# Support Vector Machine*

February 20, 2023

# 1 Standard SVM

## 1.1 Separable Classes

In the previous lectures, we explored a class of linear classification models. In this lecture, an alternative rationale for designing linear classifiers will be adopted. Similarly, We will only discuss the two-class linearly separable task.
Let $\mathbf{x}_i (i \in \{1, 2, 3, ..., n\})$ be the feature vectors of the training set $X$. These belong to either of two classes, $y_1$, $y_2$, which are assumed to be linearly separable. The goal, once more, is to design a decision boundary

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0$$

that classifies correctly all the training vectors. Such a decision boundary is not unique (Figure 1).

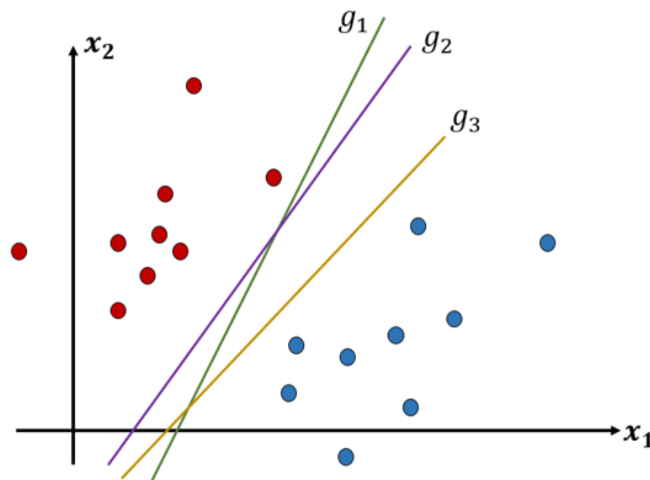



Figure 1: Multiple decision boundaries can correctly classify the training samples in a 2D feature space.

*References
- Christopher Bishop. Pattern Recognition and Machine Learning. 2006
- Sergios Theodoridis, Konstantinos Koutroumbas. Pattern Recognition. 2009
- Multiclass Classification Using Support Vector Machines. August 25, 2021

However, which one would any sensible engineer choose as the classifier for operation in practice, where data outside the training set will be fed to it? No doubt the answer is the purple one. The reason is that this boundary leaves more "room "on either side so that data in both classes can move a bit more freely, with less risk of causing an error. Thus such a boundary can be trusted more when it is faced with the challenge of operating with unknown data (Figure 2).
Here we have touched on a very important issue in the classifier design stage. It is known as the **generalization performance** of the classifier. This refers to the capability of the classifier, designed using the training data set, to operate satisfactorily with data outside this set.

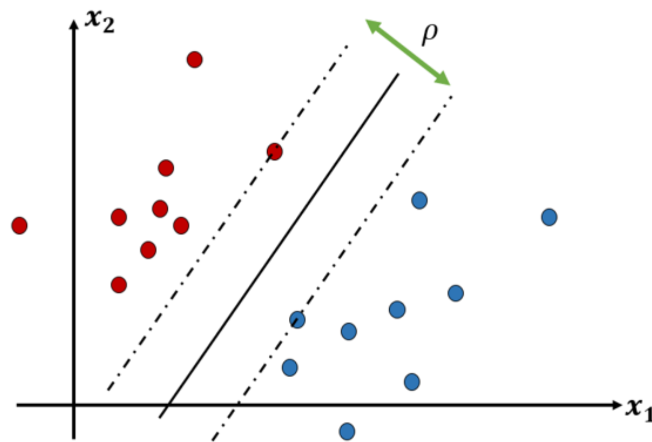



Figure 2: A natural choice of decision boundary would be the one that generates the maximum margin of both classes.

After the above brief discussion, we are ready to accept that a very sensible choice for the boundary would be the one that leaves the maximum margin from both classes. Let's now quantify the margin that a decision boundary leaves from both classes.
We have two classes, one positive class $y_1$ with the label +1, one negative class $y_2$ with the label -1. We start from the assumption that the weight vector $\mathbf{w}$ and the bias $b$ are constrained so that the output of the linear model is always larger than 1 or smaller than -1.

$$\begin{cases} \mathbf{w}^T\mathbf{x} + b \geq +1 & \text{if } y_i = +1 \\ \mathbf{w}^T\mathbf{x} + b \leq -1 & \text{if } y_i = -1 \end{cases}$$

First, we consider the direction of $\mathbf{w}$. We have two points $\mathbf{x}_1$ and $\mathbf{x}_2$ both of which lie on the boundary. Because $g(\mathbf{x}_1) = g(\mathbf{x}_2) = 0$, we have:

$$\mathbf{w}^T(\mathbf{x}_1 - \mathbf{x}_2) = 0$$

and hence the $\mathbf{w}$ is orthogonal to every vector lying within the boundary, and so $\mathbf{w}$ determines the orientation of the decision boundary (Figure 3).

