vector.

Definition: A vector is an object that has both a magnitude and a direction. We will now look at these two concepts.

#### The magnitude

**The magnitude or length of a vector *x* is written** ∥***x***∥ **and is called its norm.**

For our vector ⃗𝑂⃗⃗⃗⃗𝐴⃗→, ∥*OA*∥ is the length of the segment *OA*

From Figure, we can easily calculate the distance OA using Pythagoras' theorem:

*OA*2=*OB*2+*AB*2 *OA*2=32+42 *OA*2=25 *OA*=5

∥*OA*∥=5

#### The direction

The direction is the second component of a vector.

Definition: The **direction** of a vector **u**(*u*1,*u*2) is the vector 𝑢1 , 𝑢2

Where does the coordinates of **w** come from?

## Understanding the definition

‖𝑢‖

‖𝑢‖

To find the direction of a vector, we need to use its angles.

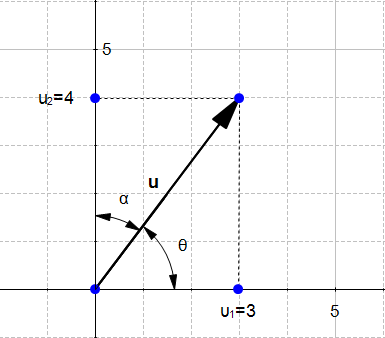


Figure 4 displays the vector **u**(*u*1,*u*2) with *u*1=3 and *u*2=4 We could say that:

*Naive definition 1: The direction of the vector* ***u*** *is defined by the angle θ with respect to the horizontal axis, and with the angle α with respect to the vertical axis.* This is tedious. Instead of that we will use the cosine of the angles.

In a right triangle, the cosine of an angle *β* is defined by :

### cos(β)=adjacent/hypotenuse

In Figure 4 we can see that we can form two right triangles, and in both case the adjacent side will be on one of the axis. Which means that the definition of the cosine implicitly contains the axis related to an angle. We can rephrase our naïve definition to :

### Naive definition 2: The direction of the vector **u** is defined by the cosine of the angle θ and the cosine of the angle α.

Now if we look at their values:

*cos*(*θ*) = 𝑢1

‖𝑢‖

*cos*(*α*) = 𝑢2

‖𝑢‖

Hence the original definition of the vector ***w***. That's why its coordinates are also called

*direction cosine*.

***Computing the direction vector***

We will now compute the direction of the vector **u** from Figure 4.

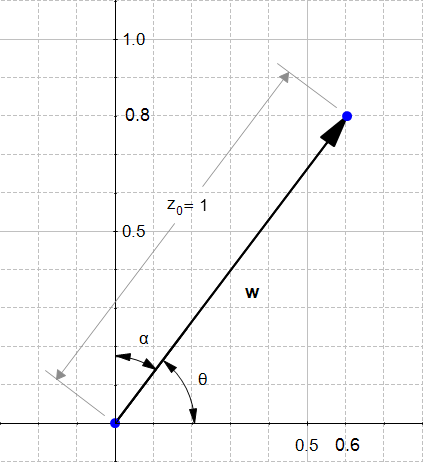
*cos*(*θ*) = 𝑢1 = 3/5 = 0.6

‖𝑢‖

*cos*(*α*) = 𝑢2 = 4/5 = 0.8

‖𝑢‖

The direction of **u**(3,4) is the vector **w**(0.6,0.8) If we draw this vector we get Figure 5:



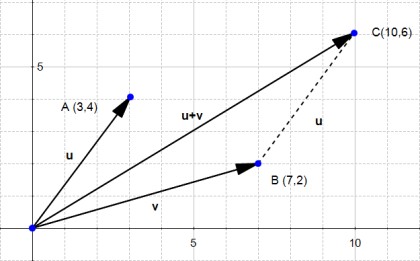
We can see that ‗**w’** as indeed the same look as **u** except it is smaller. Something interesting about direction vectors like **w** is that their norm is equal to 1. That's why we often call them **unit vectors**.

