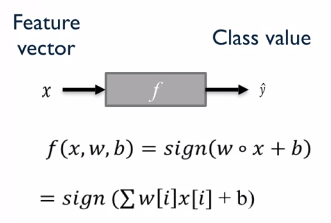
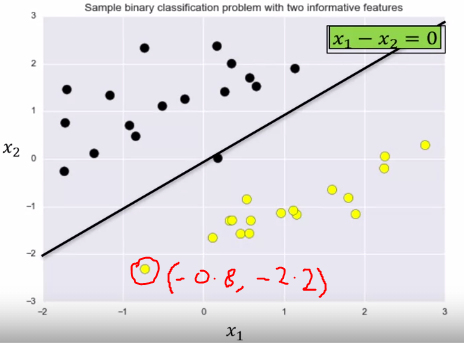
**Support Vector Machines:**

**Linear Classifiers:**

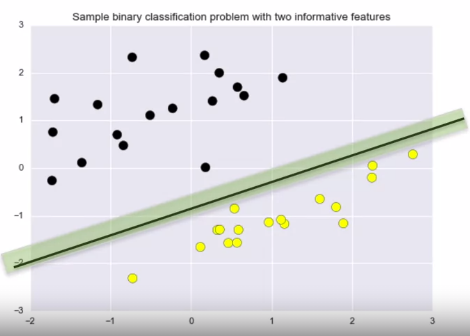
Linear classifiers take linear functions and apply a **sign** (not a sine function) function to produce a binary output, the two binary output respond to the two target labels.



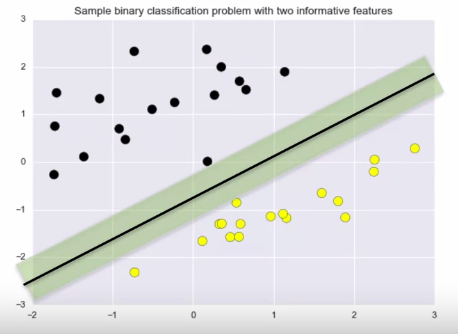
e.g. Imagine we have the following data and the following linear relationship:

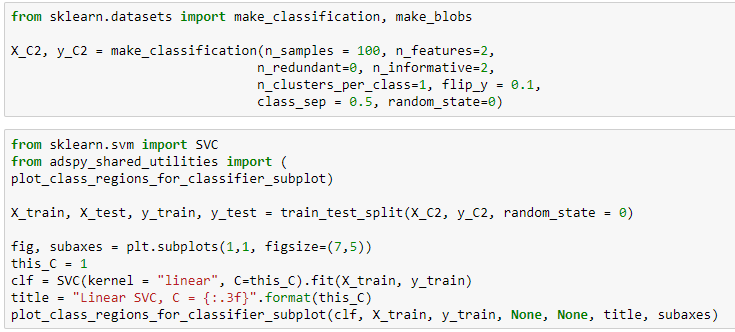


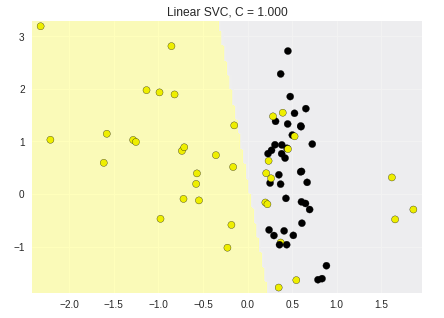
A **Classification Margin** is the maximum width a **decision boundaries** margin can be increase before it hits a data point. E.g.:



A **Linear Support Vector Machine (LSVM)** is defined as the best classifier that has the **largest** **classification margin**. The classifier shown below is a LSVM, compare it to the figure above and we can see that the classification margin is much bigger.



