Kernels in Support Vector Machine

Kernel

SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be different types. For example **Linear, Nonlinear, Polynomial, Radial basis function (RBF), and Sigmoid.**

4. Examples of SVM Kernels

Let us see some common kernels used with SVMs and their uses:

4.1. Polynomial kernel

It is popular in image processing.
Equation is:

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

Polynomial kernel equation

where d is the degree of the polynomial.

4.2. Gaussian kernel

It is a general-purpose kernel; used when there is no prior knowledge about the data. Equation is:

$$k(x, y) = \exp\left(-\frac{||x - y||^2}{2\sigma^2}\right)$$

Gaussian kernel equation

4.3. Gaussian radial basis function (RBF)

It is a general-purpose kernel; used when there is no prior knowledge about the data.
Equation is:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma||\mathbf{x_i} - \mathbf{x_j}||^2)$$

Gaussian radial basis function (RBF)

, for:

$$\gamma > 0$$

Gaussian radial basis function (RBF)

Sometimes parametrized using:

Sometimes parametrized using:

$$\gamma = 1/2\sigma^2$$

Gaussian radial basis function (RBF)

The kernel functions return the inner product between two points in a suitable feature space. When the data we are working with is not linearly separable (therefore leading to poor linear SVM classification results), it is possible to apply a technique known as the Kernel Trick. This method is able to map our non-linear separable data into a higher dimensional space, making our data linearly separable.

<u>When to use which Kernel</u>

- The most used type of kernel function is **RBF.** Because it has localized and finite response along the entire x-axis.
- The kernel is effectively a similarity measure. In the absence of expert knowledge, the Radial Basis Function kernel makes a good default kernel (once you have established it is a problem requiring a non-linear model). The linear kernel works fine if your dataset if linearly separable.
- Both Logistic Regression and SVM are supervised classification algorithms. We can decide which model to go with beforehand, if we have an idea about total number of **features (n)** and **total training data points (m).**

- If **n is relatively larger than m** (like for text classification), then LR and linear SVM (with linear kernel) can be used and both have similar complexity.
- If **n is small** and **m is intermediate(just enough to make the modelling**) , then SVM with kernel like Gaussian kernel provides better results and will be a bit harder to work on than LR.
- If **n is small** and **m is very large**, then this tend to underfit the test data. So gather additional features or add polynomial features to make it close to our 1st case and use LR or linear SVM
- If **n is very large** and **m is small**, then this tend to overfit the data. So we try to remove correlations between features, remove non-significant features, gather more datasets for training or use regularisation. Eventually we again end up with 1st case and use LR or Linear SVM.

- If your data is not linearly separable, it doesn't make sense to use a linear classifier. In any case, I wouldn't bother too much about the polynomial kernel. In practice, it is less useful for efficiency (computational as well as predictive) performance reasons.

So, the **rule of thumb is**: use linear SVMs (or logistic regression) for linear problems, and nonlinear kernels such as the Radial Basis Function kernel for non-linear problems.

- First: **Why is the RBF Kernel the most widely used**? Because SVM is intrinsically a linear separator when the classes are not linearly separable we can project the data into a high dimensionality space and with a high probability find a linear separation. This is Cover's theorem and the **RBF Kernel does exactly that: it projects the data into infinite dimensions and then finds a linear separation.**
- The linear kernel works great when you have a lot of features because then chances are your data is already linearly separable and a SVM will find the best separating hyperplane. Linear kernels are then great for very sparse data like text.
- When data is not linearly separable the first choice is always a RBF kernel because they are very flexible and for the reasons I explained in the first paragraph.

The practical way to decide which kernel to use is by cross-validation.

To determine the optimal Kernel via experimentation. Doing so involves:

1. Implementing a version of one's SVM using each kernel
2. Evaluating the SVM's performance with each kernel via **Cross Validation**
3. Selecting the kernel that yielded optimal results

- Your choice is definitely data dependent. The first thing to try is linear SVM. The linear kernel works fine if your dataset if linearly separable. Next you could try SVM with RBF kernel. The downside of the latter is that its complexity grows with the size of the training set. Other kernels like polynomial kernel are rarely used due to poor efficiency.
- Selection of parameters is task dependent and should be done on the training data using cross-validation. Tuning the parameter "C" is very important. A larger C makes your model more complex and lowers its bias, hence you'll have higher chance of overfitting. A low C gives you a model with low variance and high bias.
- There are some kernels that are best for a particular tasks, while other kernels are not. For example you might apply **'RBF'** kernels for most of the tasks, but for tasks dealing with text data **'string'** kernel might work better. And the other important parameter 'C' determines overfitting/underfitting. High 'C' might lead to overfitting and low 'C' might lead to underfitting.
- There are many different types of Kernels which can be used to create this higher dimensional space, some examples are linear, polynomial, Sigmoid and Radial Basis Function (RBF).
- An additional parameter called gamma can be included to specify the influence of the kernel on the model.
- It is usually suggested to use linear kernels if the number of features is larger than the number of observations in the dataset (otherwise RBF might be a better choice).
- When working with a large amount of data using RBF, speed might become a constraint to take into account. Reducing the number of features in Machine Learning

plays a really important role especially when working with large datasets. This can in fact: speed up training, avoid overfitting and ultimately lead to better classification results thanks to the reduced noise in the data.

SVM:

The main objective in SVM is to find the optimal hyperplane to correctly classify between data points of different classes. The hyperplane dimensionality is equal to the number of input features minus one (e.g. when working with three feature the hyperplane will be a two-dimensional plane).

Data points on one side of the hyperplane will be classified to a certain class while data points on the other side of the hyperplane will be classified to a different class.

The data points closest to the hyperplane are called Support Vectors. Support Vectors determines the orientation and position of the hyperplane, in order to maximise the classifier margin.

There are two main types of classification SVM algorithms **Hard Margin** and **Soft Margin**:

- **Hard Margin:** aims to find the best hyperplane without tolerating any form of misclassification.

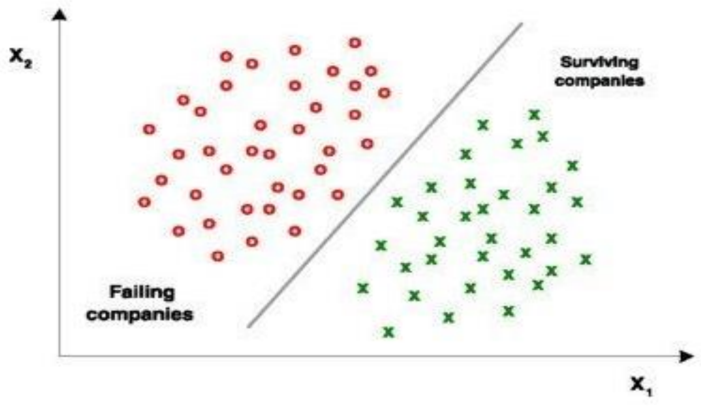
- **Soft Margin:** we add a degree of tolerance in SVM. In this way we allow the model to voluntary misclassify a few data points if that can lead to identifying a hyperplane able to generalise better to unseen data.

Soft Margin SVM can be implemented in by adding a C penalty term. The bigger C and the more penalty the algorithm gets when making a misclassification.