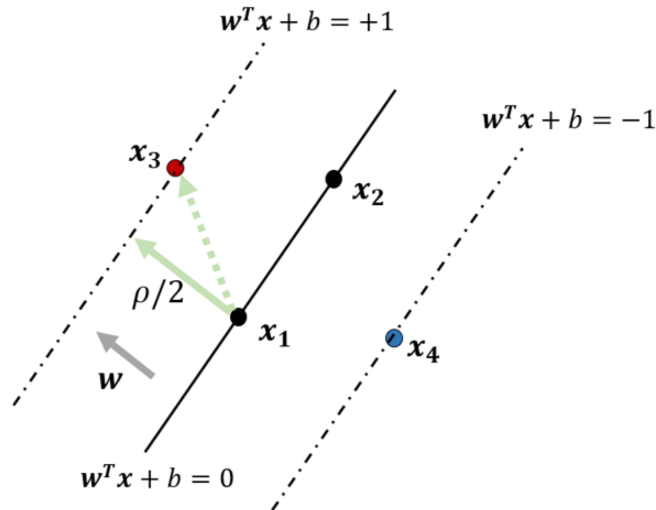



Figure 3: The margin from the closest points of both classes to the decision boundary.

For computing the margin $\rho$, we start by looking at the closest distance from point $\mathbf{x}_3$ of the positive class to the boundary. This can be computed by projecting the line segment vector $\mathbf{x}_3 - \mathbf{x}_1$ over the orientation vector $\mathbf{w}$:

$$\frac{1}{2}\rho = \frac{\mathbf{w}^T(\mathbf{x}_3 - \mathbf{x}_1)}{\|\mathbf{w}\|} = \frac{(\mathbf{w}^T\mathbf{x}_3 + b) - (\mathbf{w}^T\mathbf{x}_1 + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

where $\|\cdot\|$ means the norm (i.e., magnitude) of a vector. For the closest point $\mathbf{x}_4$ of the negative class, we derive the distance in the same way. Therefore, we obtain the margin:

$$\rho = \frac{2}{\|\mathbf{w}\|}$$

which means that to maximize the margin $\rho$ we need to minimize the norm of the weight vector $\mathbf{w}$.
Based on the analysis above, our task now becomes: find a decision boundary with the parameters $\mathbf{w}$ and $b$ so as to:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \geq 0 \quad \forall i = 1, 2, .., n \end{aligned}$$

## 1.2 SVM Model Optimization (Optional)

Obviously, minimizing the norm $\|\mathbf{w}\|$ makes the margin maximum $\rho$. This is a nonlinear (quadratic) optimization task subject to a set of linear inequality constraints. A common way to tackle such a problem is the Lagrangian method. In this section, we derive the steps for optimizing a standard SVM model.
**Note:** This section is discussed as we would like to present what data points are "support vectors" and why they are named in this way. We don't require you to command the content of this section, thus we will NOT ask any related questions in the final exam.
The SVM optimization problem has 1 objective and $n$ corresponding constraints ($n$ is the number of input sample vectors). For the $i_{th}$ constraint, we apply a non-negative

Lagrangian multiplier $\lambda_i$. The overall Lagrangian function of the original problem is given by:

$$L(\mathbf{w}, b, \lambda) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \lambda_i (y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1)$$

where $\lambda$ is the vector that contains all the Lagrangian multipliers $(\lambda_1, \lambda_2, ..., \lambda_n)$. Applying KKT [1] conditions, we know that the minimizer of the Lagrangian function must satisfy:

$$\begin{aligned} y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 &\geq 0 \\ \lambda_i &\geq 0 \\ \lambda_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1) &= 0 \\ \frac{\partial L(\mathbf{w}, b, \lambda)}{\partial \mathbf{w}} &= 0 \\ \frac{\partial L(\mathbf{w}, b, \lambda)}{\partial b} &= 0 \end{aligned}$$

From the KKT conditions above we obtain:

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0$$

After computation a lot of $\lambda_i$ will become 0. Only those vectors lying on the margin hyperplanes $\mathbf{w}^T\mathbf{x}_i + b = \pm 1$ will have positive $\lambda_i$. This means the Lagrangian multiplier vector $\lambda$ is a sparse vector. Thus, the vector parameter $\mathbf{w}$ of the optimal solution is a linear combination of $N_s \leq N$ feature vectors that are associated with $\lambda_i \geq 0$. That is,

$$\mathbf{w} = \sum_{i=1}^{N_s} \lambda_i y_i \mathbf{x}_i.$$

These input vectors which contribute to $\mathbf{w}$ are known as **support vectors** and the optimum decision boundary derived is known as a **Support Vector Machine (SVM)**.