#### The sum of two vectors

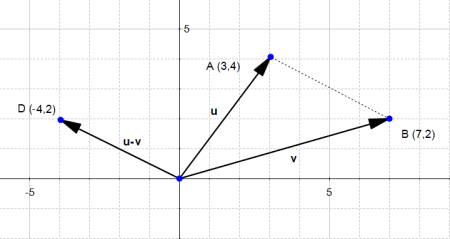
Given two vectors **u**(*u*1,*u*2) and **v**(*v*1,*v*2) then :

**u**+**v**=(*u*1+*v*1,*u*2+*v*2)

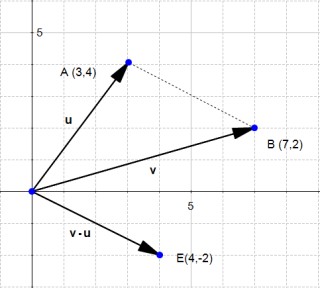
Which means that adding two vectors gives us **a third vector** whose coordinate are the sum of the coordinates of the original vectors. You can convince yourself with the example below:



#### The difference between two vectors

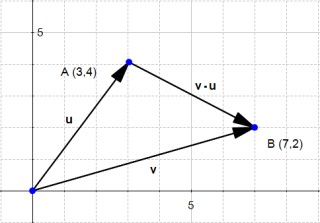
The difference works the same way: **u**−**v**=(*u*1−*v*1,*u*2−*v*2)

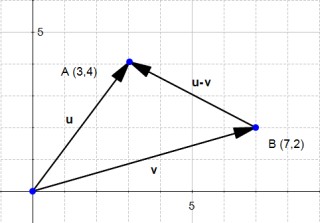
Since the subtraction is not [commutative,](http://en.wikipedia.org/wiki/Commutative_property) we can also consider the other case:

**v**−**u**=(*v*1−*u*1,*v*2−*u*2)

The last two pictures describe the "*true*" vectors generated by the difference of **u** and **v**. However, since a vector has a magnitude and a direction, we often consider that parallel translate of a given vector (vectors with the same magnitude and direction but with a different origin) are the same vector, just drawn in a different place in space.

So don't be surprised if you meet the following:





If you do the math, it looks wrong, because the end of the vector **u−v** is not in the right point, but it is a convenient way of thinking about vectors which you'll encounter often.

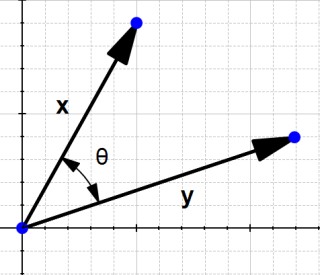
#### The dot product

One **very** important notion to understand SVM is [the dot product.](http://en.wikipedia.org/wiki/Dot_product)

Definition: Geometrically, it is the product of the Euclidian magnitudes of the two vectors and the cosine of the angle between them

Which means if we have two vectors **x** and **y** and there is an angle *θ* (theta) between them, their dot product is:

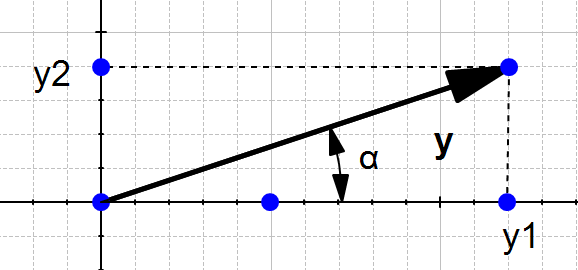
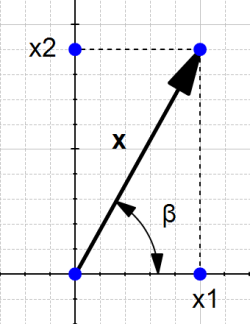
**x**⋅**y**=∥*x*∥∥*y*∥*cos*(*θ*) **Why?**

To understand let's look at the problem geometrically.

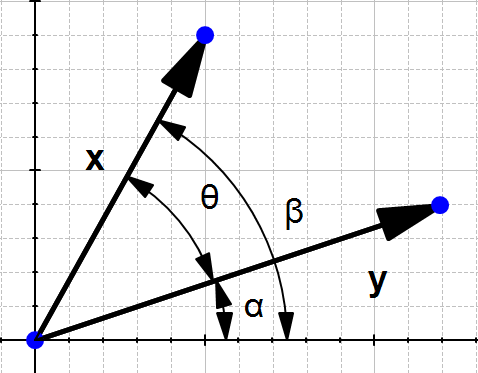
In the definition, they talk about *cos*(*θ*), let's see what it is. By definition we know that in a right-angled triangle:

*cos*(*θ*)=*adjacent/hypotenuse*

In our example, we don't have a right-angled triangle.

However if we take a different look Figure 12 we can find two right-angled triangles formed by each vector with the horizontal axis.

