Published in Towards Data Science

Soner Yıldırım Follow

Jun 1, 2020 · 6 min read · ✨ · Listen

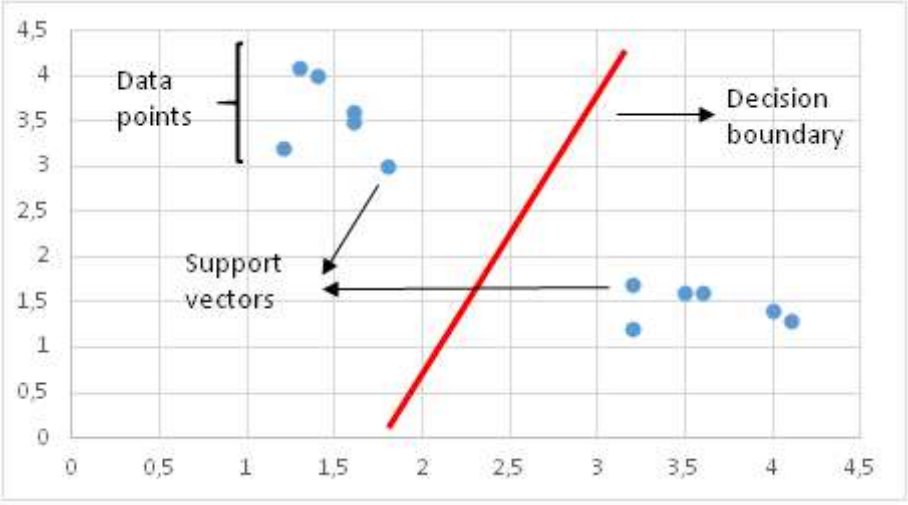
Hyperparameter Tuning for Support Vector Machines — C and Gamma Parameters

Understand the hyperparameters for Support Vector Machines

well. In this post, we dive deep into two important parameters of support vector machines which are **C** and **gamma**. So I will assume you have a basic understanding of the algorithm and focus on these parameters.

Most of the machine learning and deep learning algorithms have some parameters that can be adjusted which are called **hyperparameters**. We need to set hyperparameters before we train the models. Hyperparameters are very critical in building robust and accurate models. They help us find the balance between bias and variance and thus, prevent the model from overfitting or underfitting. To be able to adjust the hyperparameters, we need to understand what they mean and how they change a model. It would be a tedious and never-ending task to randomly trying a bunch of hyperparameter values.

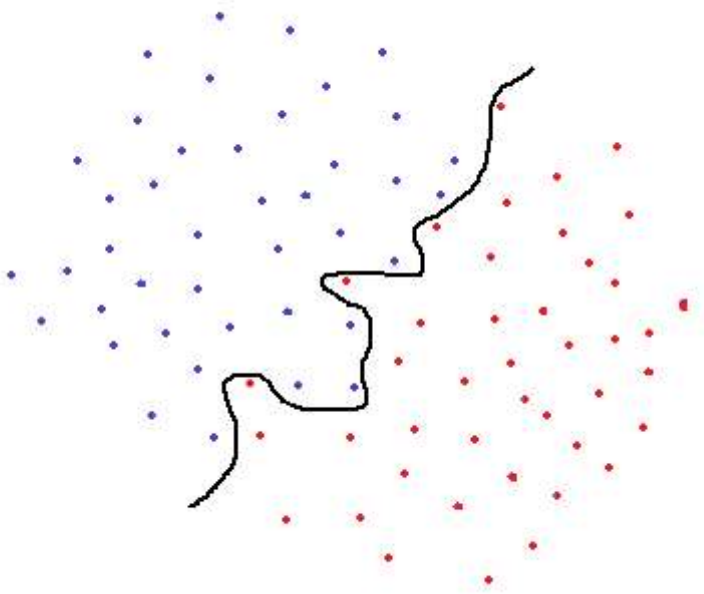
We emphasized the importance of hyperparameters. Let's start our discussion on **C** and **gamma**. SVM creates a **decision boundary** which makes the distinction between two or more classes. How to draw or determine the decision boundary is the most critical part in SVM algorithms. When the data points in different classes are linearly separable, it is an easy task to draw a decision boundary.