## 1.3 Geometrical Interpretation

Figure 4 gives an illustration of the support vectors in an SVM model. Intuitively, we can see that only samples located on the margin of each class will determine the decision boundary. Samples farther away from the margin have little influence on the boundary. This, again, verifies the conclusions from Section 1.2, that only those vectors lying on the margin hyperplanes $\mathbf{w}^T\mathbf{x}_i + b = \pm 1$ will contribute to $\mathbf{w}$.

[1] Reference
- Stephen Boyd. Convex Optimization. Chapter 5. 2004.

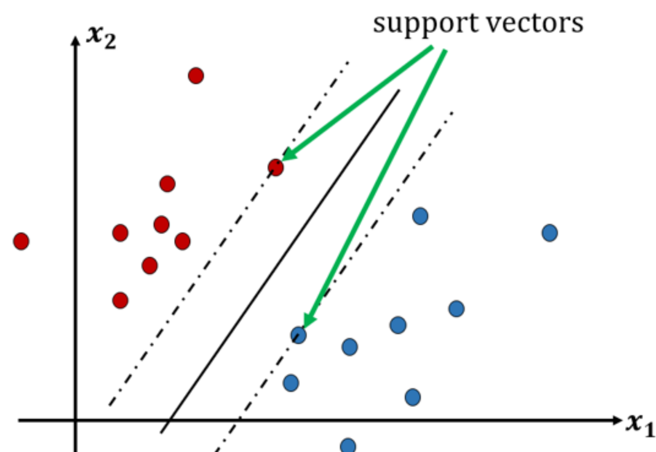



Figure 4: Support vectors on the class margin.

SVM has several interesting properties:

- The optimal decision boundary of an SVM is unique. Intuitively, we can see that there cannot exist multiple decision boundaries achieving the same maximum margin. From a mathematical point of view, the loss function in Section 1.2 is a strict convex one. This guarantees that any local minimum is also global and unique.
- SVM is robust to data outliers, as vectors farther away from the margin of the classes do not influence the model.
- SVM is also little affected by data distribution and density.
- SVM appears to work well in high dimensional feature spaces. One possible explanation is that SVM is determined by the support data vectors and not directly by the features.

Nevertheless, SVM has its limitations. It is usually computational expensive due to the optimization technique it adopts. Moreover, it performs badly when classes are highly overlapped. If classes are slightly overlapped, the soft-margin SVM can be adopted. We will introduce this in Section 2.

# 2 Soft-Margin SVM

When the classes are not separable, the above setup is no longer valid. Figure 5 illustrates the case in which the two classes are not separable. Any attempt to draw a decision boundary will never end up with a class separation band with no data points inside it, as was the case in the linearly separable task. Recall that the margin is defined as the distance between the pair of parallel hyperplanes described by:

$$\mathbf{w}^T\mathbf{x} + b = \pm 1$$

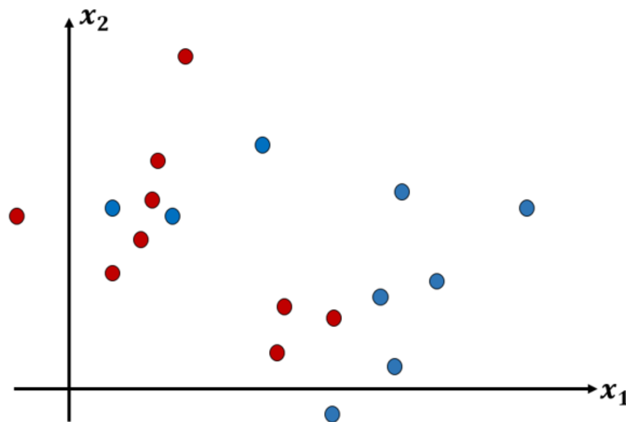



Figure 5: Non-separable classes.