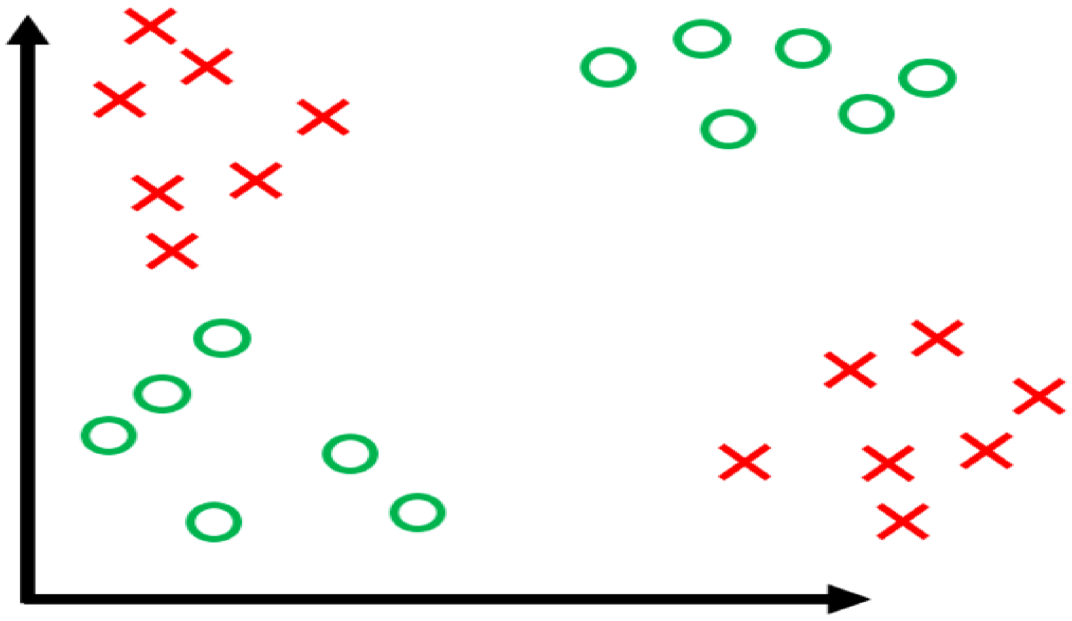
Linearly Separable: The idea of linearly separable is easiest to visualize and understand in 2 dimensions. Let the two classes be represented by colors red and green.

A dataset is said to be linearly separable if it is possible to draw a line that can separate the red and green points from each other.

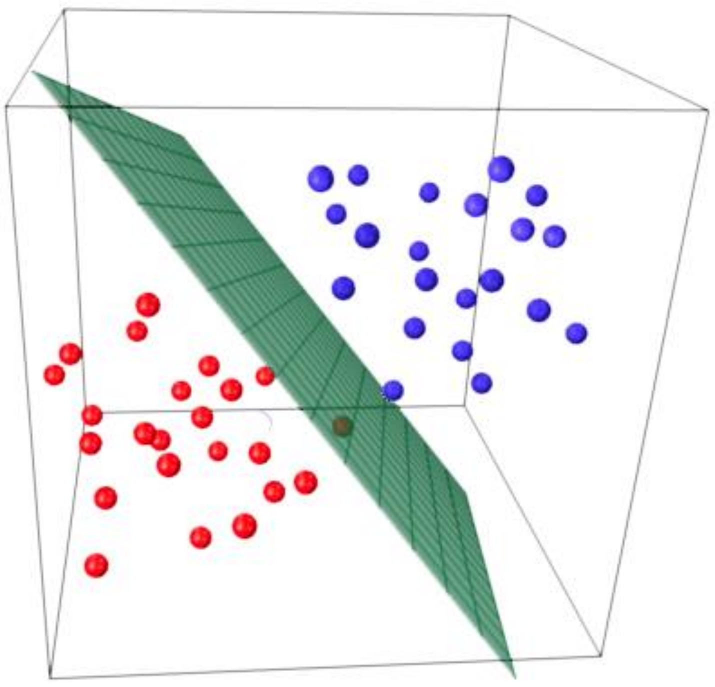
Here are same examples of **linearly separable data:**



And here are some **examples of linearly non-separable data**



This concept can be extended to three or more dimensions as well. For example, below is an **example of a three dimensional dataset that is linearly separable**



Algebraic definition:

Algebraically, the separator is a linear function, i.e. if data point x is given by (x1, x2), when the separator is a function f(x) = w1*x1 + w2*x2 + b

All points for which f(x) = 0, are on the separator line. All points for which f(x) > 0 are on one side of the line, and all points for which f(x) < 0 are on the other side.

Support vectors are the data points that lie closest to the decision surface (or hyperplane). They are the data points most difficult to classify. In 2 dimensions, can separate by a line. In higher dimensions, need hyperplanes.

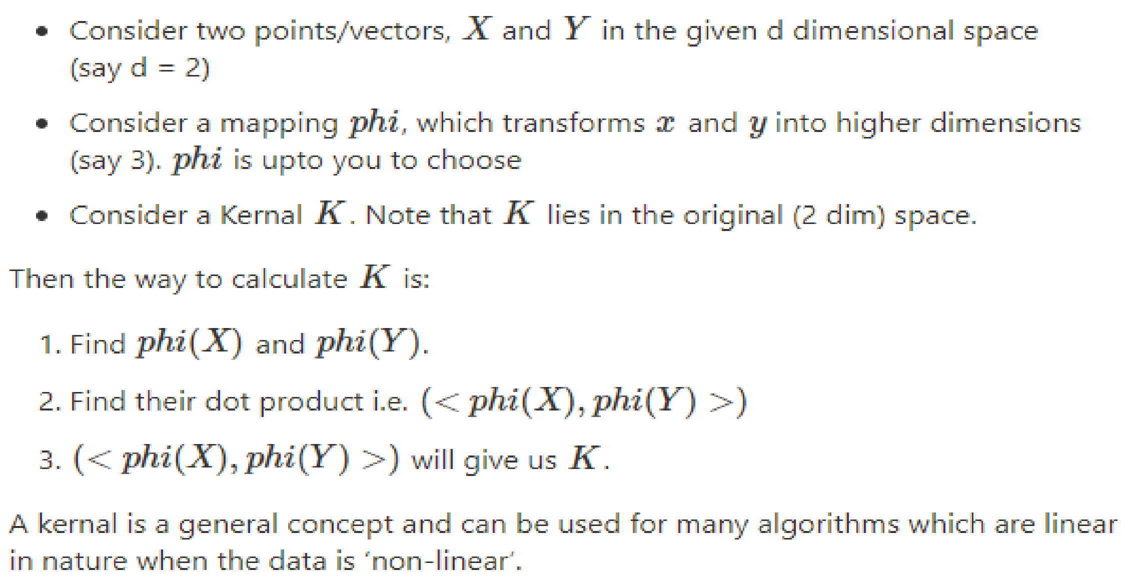
Kernel

A **kernel** is just a transformation of your input data that allows you (or an algorithm like SVMs) to treat/process it more easily.

Intuitively, a kernel function measures the **similarity** between two data points. So, for instance, if your task is object recognition/ classify images, then a good kernel will assign a high score to a pair of images that contain the same objects, and a low score to a pair of images with different objects.

A **kernel** is a similarity function. It is a function that you, as the domain expert, provide to a machine learning algorithm. It takes two inputs and spits out how similar they are.

1. For a dataset with n features (~n-dimensional), SVMs find an n-1 dimensional hyperplane to separate it (let us say for classification)
2. Thus, SVMs perform very badly with datasets that are not linearly separable
3. But, quite often, it's possible to transform our not-linearly-separable dataset into a higher-dimensional dataset where it becomes linearly separable, so that SVMs can do a good job
4. Enter The Kernel Trick
    1. Thankfully, the only thing SVMs need to do in the (higher-dimensional) feature space (while training) is computing the pair-wise dot products
    2. For a given pair of vectors (in a lower-dimensional feature space) and a transformation into a higher-dimensional space, there exists a function (The Kernel Function) which can compute the dot product in the higher-dimensional space without explicitly transforming the vectors into the higher-dimensional space first.



- Consider two points/vectors, X and Y in the given d dimensional space (say d = 2)
- Consider a mapping phi, which transforms x and y into higher dimensions (say 3). phi is upto you to choose
- Consider a Kernal K. Note that K lies in the original (2 dim) space.

Then the way to calculate K is:

1. Find $phi(X)$ and $phi(Y)$.
2. Find their dot product i.e. $(<phi(X), phi(Y)>)$
3. $(<phi(X), phi(Y)>)$ will give us K.

A kernal is a general concept and can be used for many algorithms which are linear in nature when the data is 'non-linear'.

SVM

The SVM is a machine learning algorithm which solves classification problems. Kernel is a way of computing the dot product of two vectors x and y in some (possibly very high dimensional) feature space, which is why kernel functions are sometimes called "generalized dot product".

A kernel is a function k that corresponds to this dot product, i.e. k(x,y)=ϕ(x)Tϕ(y)

A very simple and intuitive way of thinking about **kernels** (at least for SVMs) is a similarity function. Given two objects, the kernel outputs some similarity score. The objects can be anything starting from two integers, two real valued vectors, trees whatever provided that the kernel function knows how to compare them.

The arguably simplest example is the linear kernel, also called dot-product. Given two vectors, the similarity is the length of the projection of one vector on another.