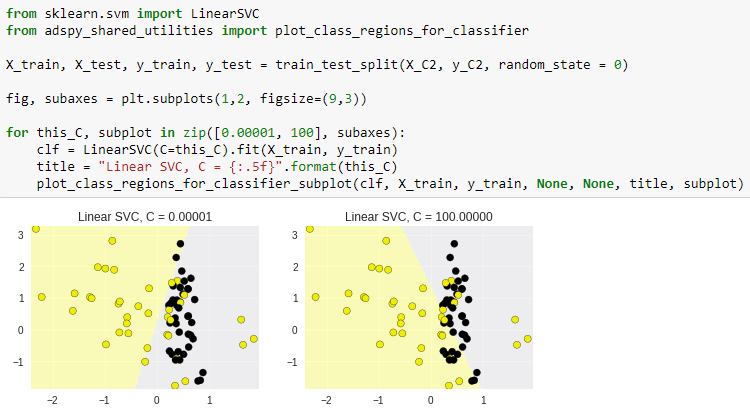




**Regularization Parameter C:**

* Large C results in less regularization which results in a model that fits the training data well, but over generalizes the data.
* Small C results in more regularization which results in a model that generalizes well but might not perform well on training data.

Having a **large value of C** makes the **decision boundary** try and find the optimal line so that the training data can be predicted as well as possible, this normally results in a **small classification margin**. Having **small values of C** will make the model try and **maximise the width** of the **classification margins** but might not find an optimal **decision boundary** for the training data.



**Linear Models: Pros and Cons:**

**Pros:**

* Simple and easy to train
* Fast predictions
* Scales well to very large datasets
* Predictions are relatively easy to interpret.

**Cons:**

* For lower dimensional data (fewer features) other models may have superior generalization.
* Often the data might not be linearly separable.

**Kernelized Support Vector Machines:**

We have seen that for easily separable data LSVM work well at finding a decision boundary and the classification margins. However, when variables are not easily separable a LSVM will not be able to classify this data well.

**SVM** can be used for both **Classification** and **Regression** problems.

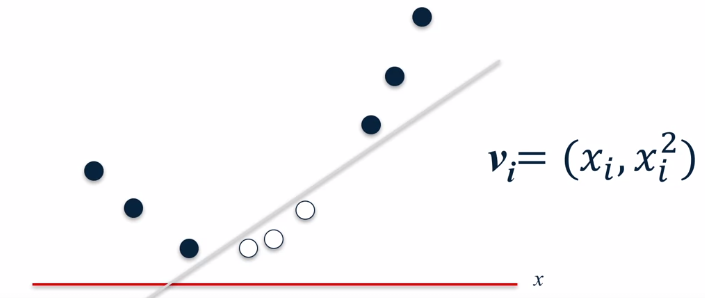
What SVMs do is they take the initial data and they transform it into a **higher dimension** in the hope that in this higher dimension the variables will be easier to separate.

If data is hard to separate in its base dimensions, then the SVM will increase its dimension by 1 and this new dimension will be the square of the original value. Each increase in dimension the original data will be but to the power of this new dimension. e.g.

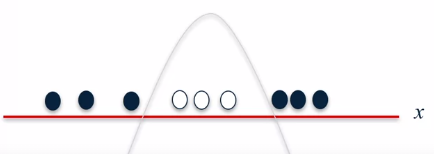
The higher dimensional data is then fit with a new **linear decision boundary** and **classification margin**. If we then take this linear decision boundary equation and convert it back to the original dimension of the data, we would find that the line is now a curve and still accurately splits the data.



The above 1d data can’t be classified using a linear classifier as the data can’t be split. We instead transform the data into 2d by squaring the original 1d values and adding it as a feature.

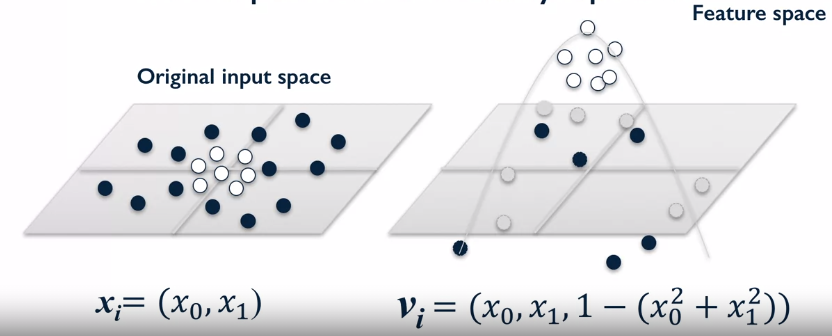


We would then take this linear equation that separates the variables nicely and transform it back to the original input space:

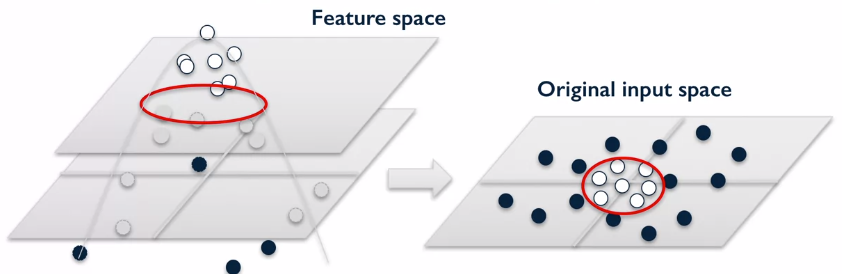


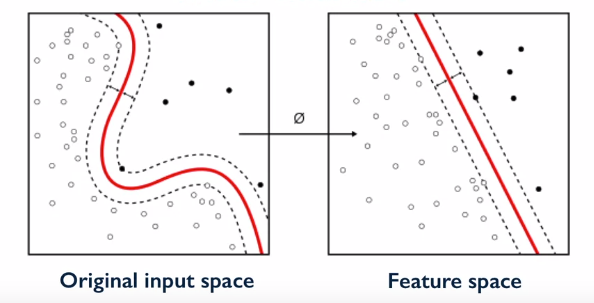
**2D SVM Example:**

We now have 2 features in our data to predict a target value.



With this transformation it is easy to find a hyperplane that easily separates the classes e.g. z=3.



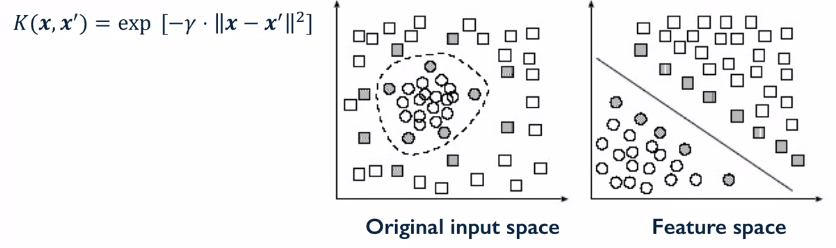


We transform the data into a higher dimension until the classes become easily separable by a linear equation. We then take this linear equation and transform it back into the original input space which will result in a very different function that will separate the classes well.

The different **Kernels** in the SVM result in different data transformations, in this module we're going to focus on **radial basis** and **polynomial kernels**. The kernel function in SVM find the similarity between two data points in the new feature space.

**Radial Basis Function Kernel:**

This kernel finds a linear decision boundary in a higher dimension and then transforms this equation back to the original data to form a **circular/elliptical boundary**.

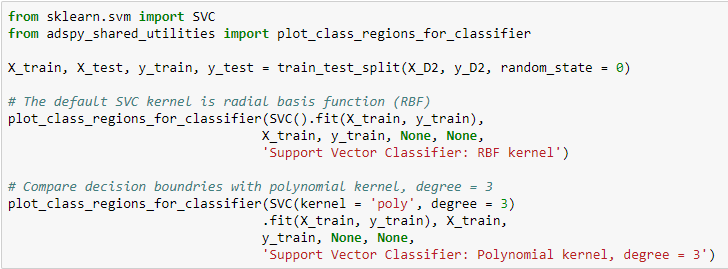


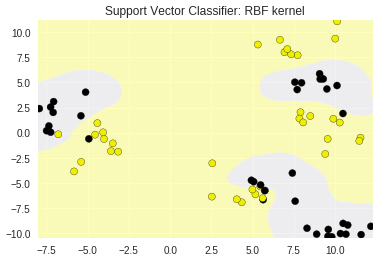
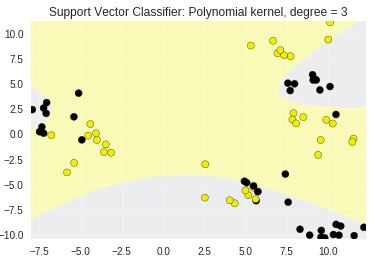
This is an exponentially decaying function of the distances between the vectors and the original input space. where **x'** is the transformed x points. || x – x’|| is just the distance between the two vectors.