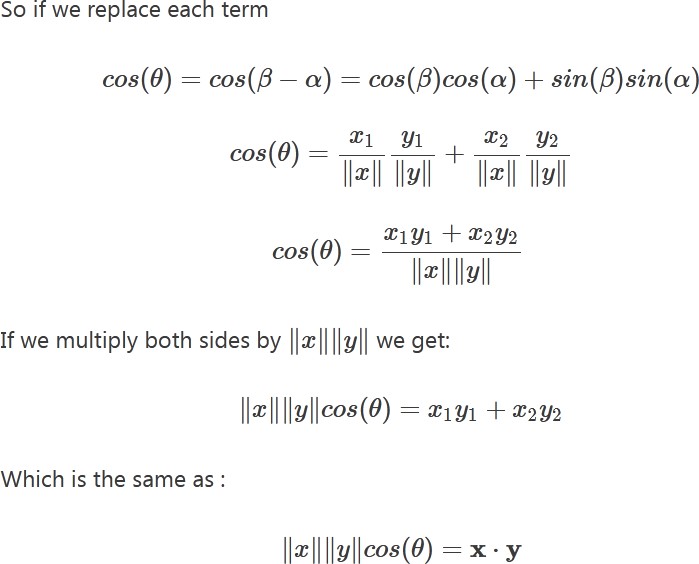
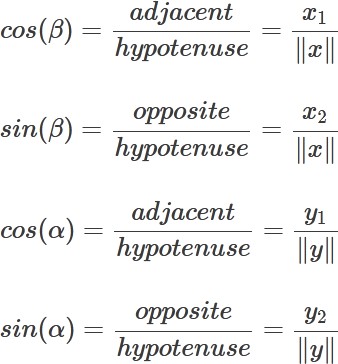
So now we can view our original schema like this:

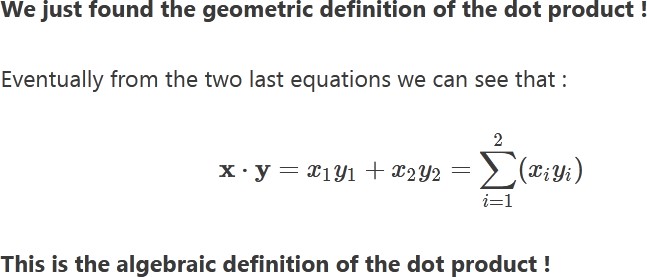


We can see that *θ*=*β*−*α*

So computing *cos*(*θ*) is like computing *cos*(*β*−*α*)

There is a special formula called the *difference identity for cosine* which says that:

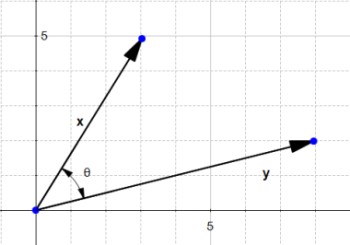
*cos*(*β*−*α*)=*cos*(*β*)*cos*(*α*)+*sin*(*β*)*sin*(*α*) Let's use this formula!



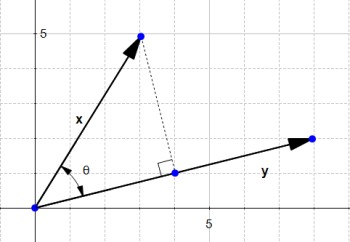
#### A few words on notation

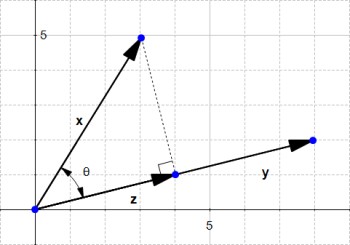
The dot product is called like that because we write a dot between the two vectors. Talking about the dot product **x**⋅**y** is the same as talking about the **inner product** ⟨*x*,*y*⟩ (in linear algebra) **scalar product** because we take the product of two vectors and it returns a scalar (a real number)

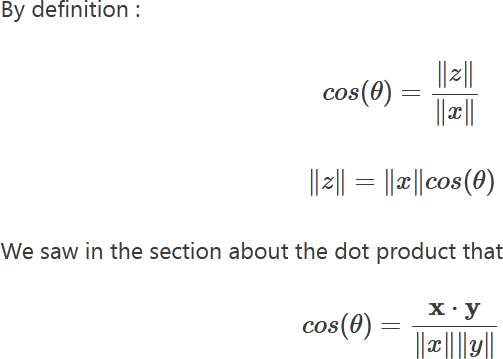
#### The orthogonal projection of a vector