Another interesting kernel examples is Gaussian kernel. Given two vectors, the similarity will diminish with the radius of σ. The distance between two objects is "reweighted" by this radius parameter.

1. **Kernel**: The function used to map a lower dimensional data into a higher dimensional data.
2. **Hyper Plane**: In SVM this is basically the separation line between the data classes. Although in SVR we are going to define it as the line that will help us predict the continuous value or target value
3. **Boundary line**: In SVM there are two lines other than Hyper Plane which creates a margin. The support vectors can be on the Boundary lines or outside it. This boundary line separates the two classes. In SVR the concept is same.
4. **Support vectors**: This are the data points which are closest to the boundary. The distance of the points is minimum or least.

When we don’t have linear separable set of training data, the Kernel trick comes handy. ***The idea is mapping the non-linear separable data-set into a higher dimensional space where we can find a hyperplane that can separate the samples.***

In SVM, maximization depends only on the dot products of support vectors, that means if we use a mapping function that maps our data into a higher dimensional space, then, the maximization and decision rule will depend on the dot products of the mapping function for different samples.

And Voila!! If we have a function K defined as below

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j) \quad , \tag{1.21}$$

then we just need to know K and not the mapping function itself. This function is known as **Kernel function** and it reduces the complexity of finding the mapping function. So, **Kernel function defines inner product in the transformed space.**

Let us look at some of the most used kernel functions

$$\begin{aligned}
K(x_i, x_j) &= (x_i \cdot x_j + 1)^p;\ \text{polynomial kernel.} \qquad (1\\
K(x_i, x_j) &= e^{\frac{-1}{2\sigma^2}(x_i - x_j)^2};\ \text{Gaussian kernel; Special case of Radial Basis Function.}\\
K(x_i, x_j) &= e^{-\gamma(x_i - x_j)^2};\ \text{RBF Kernel}\\
K(x_i, x_j) &= \tanh(\eta\, x_i \cdot x_j + \nu);\ \text{Sigmoid Kernel; Activation function for NN.}
\end{aligned}$$

Support Vector Machine

SVM is a supervised machine learning algorithm which can be used for both classification and regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well.

You need to remember a thumb rule to identify the right hyper-plane: “Select the hyper-plane which segregates the two classes better.

Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. In SVM, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, SVM has a technique called the **kernel trick**. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels. It is mostly useful in non-linear separation problem.

“**Hard-Margin**” SVM, i.e., an SVM that is capable of perfectly classifying data that are completely linearly separable.

Hyperplane is an (n minus 1)-dimensional subspace for an n-dimensional space. For a 2-dimension space, its hyperplane will be 1-dimension, which is just a line. For a 3-dimension space, its hyperplane will be 2-dimension, which is a plane that slice the cube.

In the linearly separable case, SVM is trying to find the hyperplane that maximizes the margin, with the condition that both classes are classified correctly. But in reality, datasets are probably never linearly separable, so the condition of 100% correctly classified by a hyperplane will never be met.

SVM address non-linearly separable cases by introducing two concepts: **Soft Margin** and **Kernel Tricks.**
Soft Margin: try to find a line to separate, but tolerate one or few misclassified dots (e.g. the dots circled in red)
Kernel Trick: try to find a non-linear decision boundary

Applying Soft Margin, SVM tolerates a few dots to get misclassified and tries to balance the trade-off between finding a line that maximizes the margin and minimizes the misclassification.

Degree of tolerance
How much tolerance(soft) we want to give when finding the decision boundary is an important hyper-parameter for the SVM (both linear and nonlinear solutions). it is represented as the penalty term — ‘C’. The bigger the C, the more penalty SVM gets when it makes misclassification.

Kernel Trick

What Kernel Trick does is it utilizes existing features, applies some transformations, and creates new features. Those new features are the key for SVM to find the nonlinear decision boundary. We can choose ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable as our kernel/transformation.

The most popular/basic RBF kernel is the Gaussian Radial Basis Function:

$$\phi(x, center) = exp(-\gamma \|x - center\|^2)$$

Gamma (γ) controls the influence of new features — **Φ(x, center)** on the decision boundary. The higher the gamma, the more influence of the features will have on the decision boundary, more wiggling the boundary will be.

To sum up, SVM in the linear non-separable cases:

- By combining the **soft margin** (tolerance of misclassifications) and **kernel trick** together, Support Vector Machine is able to structure the decision boundary for linear non-separable cases.

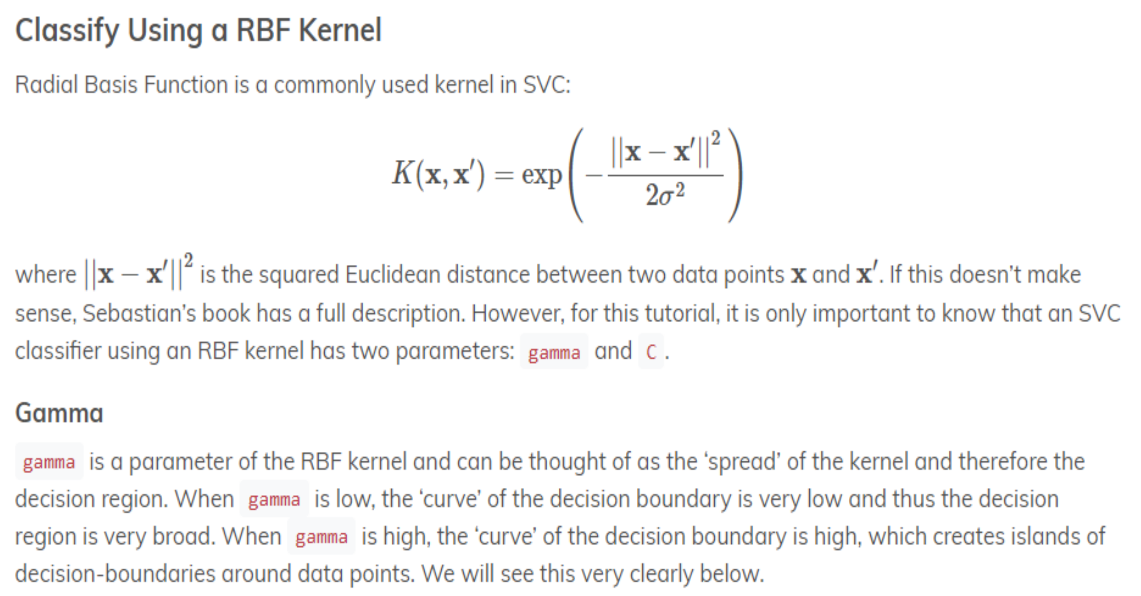
<u>Parameter C and Gamma</u>

- Hyper-parameters like C or Gamma control how wiggling the SVM decision boundary could be.

1. higher the C, the more penalty SVM was given when it misclassified, and therefore the less wiggling the decision boundary will be
2. higher the gamma, the more influence the feature data points will have on the decision boundary, thereby the more wiggling the boundary will be

Wiggle: Move or cause to move up and down or from side to side with small rapid movements

The most basic way to use a SVC is with a linear kernel, which means the decision boundary is a straight line (or hyperplane in higher dimensions).



Classify Using a RBF Kernel

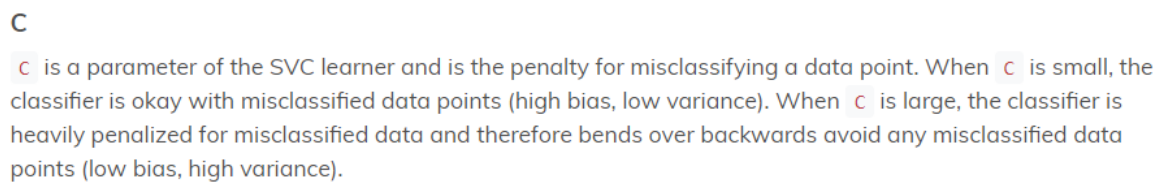
Radial Basis Function is a commonly used kernel in SVC:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

where $||\mathbf{x} - \mathbf{x}'||^2$ is the squared Euclidean distance between two data points $\mathbf{x}$ and $\mathbf{x}'$. If this doesn't make sense, Sebastian's book has a full description. However, for this tutorial, it is only important to know that an SVC classifier using an RBF kernel has two parameters: `gamma` and `C`.