So just as we saw with the simple 1D and 2D examples earlier, the kernelized support vector machine tries to find the decision boundary with maximum margin between classes using a linear classifier in the transformed feature space not the original input space.

The linear decision boundary learn feature space by linear SVM corresponds to a non-linear decision boundary in the original input space. So, in this example, an ellipse like closed region in the input space. Now, one of the mathematically remarkable things about kernelized support vector machines, something referred to as the **kernel trick**, is that internally, the algorithm doesn't have to perform this actual transformation on the data points to the new high dimensional feature space.

Instead, the kernelized SVM can compute these more complex decision boundaries just in terms of similarity calculations between pairs of points in the high dimensional space where the transformed feature representation is implicit.



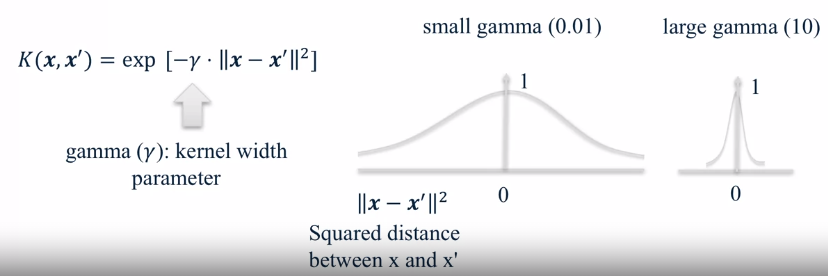
 

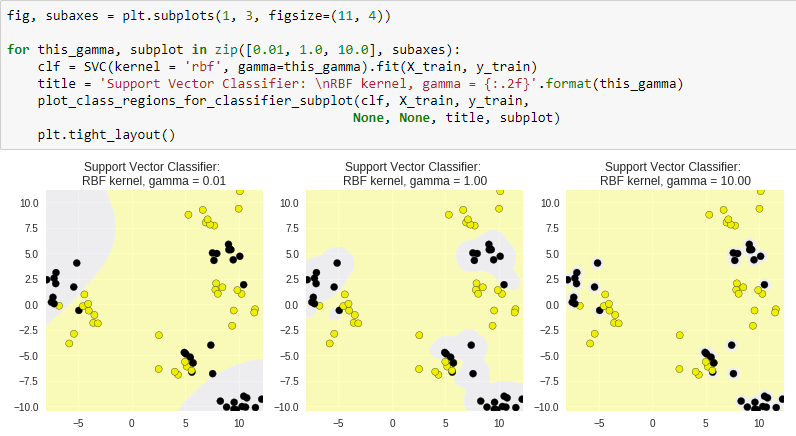
Here we can see how the different choice of kernels effects the decision boundaries in the original feature space. Just to recap, the SVM increases the dimensions of the data until there is a clear enough separation for a LSMV to produce a linear decision boundary with maximum classification margins. This LSMV equation for the decision boundary is then converted back to the original feature dimension with the specified kernel function e.g. radial or polynomial in the cases above.

**Gamma Parameter:**

**Small gamma** means a **larger similarity radius**. So that **points farther apart are considered similar**. Which results in more points being group together and **smoother decision boundaries**.

On the other hand, for **larger values of gamma**, the kernel value decays more quickly, and points have to be very close to be considered similar. This results in more complex, tightly constrained decision boundaries.

