Classifying data with SVMs

- **Support vector machines** (SVM) is one of the techniques we will use that doesn't have an easy probabilistic interpretation.
- The idea behind SVMs is that we find the plane that separates the group of the dataset the "best".
- Here, separation means that the choice of the plane maximizes the margin between the closest points on the plane.
- These points are called ***support vectors***.

Getting Ready

Let's get some data and get started:

```
>>> from sklearn import datasets
>>> X, y = datasets.make_classification()
```

Workflow

The mechanics of creating a support vector classifier is very simple; there are a few options available. Therefore, we'll do the following:

1. Create an SVC object and fit it to some fake data.
2. Fit the SVC object to some example data.
3. Talk a little about the SVC options.

Import support vector classifier (SVC) from the support vector machine module:

```
>>> from sklearn.svm import SVC
>>> base_svm = SVC()
>>> base_svm.fit(X, y)
```

Let's look at some of the attributes:

- `C`: In cases where we don't have a well-separated set, C will scale the error on the margin. As C gets higher, the penalization for the error becomes larger and the SVM will try to find a narrow margin even if it misclassifies more points.
- `class_weight`: This denotes how much weight to give to each class in the problem. This is given as a dictionary where classes are the keys and values are the weights associated with these classes.
- `gamma`: This is the gamma parameter for kernels and is supported by rgb, sigmoid, and ploy.
- `kernel`: This is the kernel to use; we'll use linear in the following section, but `rgb` is the popular and default choice.

Implementation

- Support Vector Machines will try to find the plane that best bifurcates the two classes.
- Let's look at a simple example with two features and a wellseparated outcome.

First, let's fit the dataset, and then we'll plot what's going on:

```
>>> X, y = datasets.make_blobs(n_features=2, centers=2)
>>> from sklearn.svm import LinearSVC
```

```
>>> svm = LinearSVC()
>>> svm.fit(X, y)
```

Now that we've fit the support vector machine, we'll plot its outcome at each point in the graph. This will show us the approximate decision boundary:

```
>>> from itertools import product
>>> from collections import namedtuple
>>> Point = namedtuple('Point', ['x', 'y', 'outcome'])
>>> decision_boundary = []
>>> xmin, xmax = np.percentile(X[:, 0], [0, 100])
>>> ymin, ymax = np.percentile(X[:, 1], [0, 100])
>>> for xpt, ypt in product(np.linspace(xmin-2.5, xmax+2.5, 20),
    np.linspace(ymin-2.5, ymax+2.5, 20)):
    p = Point(xpt, ypt, svm.predict([xpt, ypt]))
    decision_boundary.append(p)
```

```
>>> import matplotlib.pyplot as plt
>>> f, ax = plt.subplots(figsize=(7, 5))
>>> import numpy as np
>>> colors = np.array(['r', 'b'])
>>> for xpt, ypt, pt in decision_boundary:
    ax.scatter(xpt, ypt, color=colors[pt[0]], alpha=.15)
    ax.scatter(X[:, 0], X[:, 1], color=colors[y], s=30)
    ax.set_ylim(ymin, ymax)
    ax.set_xlim(xmin, xmax)
    ax.set_title("A well separated dataset")
```

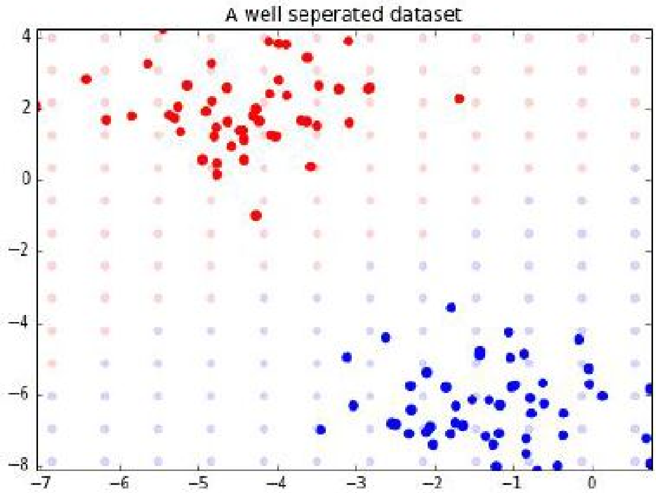


Figure 1

The following is the output:

Let's look at another example, but this time the decision boundary will not be so clear:

```
>>> X, y = datasets.make_classification(n_features=2,
n_classes=2,
n_informative=2,
n_redundant=0)
```

As we can see, this is not a problem that will easily be solved by a linear classification rule. While we will not use this in practice, let's have a look at the decision boundary. First, let's retrain the classifier with the new datapoints:

```
>>> svm.fit(X, y)
>>> xmin, xmax = np.percentile(X[:, 0], [0, 100])
```