Gamma

`gamma` is a parameter of the RBF kernel and can be thought of as the 'spread' of the kernel and therefore the decision region. When `gamma` is low, the 'curve' of the decision boundary is very low and thus the decision region is very broad. When `gamma` is high, the 'curve' of the decision boundary is high, which creates islands of decision-boundaries around data points. We will see this very clearly below.

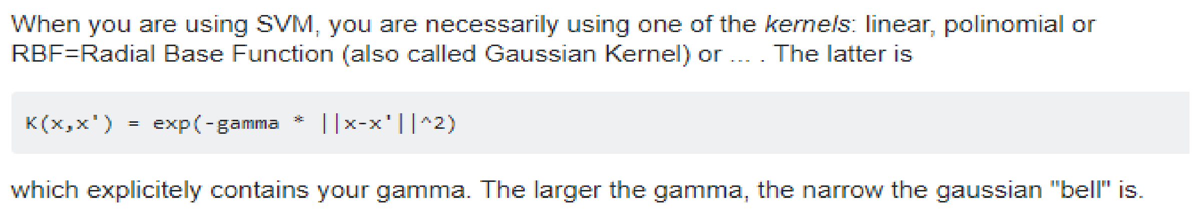


C

`C` is a parameter of the SVC learner and is the penalty for misclassifying a data point. When `C` is small, the classifier is okay with misclassified data points (high bias, low variance). When `C` is large, the classifier is heavily penalized for misclassified data and therefore bends over backwards avoid any misclassified data points (low bias, high variance).

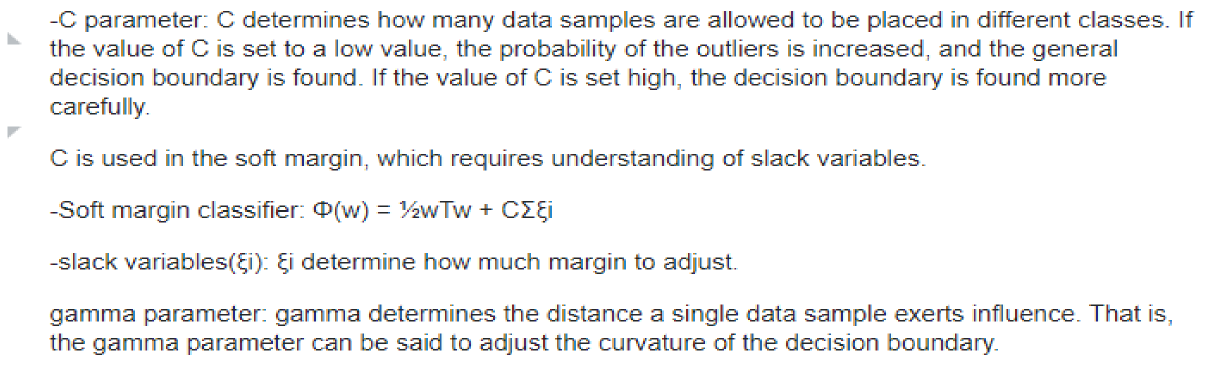
C - High C tries to minimize the misclassification of training data
And a low value tries to maintain a smooth classification. This makes sense to me.

C is the cost of misclassification as correctly. A large C gives you low bias and high variance. Low bias because you penalize the cost of misclassification a lot. A small C gives you higher bias and lower variance.


When you are using SVM, you are necessarily using one of the *kernels*: linear, polinomial or RBF=Radial Base Function (also called Gaussian Kernel) or ... . The latter is

```
K(x,x') = exp(-gamma * ||x-x'||^2)
```

which explicitely contains your gamma. The larger the gamma, the narrow the gaussian "bell" is.


-C parameter: C determines how many data samples are allowed to be placed in different classes. If the value of C is set to a low value, the probability of the outliers is increased, and the general decision boundary is found. If the value of C is set high, the decision boundary is found more carefully.

C is used in the soft margin, which requires understanding of slack variables.

-Soft margin classifier: Φ(w) = ½wTw + CΣξi

-slack variables(ξi): ξi determine how much margin to adjust.

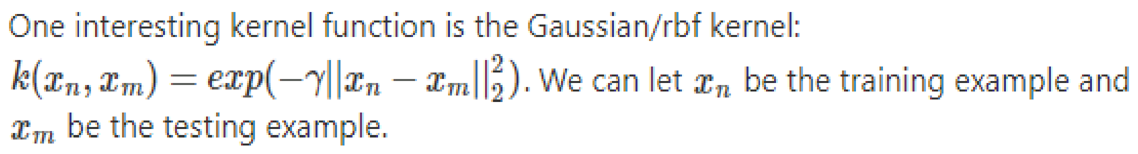
gamma parameter: gamma determines the distance a single data sample exerts influence. That is, the gamma parameter can be said to adjust the curvature of the decision boundary.

C (Cost) Parameter

Most of the time our data is not linearly separable To combat this, we can introduce the notion of “slack variables”, which are variables that allow us to relax the constraint/data point meaning that we no longer will have to correctly and confidently classify every single training point.
We can define a slack variable as a value ζ that, roughly, indicates how much we must move our point so that it is correctly and confidently classified. This makes sense - small slack variables means that we have a correct classification but aren’t very confident, while large slack variables means that we have not classified the point correctly.

We should also penalize our objective function whenever we use slack variables, otherwise we can just set the slack variables to extremely high values and solve the optimization.

This is where the variable C comes from - it is a hyperparameter that controls how much we penalize our use of slack variables. If we let C→0 then we don’t penalize slack variables at all, and as we increase C, we penalize our slack variables more and more - it’s essentially a trade-off between penalizing slack variables and obtaining a large margin for our SVM.


One interesting kernel function is the Gaussian/rbf kernel: $k(x_n, x_m) = exp(-\gamma||x_n - x_m||_2^2)$. We can let x_n be the training example and x_m be the testing example.

There's a few interesting things about this kernel:

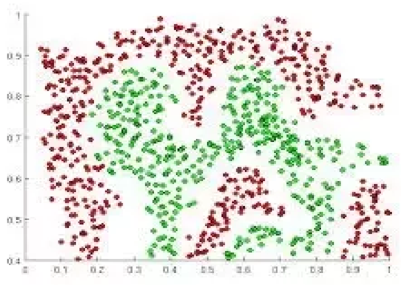
1. The value depends on the L2-squared distance between its inputs. One interpretation of this kernel can then be the idea that it measures how similar two feature vectors are.

2. The kernel can be interpreted as corresponding to a feature transformation onto an infinite dimensional space (this can be seen by writing out the Taylor polynomial expansion for the exponential function). This is another reason why kernels are useful - they can be computed in linear time as opposed to having to compute inner products between feature transformations, which usually have quadratic or worse time complexity in their size.

3. The hyperparameter γ controls the tradeoff between error due to bias and variance in your model. If you have a very large value of gamma, then even if your two inputs are quite "similar", the value of the kernel function will be small - meaning that the support vector x_n does not have much influence on the classification of the training example x_m. This allows the SVM to capture more of the complexity and shape of the data, but if the value of gamma is too large, then the model can overfit and be prone to low bias/high variance. On the other hand, a small value for gamma implies that the support vector has larger influence on the classification of x_m. This means that the model is less prone to overfitting, but you may risk not learning a decision boundary that captures the shape and complexity of your data. This leads to a high bias, low variance model.

<u>Gamma Parameter</u>

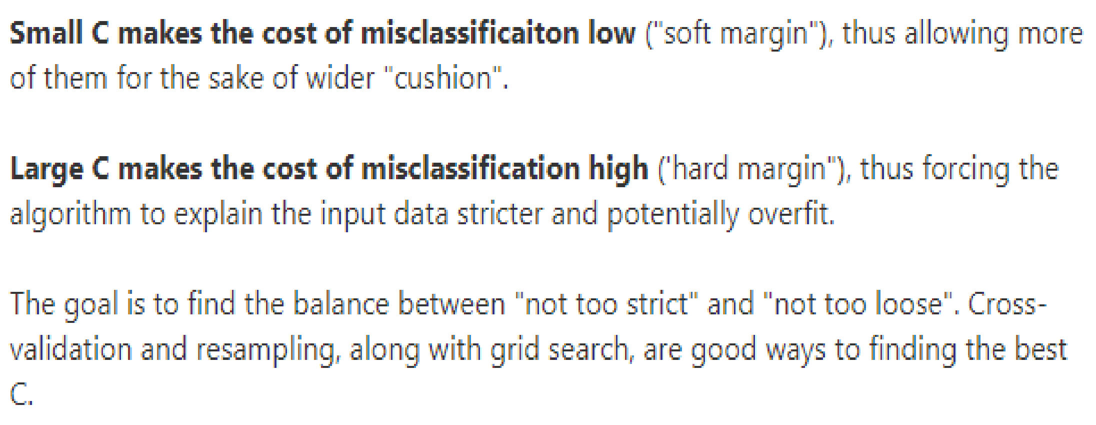
It is the parameter of a Gaussian Kernel (to handle non-linear classification).