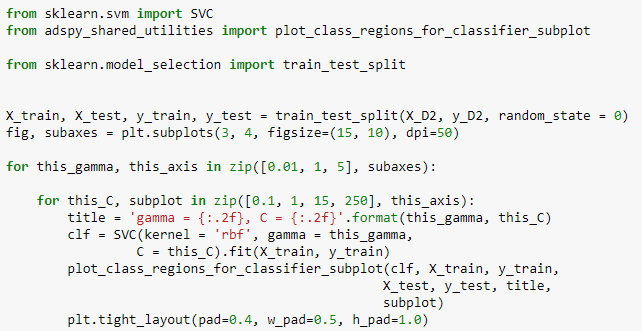


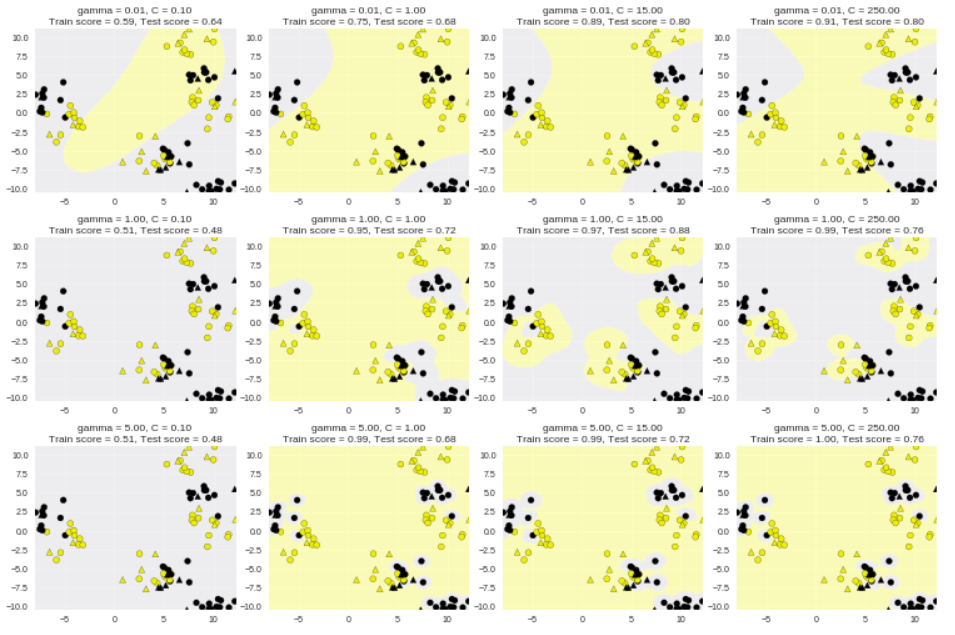


We can see that with a small value of gamma the decision boundary is simple and might generalize well. With larger values of gamma, we can see that the model in very complex and fits the training data extremely well, meaning that it might not generalize well to new data points.

**C Parameter:**

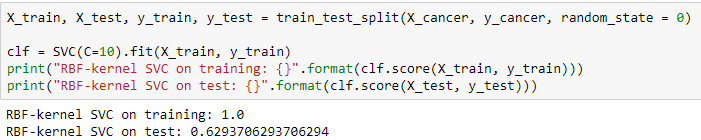
This works in the same way as for linear classification, **small values of C allow for large classification margins** that generalize well to new data points. Whereas **large values of C allow for a more optimal decision boundary** to be used to fit the training data better.

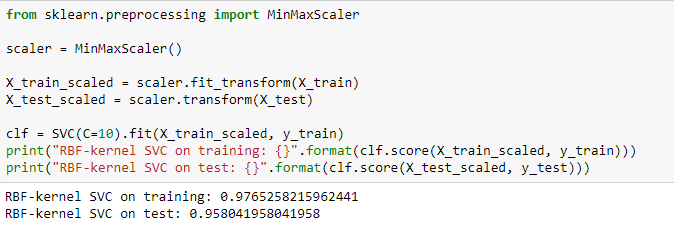




**Don’t forget to scale your data!!**

The following data is from a large data set within Scikit-learn:





By scaling the data, we can see that the testing set score improved dramatically.

**Pros and Cons of SVM:**

**Pros:**

* Performs well on a range of datasets, from text to images.
* Very versatile as the kernels allow for a lot of customization.
* Works well for data with few and many features.

**Cons:**

* SVM are calculation intensive and can take a long time to run if the df is large.
* Requires careful normalization and parameter tuning for good results.
* They don’t provide direct probability estimates for predictions. (can work around this using Platt scaling).
* Their results can be hard to interoperate.