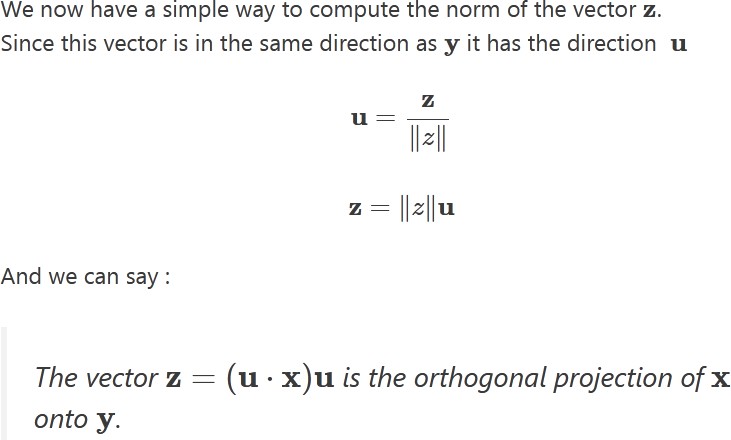
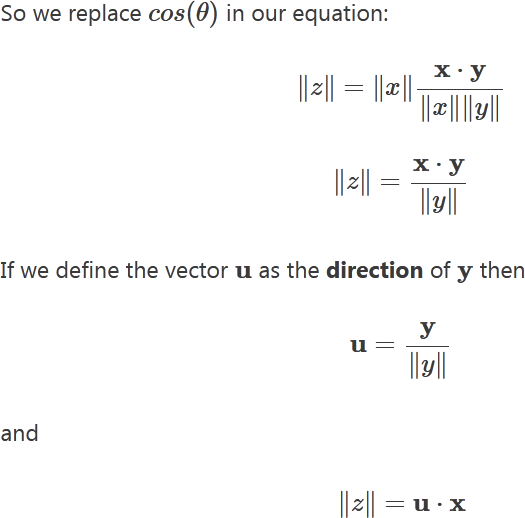
Given two vectors **x** and **y**, we would like to find the orthogonal projection of **x** onto **y**.

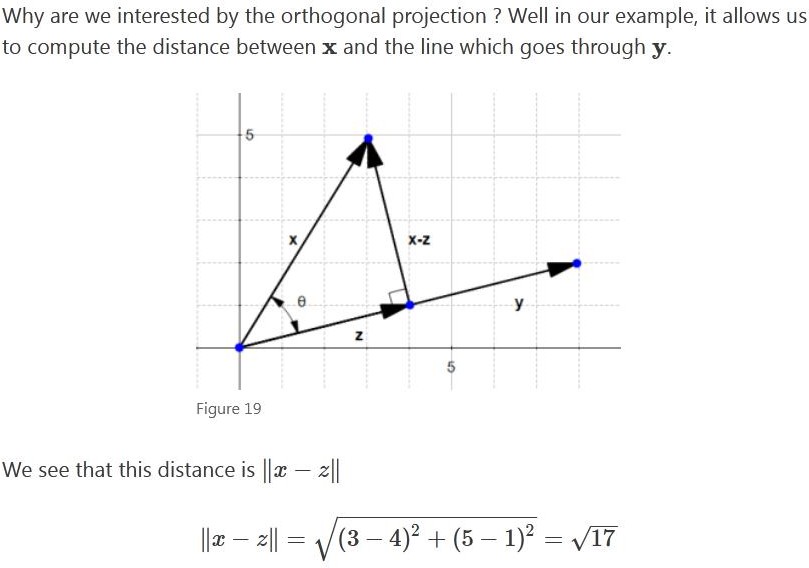
To do this we project the vector **x** onto **y**



This gives us the vector **z**







#### The SVM hyperplane: Understanding the equation of the hyperplane

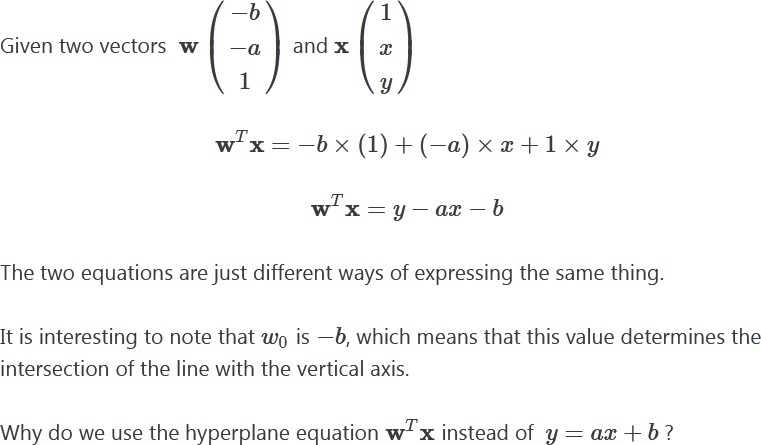
You probably learnt that an equation of a line is: *y*=*ax*+*b*. However when reading about hyperplane, you will often find that the equation of a hyperplane is defined by:

**w***T***x**=0

How does these two forms relate?