They are not linearly separable in 2D so you want to transform them to a higher dimension where they will be linearly sepparable. Imagine "raising" the green points, then you can sepparate them from the red points with a plane (hyperplane)

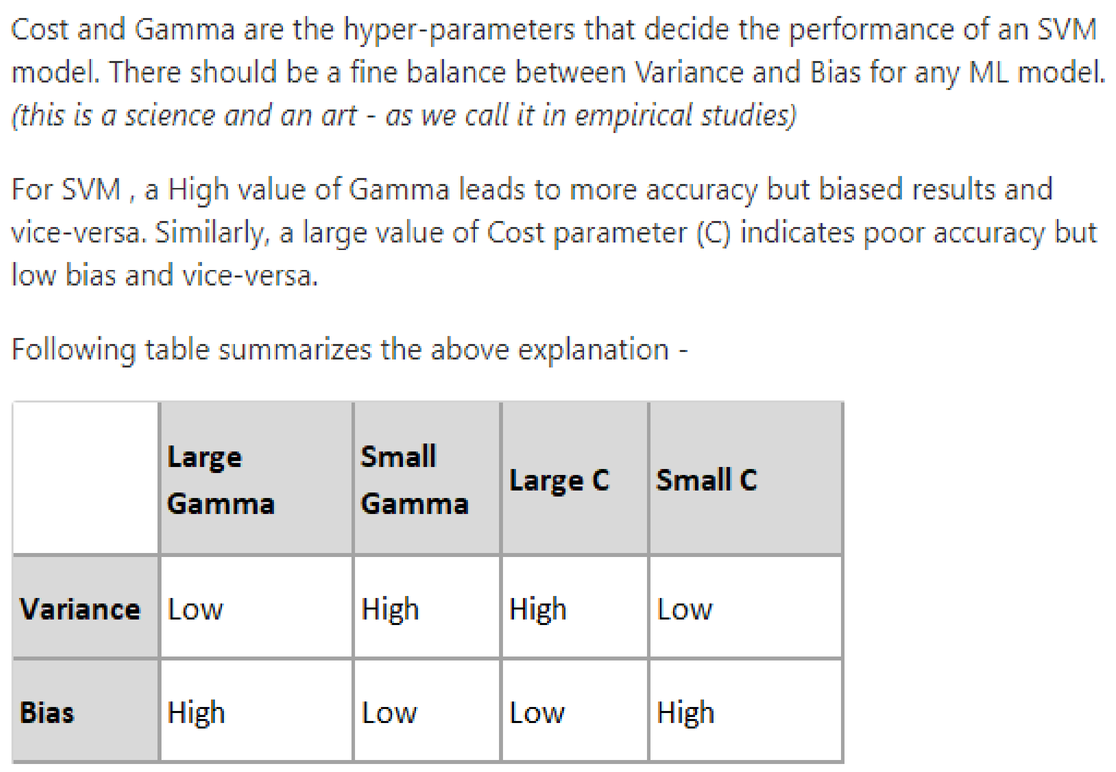
To "raise" the points you use the RBF kernel, gamma controls the shape of the "peaks" where you raise the points. A small gamma gives you a pointed bump in the higher dimensions, a large gamma gives you a softer, broader bump.

So a small gamma will give you low bias and high variance while a large gamma will give you higher bias and low variance.


Small C makes the cost of misclassificaiton low ("soft margin"), thus allowing more of them for the sake of wider "cushion".

Large C makes the cost of misclassification high ('hard margin"), thus forcing the algorithm to explain the input data stricter and potentially overfit.

The goal is to find the balance between "not too strict" and "not too loose". Cross-validation and resampling, along with grid search, are good ways to finding the best C.


Cost and Gamma are the hyper-parameters that decide the performance of an SVM model. There should be a fine balance between Variance and Bias for any ML model. *(this is a science and an art - as we call it in empirical studies)*

For SVM , a High value of Gamma leads to more accuracy but biased results and vice-versa. Similarly, a large value of Cost parameter (C) indicates poor accuracy but low bias and vice-versa.

Following table summarizes the above explanation -

	Large Gamma	**Small Gamma**	**Large C**	**Small C**
Variance	Low	High	High	Low
Bias	High	Low	Low	High

The C parameter 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. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable.

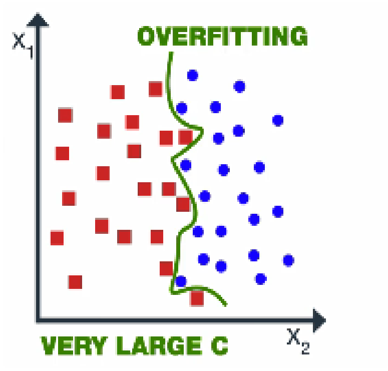
In Support Vector Machine, we need to choose different parameters to optimize our algorithms.

- Choice of kernel (Similarity function)
  - Linear kernel
  - Polynomial kernel
  - Logisitic/ Sigmoid kernel
  - Gaussian/RBF kernel
- Choice of parameter C
- Choice of Gamma (if using Gaussian kernel)

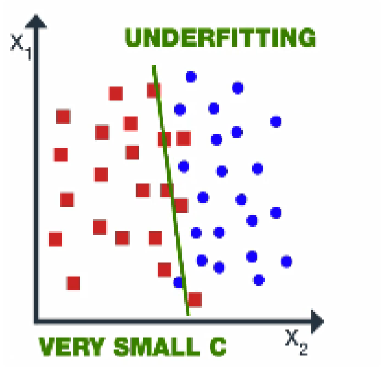
Parameter C

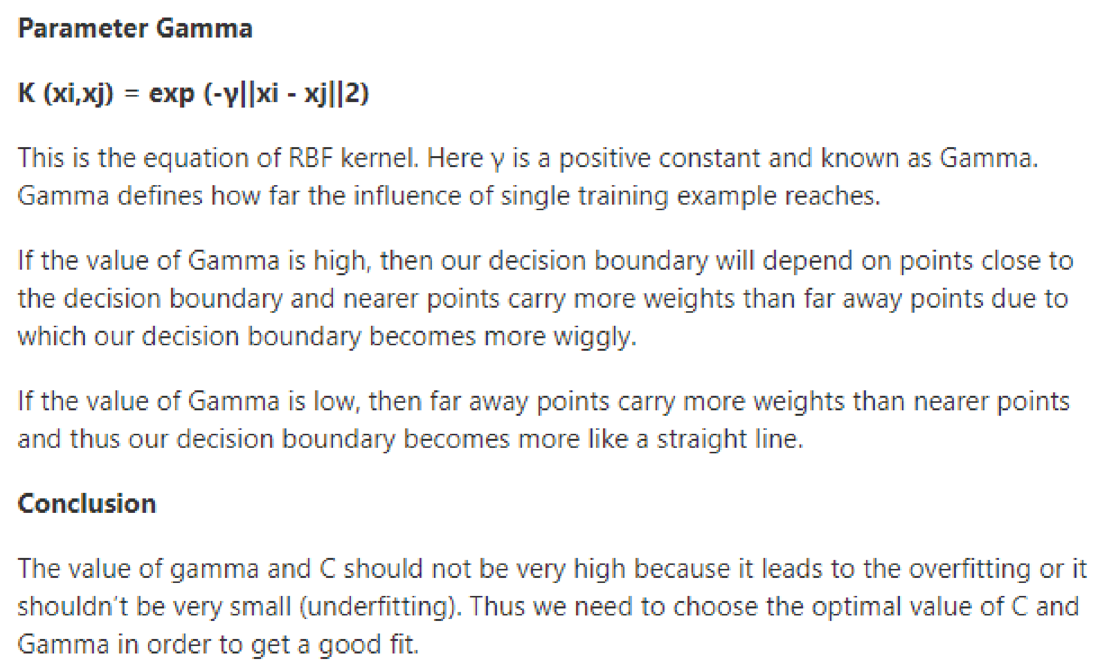
The C parameter controls the tradeoff between classification of training points accurately and a smooth decision boundary or in a simple word, it suggests the model to choose data points as a support vector.

