

Figure 5-8. Similarity features using the Gaussian RBF

You may wonder how to select the landmarks. The simplest approach is to create a landmark at the location of each and every instance in the dataset. This creates many dimensions and thus increases the chances that the transformed training set will be linearly separable. The downside is that a training set with  $m$  instances and  $n$  features gets transformed into a training set with  $m$  instances and  $m$  features (assuming you drop the original features). If your training set is very large, you end up with an equally large number of features.

## Gaussian RBF Kernel

Just like the polynomial features method, the similarity features method can be useful with any Machine Learning algorithm, but it may be computationally expensive to compute all the additional features, especially on large training sets. However, once again the kernel trick does its SVM magic: it makes it possible to obtain a similar result as if you had added many similarity features, without actually having to add them. Let's try the Gaussian RBF kernel using the SVC class:

```
rbf_kernel_svm_clf = Pipeline((
    ("scaler", StandardScaler()),
    ("svm_clf", SVC(kernel="rbf", gamma=5, C=0.001))
))
rbf_kernel_svm_clf.fit(X, y)
```

This model is represented on the bottom left of [Figure 5-9](#). The other plots show models trained with different values of hyperparameters  $\gamma$  and  $C$ . Increasing  $\gamma$  makes the bell-shape curve narrower (see the left plot of [Figure 5-8](#)), and as a result each instance's range of influence is smaller: the decision boundary ends up being more irregular, wiggling around individual instances. Conversely, a small  $\gamma$  value makes the bell-shaped curve wider, so instances have a larger range of influence, and the decision boundary ends up smoother. So  $\gamma$  acts like a regularization hyperparameter: if your model is overfitting, you should reduce it, and if it is underfitting, you should increase it (similar to the  $C$  hyperparameter).

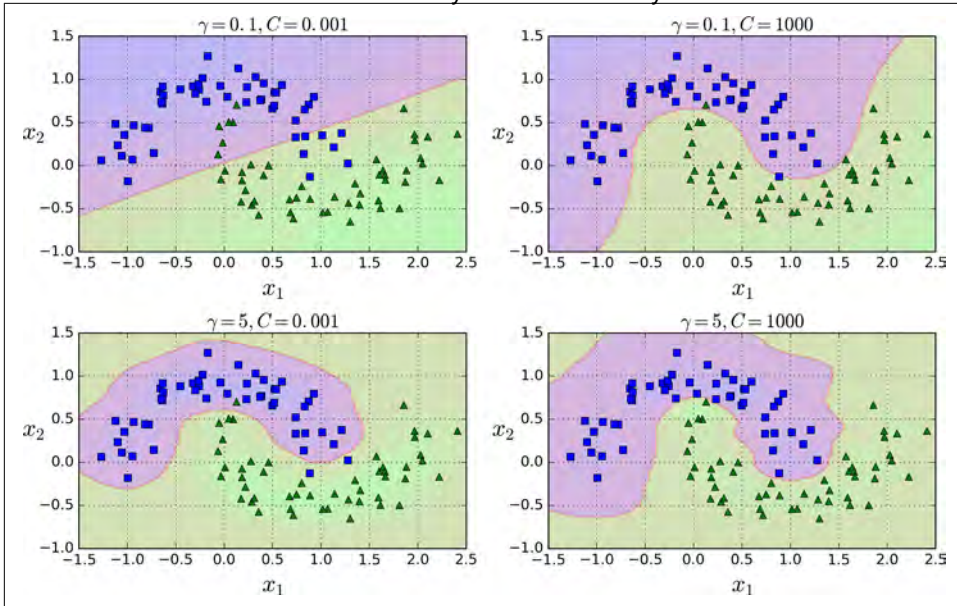


Figure 5-9. SVM classifiers using an RBF kernel

Other kernels exist but are used much more rarely. For example, some kernels are specialized for specific data structures. *String kernels* are sometimes used when classifying text documents or DNA sequences (e.g., using the *string subsequence kernel* or kernels based on the *Levenshtein distance*).



With so many kernels to choose from, how can you decide which one to use? As a rule of thumb, you should always try the linear kernel first (remember that `LinearSVC` is much faster than `SVC(kernel="linear")`), especially if the training set is very large or if it has plenty of features. If the training set is not too large, you should try the Gaussian RBF kernel as well; it works well in most cases. Then if you have spare time and computing power, you can also experiment with a few other kernels using cross-validation and grid search, especially if there are kernels specialized for your training set's data structure.

## Computational Complexity

The `LinearSVC` class is based on the *liblinear* library, which implements an **optimized algorithm** for linear SVMs.<sup>1</sup> It does not support the kernel trick, but it scales almost

<sup>1</sup> "A Dual Coordinate Descent Method for Large-scale Linear SVM," Lin et al. (2008).