Linearly separable data points

However, real data is noisy and not linearly separable in most cases. A standard SVM

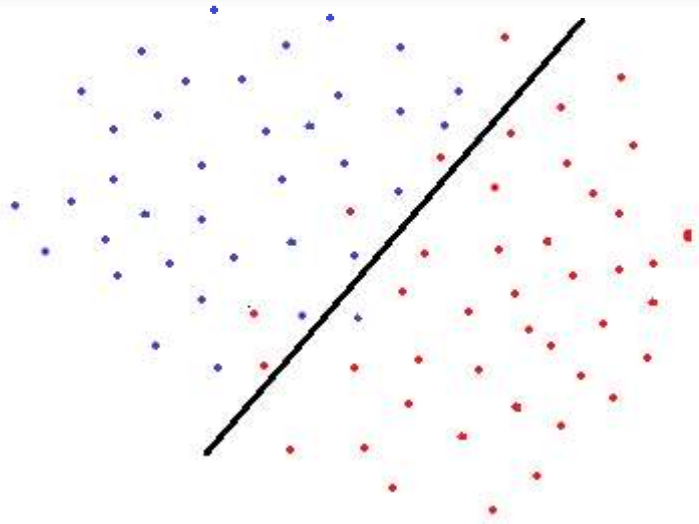
Consider the data points in a 2-dimensional space below:



Standard SVM

A standard SVM would try to separate blue and red classes by using the black curve line as a decision boundary. However, this is a too specific classification and highly likely to end up overfitting. An overfit SVM achieves a high accuracy with training set but will not perform well on new, previously unseen examples. This model would be very sensitive to noise and even very small changes in data point values may change the classification results. The SVM that uses this black line as a decision boundary is not generalized well to this dataset.

To overcome this issue, in 1995, Cortes and Vapnik, came up with the idea of "**soft margin**" SVM which allows some examples to be misclassified or be on the wrong side of decision boundary. Soft margin SVM often result in a better generalized model. In our example, the decision boundary for soft margin SVM might look like the black straight line as below:



Soft margin SVM

There are some misclassified points but we end up having a more generalized model. When determining the decision boundary, a soft margin SVM tries to solve an optimization problem with the following goals:

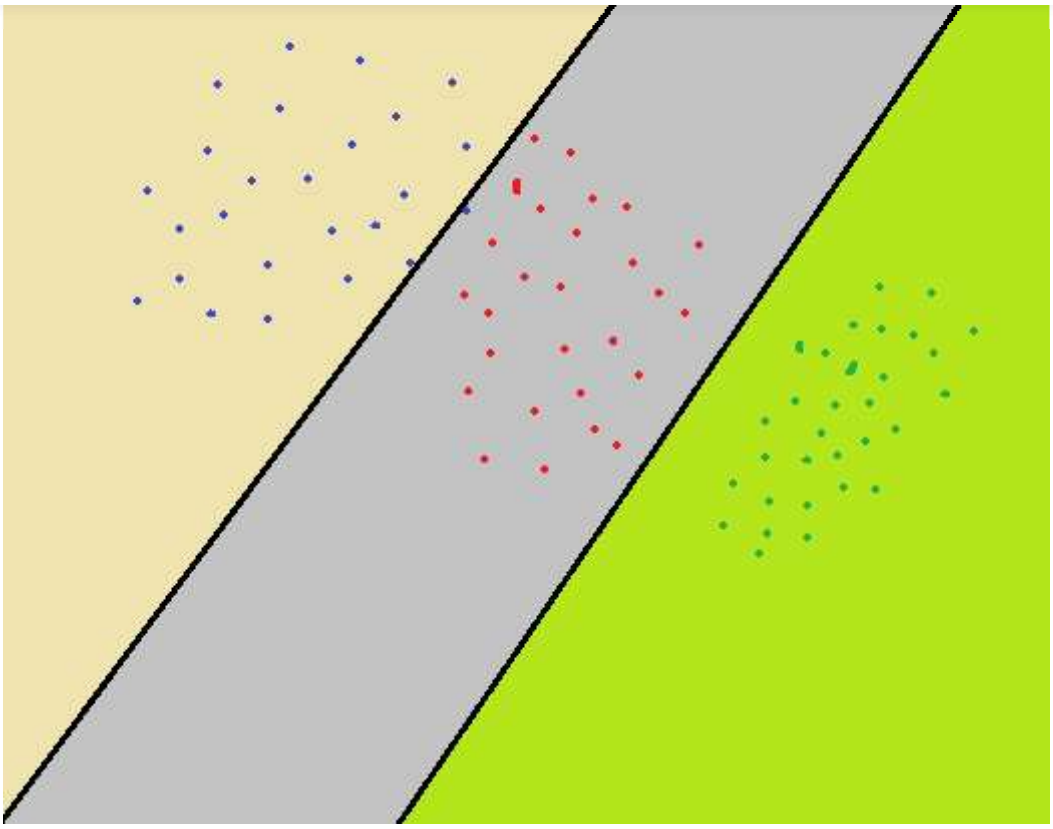
- Increase the distance of decision boundary to classes (or support vectors)
- Maximize the number of points that are correctly classified in the training set