If the value of C is large then model choose more data points as a support vector and we get the higher variance and lower bias, which may lead to the problem of overfitting.



If the value of C is small then model choose fewer data points as a support vector and get lower variance/high bias.





Parameter Gamma

K (xi,xj) = exp (-γ||xi - xj||2)

This is the equation of RBF kernel. Here γ is a positive constant and known as Gamma. Gamma defines how far the influence of single training example reaches.

If the value of Gamma is high, then our decision boundary will depend on points close to the decision boundary and nearer points carry more weights than far away points due to which our decision boundary becomes more wiggly.

If the value of Gamma is low, then far away points carry more weights than nearer points and thus our decision boundary becomes more like a straight line.

Conclusion

The value of gamma and C should not be very high because it leads to the overfitting or it shouldn't be very small (underfitting). Thus we need to choose the optimal value of C and Gamma in order to get a good fit.

When to use Logistic Regression and Random Forest

Logistic regression **work best/can be used** when number of input feature is less and RF **work best/can be used** when number of feature is more/high because if number of feature will be less then the tree created will have less node so it will not give good accuracy and since RF is a heavy model it require good number of feature to create a good tree otherwise there will be no depth in tree because of less node.

Random forest mostly take care of overfitting than LR, Not always but most of the times. When number of tree is more (800-1000) it won't overfit.

Measure Multicollinearity for Categorical Variable

In regression and tree models, it is required to meet assumptions of multicollinearity. Multicollinearity means "Independent variables are highly correlated to each other".

For **Categorical variables**, multicollinearity can be detected with **Spearman Rank Correlation Coefficient (Ordinal variables)** and **Chi-Square Test (Nominal variables).**

For a **Categorical and a Continuous variable**, multicollinearity can be measured by **t-test** (if the categorical variable has 2 categories) or **ANOVA** (more than 2 categories).

For a **Continuous variable**, multicollinearity can be measured by **Pearson Correlation Coefficient**.

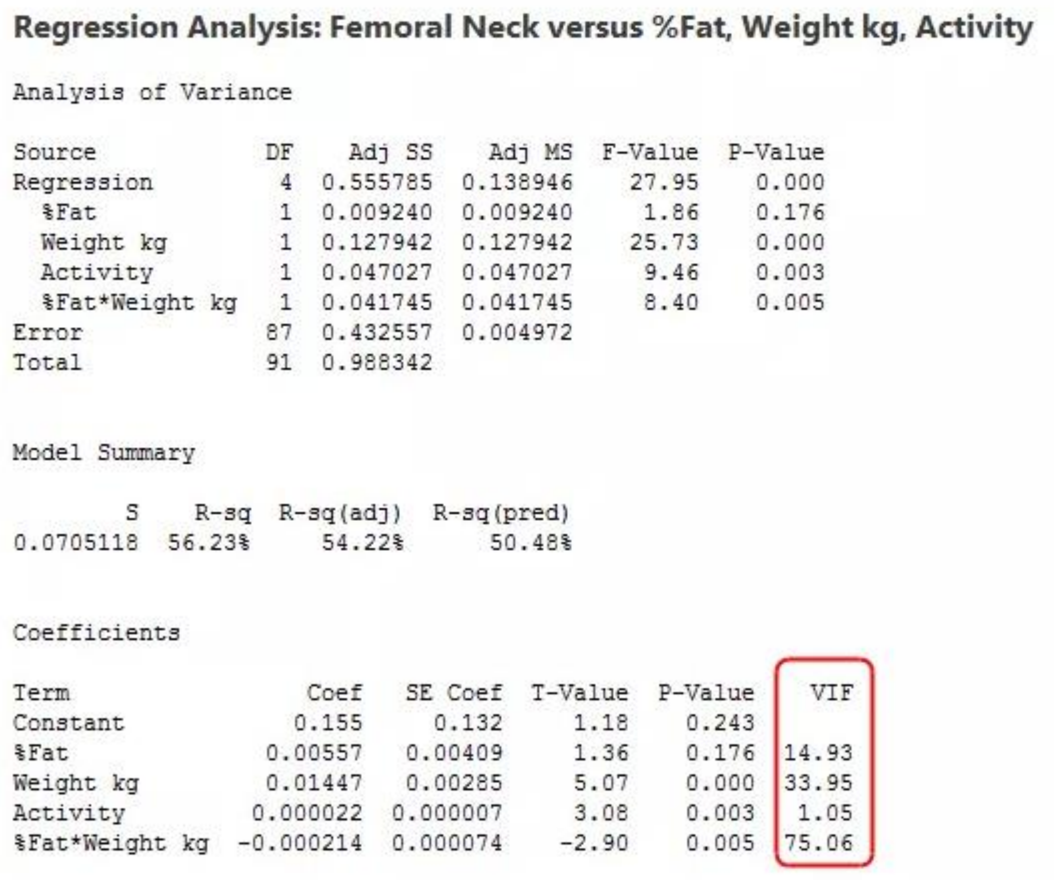
First consider the most widely-used diagnostic for multicollinearity, the variance inflation factor (VIF). The VIF may be calculated for each predictor by doing a linear regression of that predictor on all the other predictors, and then obtaining the R^2 from that regression. The VIF is just $1/(1-R^2)$.

It's called the variance inflation factor because it estimates how much the variance of a coefficient is "inflated" because of linear dependence with other predictors. Thus, a VIF of 1.8 tells us that the variance (the square of the standard error) of a particular coefficient is 80% larger than it would be if that predictor was completely uncorrelated with all the other predictors.

How Can I Deal With Multicollinearity

- **Remove highly correlated predictors from the model.** If you have two or more factors with a high VIF, remove one from the model. Because they supply redundant information, removing one of the correlated factors usually doesn't drastically reduce the R-squared. Consider using stepwise regression, best subsets regression, or specialized knowledge of the data set to remove these variables. Select the model that has the highest R-squared value.

- Use Partial Least Squares Regression (PLS) **or** Principal Components Analysis, regression methods that cut the number of predictors to a smaller set of uncorrelated components.