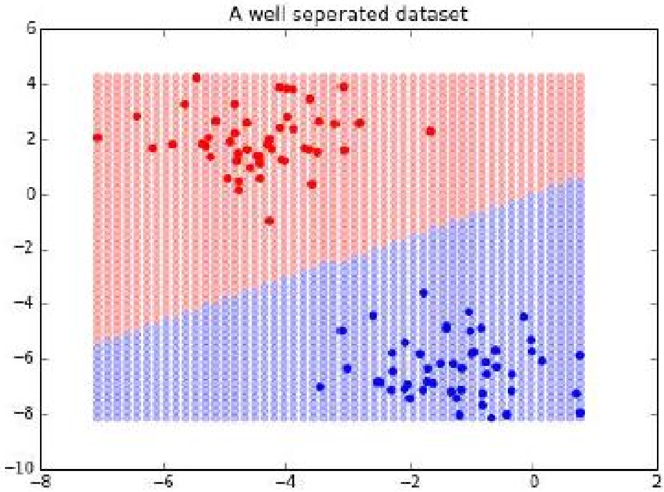


Figure 2

```
>>> ymin, ymax = np.percentile(X[:, 1], [0, 100])
>>> test_points = np.array([[xx, yy] for xx, yy in
     product(np.linspace(xmin, xmax),
     np.linspace(ymin, ymax))])
>>> test_preds = svm.predict(test_points)
```

```
>>> import matplotlib.pyplot as plt
>>> f, ax = plt.subplots(figsize=(7, 5))
>>> import numpy as np
>>> colors = np.array(['r', 'b'])
>>> ax.scatter(test_points[:, 0], test_points[:, 1],
    color=colors[test_preds], alpha=.25)
>>> ax.scatter(X[:, 0], X[:, 1], color=colors[y])
>>> ax.set_title("A well separated dataset")
```

The following is the output:

As we saw, the decision line isn't perfect, but at the end of the day, this is the best Linear SVM we will get.

Other Remarks

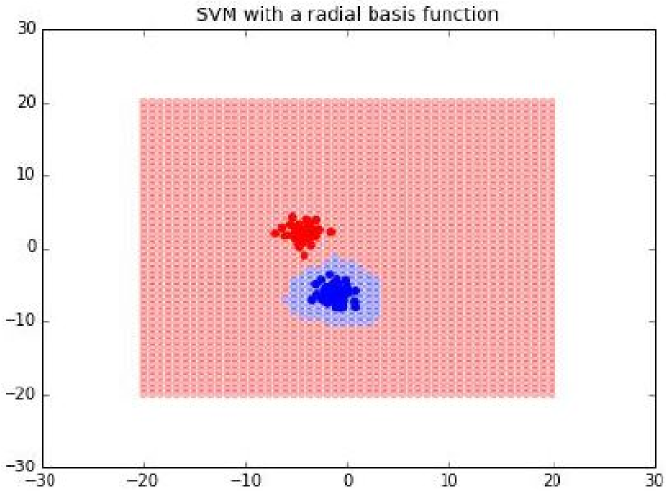
While we might not be able to get a better Linear SVM

- by default, the SVC classifier in scikitlearn will use the radial basis function.
- We've seen this function before, but let's take a look and see what it does to the decision boundaries of the dataset we just fit:

```
>>> radial_svm = SVC(kernel='rbf')
>>> radial_svm.fit(X, y)
>>> xmin, xmax = np.percentile(X[:, 0], [0, 100])
>>> ymin, ymax = np.percentile(X[:, 1], [0, 100])

>>> test_points = np.array([[xx, yy] for xx, yy in
    product(np.linspace(xmin, xmax),
    np.linspace(ymin, ymax))])
>>> test_preds = radial_svm.predict(test_points)
```

```
>>> import matplotlib.pyplot as plt
>>> f, ax = plt.subplots(figsize=(7, 5))
>>> import numpy as np
>>> colors = np.array(['r', 'b'])
>>> ax.scatter(test_points[:, 0], test_points[:, 1],
    color=colors[test_preds], alpha=.25)
```




```
>>> ax.scatter(X[:, 0], X[:, 1], color=colors[y])
>>> ax.set_title("SVM with a radial basis function")
```

The following is the output:

As we can see, the decision boundary has been altered. We can even pass in our own radial basis function, if needed: (returns the exponentiation of the dot of the X and y matrices.)

```
>>> def test_kernel(X, y):
    return np.exp(np.dot(X, y.T))
```

```
>>> test_svc = SVC(kernel=test_kernel)
>>> test_svc.fit(X, y)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
degree=3,
```

```
gamma=0.0, kernel=<function test_kernel at 0x121fdfb90>,
max_iter=-1, probability=False, random_state=None,
shrinking=True, tol=0.001, verbose=False)
```