In the hyperplane equation you can see that the name of the variables is in bold. Which means that they are vectors? Moreover, **w***T***x** is how we compute the inner product of two vectors, and if you recall, the inner product is just another name for the dot product!

Note that *y*=*ax*+*b* is the same thing as *y*−*ax*−*b*=0

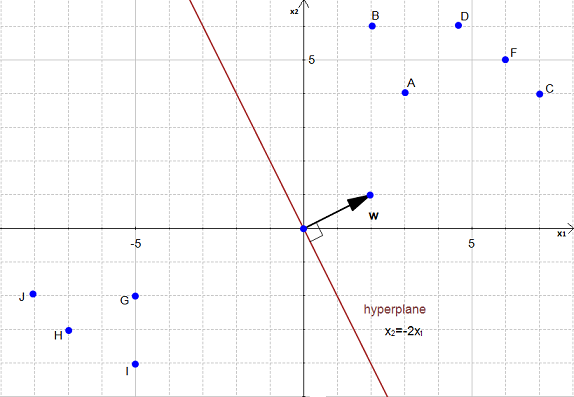


For two reasons:

* it is easier to work in more than two dimensions with this notation,
* the vector **w** will always be normal to the hyperplane(Note: I received a lot of questions about the last remark. **W** will always be normal because we use this vector to define the hyperplane, so by definition it will be normal. As you can see [this page,](http://tutorial.math.lamar.edu/Classes/CalcII/EqnsOfPlanes.aspx) when we define a hyperplane, we suppose that we have a vector that is orthogonal to the hyperplane)

And this last property will come in handy to compute the distance from a point to the hyperplane.

#### Compute the distance from a point to the hyperplane

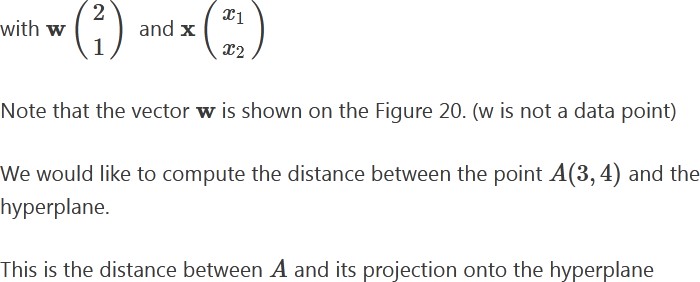
In Figure 20 we have a hyperplane, which separates two groups of data.

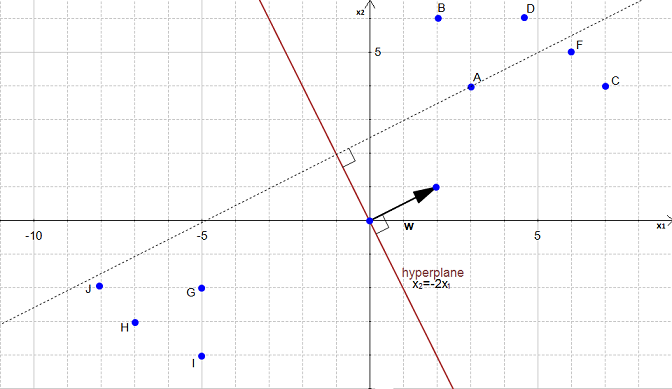
To simplify this example, we have set *w*0=0.

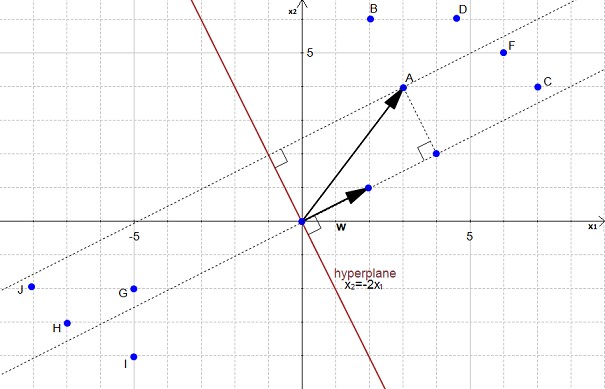
As you can see on the Figure 20, the equation of the hyperplane is:

*x*2=−2*x*1 which is equivalent to

**w***T***x**=0





We can view point *A* as a vector from origin to *A*. If we project it onto the normal vector **w**

We get the vector **p**