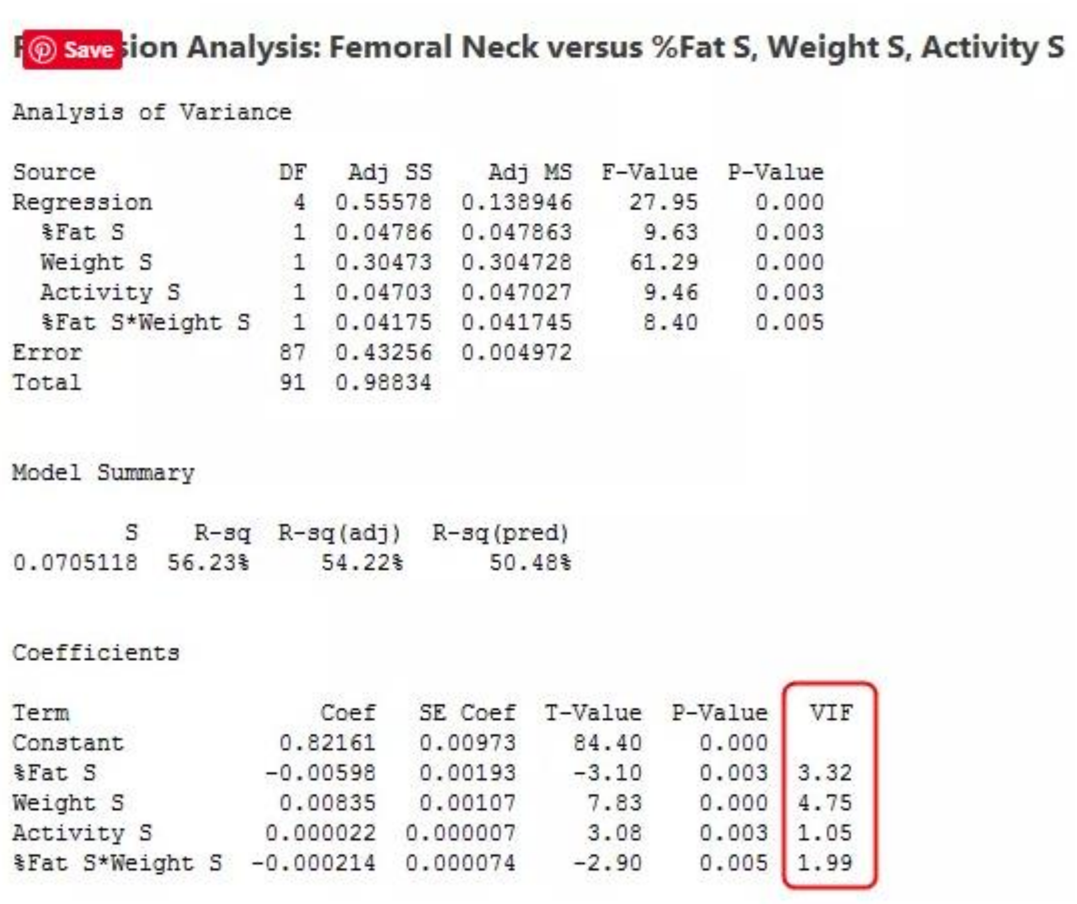
Regression Analysis: Femoral Neck versus %Fat, Weight kg, Activity

```
Analysis of Variance

Source            DF    Adj SS    Adj MS  F-Value  P-Value
Regression         4  0.555785  0.138946    27.95    0.000
  %Fat             1  0.009240  0.009240     1.86    0.176
  Weight kg        1  0.127942  0.127942    25.73    0.000
  Activity         1  0.047027  0.047027     9.46    0.003
  %Fat*Weight kg   1  0.041745  0.041745     8.40    0.005
Error             87  0.432557  0.004972
Total             91  0.988342


Model Summary

        S    R-sq  R-sq(adj)  R-sq(pred)
0.0705118  56.23%     54.22%      50.48%


Coefficients

Term                  Coef   SE Coef  T-Value  P-Value    VIF
Constant             0.155     0.132     1.18    0.243
%Fat               0.00557   0.00409     1.36    0.176  14.93
Weight kg          0.01447   0.00285     5.07    0.000  33.95
Activity          0.000022  0.000007     3.08    0.003   1.05
%Fat*Weight kg   -0.000214  0.000074    -2.90    0.005  75.06
```

Center the Independent Variables to Reduce Structural Multicollinearity

Centering the variables is a simple way to reduce structural multicollinearity. Centering the variables is also known as standardizing the variables by subtracting the mean. This process involves calculating the mean for each continuous independent variable and then subtracting the mean from all observed values of that variable. Then, use these centered variables in your model.

There are other standardization methods, but the advantage of just subtracting the mean is that the interpretation of the coefficients remains the same. The coefficients continue to represent the mean change in the dependent variable given a 1 unit change in the independent variable.



ion Analysis: Femoral Neck versus %Fat S, Weight S, Activity S

```
Analysis of Variance

Source             DF   Adj SS    Adj MS  F-Value  P-Value
Regression          4  0.55578  0.138946    27.95    0.000
  %Fat S            1  0.04786  0.047863     9.63    0.003
  Weight S          1  0.30473  0.304728    61.29    0.000
  Activity S        1  0.04703  0.047027     9.46    0.003
  %Fat S*Weight S   1  0.04175  0.041745     8.40    0.005
Error              87  0.43256  0.004972
Total              91  0.98834

Model Summary

        S    R-sq  R-sq(adj)  R-sq(pred)
0.0705118  56.23%     54.22%      50.48%

Coefficients

Term                  Coef   SE Coef  T-Value  P-Value   VIF
Constant           0.82161   0.00973    84.40    0.000
%Fat S            -0.00598   0.00193    -3.10    0.003  3.32
Weight S           0.00835   0.00107     7.83    0.000  4.75
Activity S        0.000022  0.000007     3.08    0.003  1.05
%Fat S*Weight S  -0.000214  0.000074    -2.90    0.005  1.99
```

Multicollinearity occurs when two or more variables are linearly interdependent. Including such variables might result in a biased model which will perform nicely in the validation set but completely fail in out-of-time validation or in production.

To avoid multi-collinearity, the best and the standard way is to remove the identified variables.

But there are scenarios when you need to retain some of these variables (which are linearly dependent) in your final training set for building the model. In such cases, you may consider a linear combination of these variables as a new variable and drop all the identified variables.

Suppose, you identify a strong correlation between the variables X1 and X2. In this case you want to discard one of these variables. But the business thinks that both these variables are important and should be utilized. In such a case, you may drop X1, X2 but introduce a new variable Z = X1 + X2 (or any other linear combination)

One way to address multicollinearity is to center the predictors that is subtract the mean of one series from each value. Ridge regression can also be used when data is highly collinear. Finally sequential regression can help in understanding cause-effect relationships between the predictors, in conjunction with analysing the time sequence of the predictor events.

Pearson's correlation coefficient measures the strength of the linear relationship between two variables on a **continuous** scale. For example, the relationship between height and weight of a person or price of a house to its area.

The correlation coefficient, r (rho), takes on the values of −1 through +1. Values of −1 or +1 indicate a perfect linear relationship between the two variables, and a value of 0 indicates no linear association. (There could be non-linear association).

A value between 0 and +1 means the variables are positively correlated i.e. increase in the value of one variable leads to an increase in another. For example, a house with more sqft would have a higher price.

A value between -1 and 0 means the variables are negative correlated i.e. increase in the value of one variable leads to decrease in another.

Any value that is not 0 or −1 or +1 indicate a linear relationship, although not a perfect linear relationship.

Spearman rank-order correlation coefficient measures the measure of the strength and direction of association that exists between two variables. The test is used for either **ordinal variables** or for continuous data that has failed the assumptions necessary for conducting the Pearson's product-moment correlation. For example, you could use a Spearman's correlation to understand whether there is an association between exam performance and time spent revising; whether there is an association between depression and length of unemployment.

It is denoted by r_s, can take values from +1 to -1. A r_s of +1 indicates a perfect association of ranks, a r_s of zero indicates no association between ranks and a r_s of -1 indicates a perfect negative association of ranks. The closer r_s is to zero, the weaker the association between the ranks.

Chi-Square test is used to determine the association between two **categorical** variables. The chi-square test, unlike Pearson's correlation coefficient or Spearman rho, is a measure of the significance of the association rather than a measure of the strength of the association.

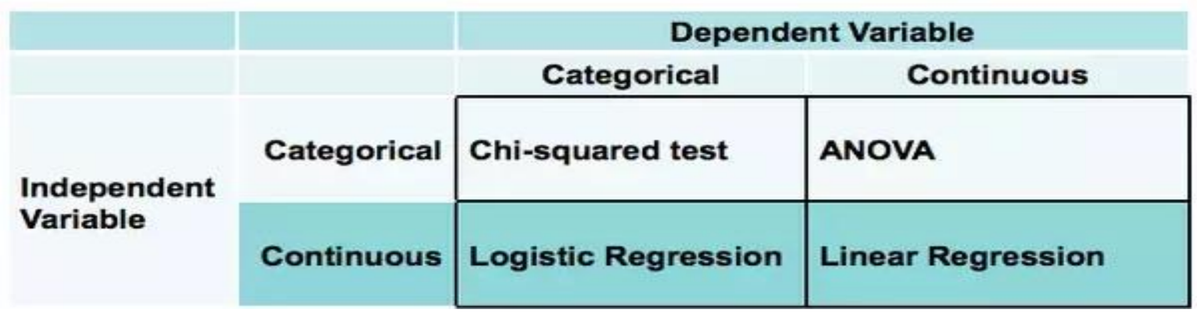
This test should be used to determine whether variables like education, political views and other preferences vary based on demographic factors like gender, race and religion. Or, to verify the influence of gender on purchase decisions and patterns.

The Chi-square test is based on hypothesis testing, and the hypothesis are defined as:

Null Hypothesis: There is no relationship between the two variables of interest.