There is obviously a trade-off between these two goals. Decision boundary might have to be very close to one particular class to correctly label all data points in training set. However, in this case, accuracy on test dataset might be lower because decision boundary is too sensitive to noise and to small changes in the independent variables. On the other hand, a decision boundary might be placed as far as possible to each class with the expense of some misclassified exceptions. This trade-off is controlled by **c parameter.**

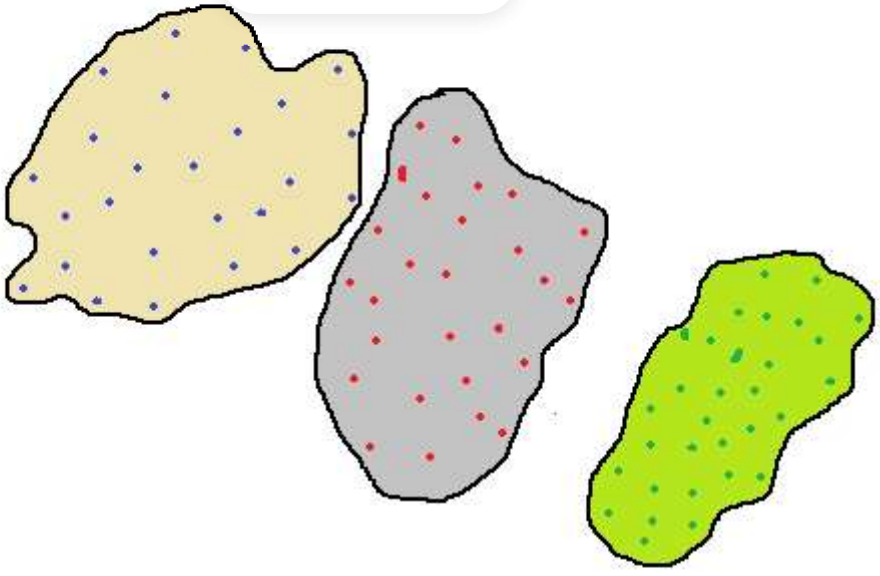
C parameter adds a penalty for each misclassified data point. If c is small, the penalty for misclassified points is low so a decision boundary with a large margin is chosen at the expense of a greater number of misclassifications. If c is large, SVM tries to minimize the number of misclassified examples due to high penalty which results in a

functions so that they become linearly separable. Kernel function is kind of a similarity measure. The inputs are original features and the output is a similarity measure in the new feature space. Similarity here means a degree of closeness. It is a costly operation to actually transform data points to a high-dimensional feature space. The algorithm does not actually transform the data points to a new, high dimensional feature space. Kernelized SVM compute decision boundaries in terms of similarity measures in a high-dimensional feature space without actually doing a transformation. I think this is why it is also called **kernel trick**.

One of the commonly used kernel functions is radial basis function (RBF). Gamma parameter of RBF controls the distance of influence of a single training point. Low values of gamma indicates a large similarity radius which results in more points being grouped together. For high values of gamma, the points need to be very close to each other in order to be considered in the same group (or class). Therefore, models with very large gamma values tend to overfit. Following visualizations explain the concept better:



Large gamma



instance, if we have a point the right bottom corner, it is classified as “green” class. On the other hand, the second image is the case with large gamma. For data points to be grouped in the same class, they must fall in the tight bounded area. Thus, a small noise may cause a data point to fall out of a class. Large gamma values are likely to end up in overfitting.

As the gamma decreases, the regions separating different classes get more generalized. Very large gamma values result in too specific class regions (overfitting).

Gamma vs C parameter

For a linear kernel, we just need to optimize the c parameter. However, if we want to use an RBF kernel, both c and gamma parameter need to optimized simultaneously. If gamma is large, the effect of c becomes negligible. If gamma is small, c affects the model just like how it affects a linear model. Typical values for c and gamma are as follows. However, specific optimal values may exist depending on the application:

$0.0001 < \text{gamma} < 10$

$0.1 < c < 100$

> *It is very significant to remember for SVM that the input data need to be normalized so that features are on the same scale and compatible.*

Thank you for reading. Please let me know if you have any feedback.