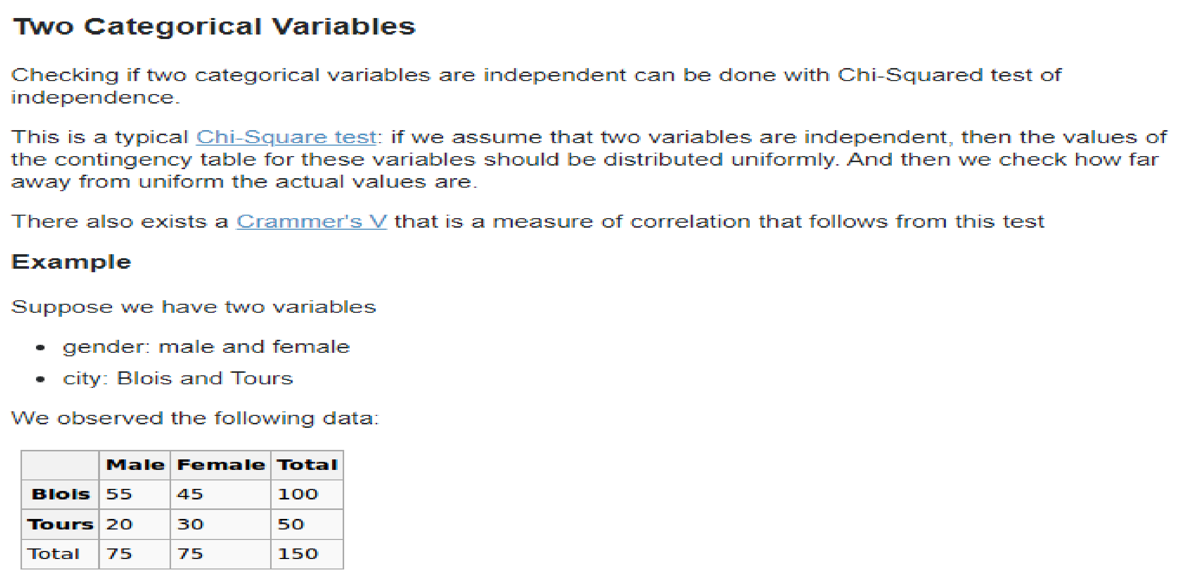
Alternate Hypothesis: There is a significant relationship between the variables of interest.

The test involves comparing the observed frequencies of the variables to the expected frequencies which are calculated assuming the variables were independent of each other. If the p-value is less than the critical value, the test is significant. This means that there is an association between two variables. Otherwise, the test is nonsignificant and implies that there is not an association.



		Dependent Variable	
		Categorical	Continuous
Independent Variable	Categorical	Chi-squared test	ANOVA
	Continuous	Logistic Regression	Linear Regression

1. Two **categorical** variable use **Chi-square**
2. Two or more **quantitative** variable (Continuous or discrete) use **Pearson correlation (r)**,
3. **One categorical and one quantitative variable** (Continuous or discrete) use **ANOVA**.



Two Categorical Variables

Checking if two categorical variables are independent can be done with Chi-Squared test of independence.

This is a typical Chi-Square test: if we assume that two variables are independent, then the values of the contingency table for these variables should be distributed uniformly. And then we check how far away from uniform the actual values are.

There also exists a Crammer's V that is a measure of correlation that follows from this test

Example

Suppose we have two variables

- gender: male and female
- city: Blois and Tours

We observed the following data:

	Male	Female	Total
Blois	55	45	100
Tours	20	30	50
Total	75	75	150

Are gender and city independent? Let's perform a Chi-Squred test. Null hypothesis: they are independent, Alternative hypothesis is that they are correlated in some way.

Under the Null hypothesis, we assume uniform distribution. So our expected values are the following

	Male	**Female**	**Total**
Blois	50	50	100
Tours	25	25	50
Total	75	75	150

So we run the chi-squared test and the resulting p-value here can be seen as a measure of correlation between these two variables.

To compute Crammer's V we first find the normalizing factor chi-squared-max which is typically the size of the sample, divide the chi-square by it and take a square root

- $\nu = \sqrt{\frac{\chi^2}{\chi^2_{\max}}}$
- with $\chi^2_{\max} = N \times (\min(N, P) - 1)$ where
  - N is the number of tuples and P the number of attributes

R

```
tbl = matrix(data=c(55, 45, 20, 30), nrow=2, ncol=2, byrow=T)
dimnames(tbl) = list(City=c('B', 'T'), Gender=c('M', 'F'))

chi2 = chisq.test(tbl, correct=F)
c(chi2$statistic, chi2$p.value)
```

Here the p value is 0.08 - quite small, but still not enough to reject the hypothesis of independence. So we can say that the "correlation" here is 0.08

We also compute V:

```
sqrt(chi2$statistic / sum(tbl))
```

And get 0.14 (the smaller v, the lower the correlation)

Consider another dataset

```
    Gender
City  M  F
   B 51 49
   T 24 26
```

For this, it would give the following

For this, it would give the following

```
tbl = matrix(data=c(51, 49, 24, 26), nrow=2, ncol=2, byrow=T)
dimnames(tbl) = list(City=c('B', 'T'), Gender=c('M', 'F'))

chi2 = chisq.test(tbl, correct=F)
c(chi2$statistic, chi2$p.value)

sqrt(chi2$statistic / sum(tbl))
```

The p-value is 0.72 which is far closer to 1, and v is 0.03 - very close to 0

Categorical vs Numerical Variables

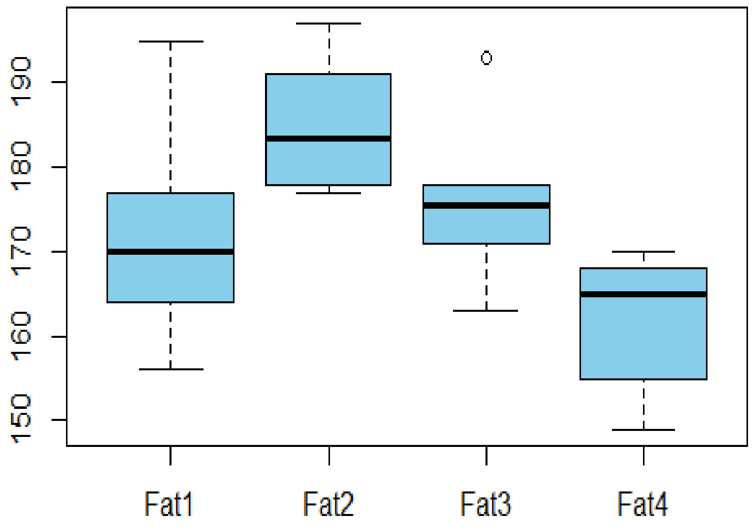
For this type we typically perform One-way ANOVA test: we calculate in-group variance and intra-group variance and then compare them.

Example

We want to study the relationship between absorbed fat from donuts vs the type of fat used to produce donuts (example is taken from here)

Example

We want to study the relationship between absorbed fat from donuts vs the type of fat used to produce donuts (example is taken from here)



Is there any dependence between the variables? For that we conduct ANOVA test and see that the p-value is just 0.007 - there's no correlation between these variables.

Fat1	Fat2	Fat3	Fat4
164	178	175	155
172	191	193	166
168	197	178	149
177	182	171	164
156	185	163	170
195	177	176	168

R

```
t1 = c(164, 172, 168, 177, 156, 195)
t2 = c(178, 191, 197, 182, 185, 177)
t3 = c(175, 193, 178, 171, 163, 176)
t4 = c(155, 166, 149, 164, 170, 168)

val = c(t1, t2, t3, t4)
fac = gl(n=4, k=6, labels=c('type1', 'type2', 'type3', 'type4'))

aov1 = aov(val ~ fac)
summary(aov1)
```

Output is

```
            Df Sum Sq Mean Sq F value  Pr(>F)
fac          3   1636   545.5   5.406 0.00688 **
Residuals   20   2018   100.9
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

So we can take the p-value as the measure of correlation here as well.

```
# Model performance
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  R2 = R2(predictions, test.data$medv)
)
```

Detecting multicollinearity

The R function `vif()` [car package] can be used to detect multicollinearity in a regression model:

```
car::vif(model1)
```

```
##    crim      zn   indus    chas     nox      rm     age     dis     rad
##    1.87    2.36    3.90    1.06    4.47    2.01    3.02    3.96    7.80
##     tax ptratio   black   lstat
##    9.16    1.91    1.31    2.97
```

In our example, the VIF score for the predictor variable `tax` is very high (VIF = 9.16). This might be problematic.

For VIF calculation usdm can also be package (I need to install "usdm")

```
library(usdm)
df = # Data Frame
vif(df)
```