The feature vectors now belong to one of the following three categories (Figure 6):

- Vectors that fall outside the band and are correctly classified. These vectors comply with the constraints:
$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$
- Vectors falling inside the band and are correctly classified. They satisfy:
$$0 \leq y_i(\mathbf{w}^T\mathbf{x}_i + b) < 1$$
- Vectors that are misclassified. They are enclosed by circles and obey the inequality:
$$y_i(\mathbf{w}^T\mathbf{x}_i + b) < 0$$

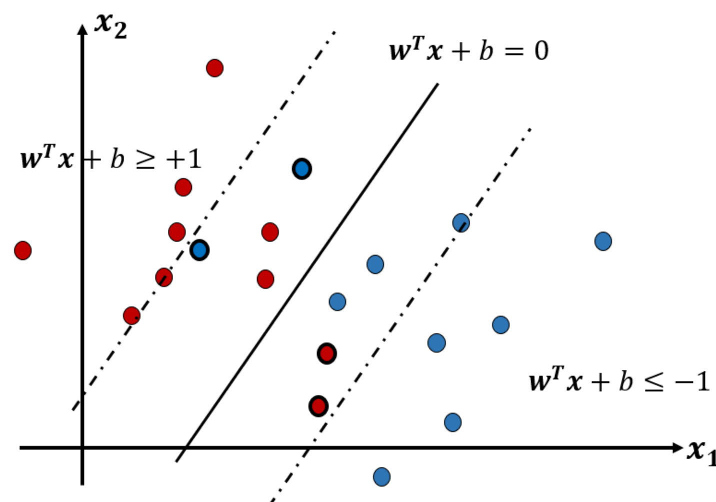



Figure 6: Misclassified data samples.

All three cases can be treated under a single type of constraints by introducing a new set of variables, namely:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$

The first category of data corresponds to $\xi_i = 0$, the second to $0 < \xi_i \leq 1$, and the third to $\xi_i > 1$. The variables $\xi_i$ are known as slack variables.
The goal now is to make the margin as large as possible but at the same time to keep the number of vectors with $\xi_i$ as small as possible. In mathematical terms, this is equivalent to adopting to minimize the cost function:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad \forall i = 1, 2, .., n \\ & \xi_i \geq 0 \quad \forall i = 1, 2, ..., n \end{aligned}$$

where C is a constant term and can be tuned by users. The larger the $C$, the smaller $\xi$ we allow that the input vectors deviate from the decision boundary. Otherwise, we give more tolerance to the vectors that could be wrongly classified by the decision boundary. The soft-margin SVM can be solved using the similar optimization technique described in Section 1.2.

# 3 Multi-class Classification using SVM (Optional)

In this section, we'll introduce the multi-class classification using Support Vector Machines (SVM). We'll also look at Python code for multi-class classification using Scikitlean SVM.

## 3.1 Binary classification vs multi-class classification

**Binary classification**. In this type, the machine should classify an instance as only one of two classes; yes/no, 1/0, or true/false. The classification question in this type is always in the form of yes/no. For example, does this image contain a human? Does this text have a positive sentiment? Will the price of a particular stock increase in the next month?
**Multi-class classification**. In this type, the machine should classify an instance as only one of three classes or more. The following are examples of multi-class classification: (1) classifying a text as positive, negative, or neutral; (2) determining the dog breed in an image; (3) categorizing a news article to sports, politics, economics, or social.

## 3.2 Generalization to multi-class classification

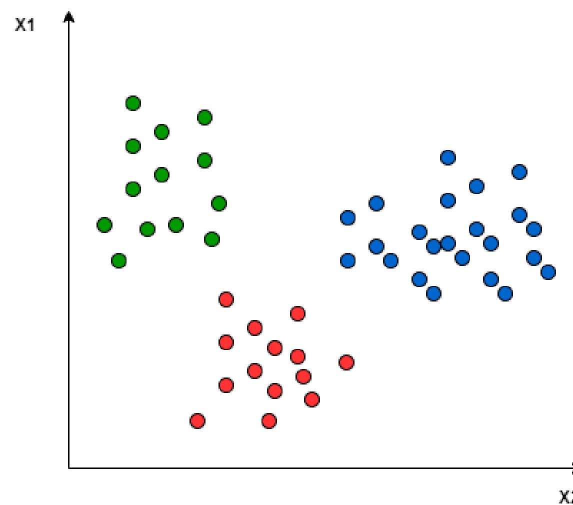
In its most simple type, SVM doesn't support multi-class classification natively. It supports binary classification and separating data points into two classes. For multi-class classification, the same principle is utilized after breaking down the multi-classification problem into multiple binary classification problems.

The idea is to map data points to high dimensional space to gain mutual linear separation between every two classes. This is called a **One-to-One approach**, which breaks down the multi-class problem into multiple binary classification problems. A binary classifier per each pair of classes. Another approach one can use is **One-to-Rest**, where the breakdown is set to a binary classifier per each class.

A single SVM does binary classification and can differentiate between two classes. So that, according to the two breakdown approaches, to classify data points from $m$ classes data set:

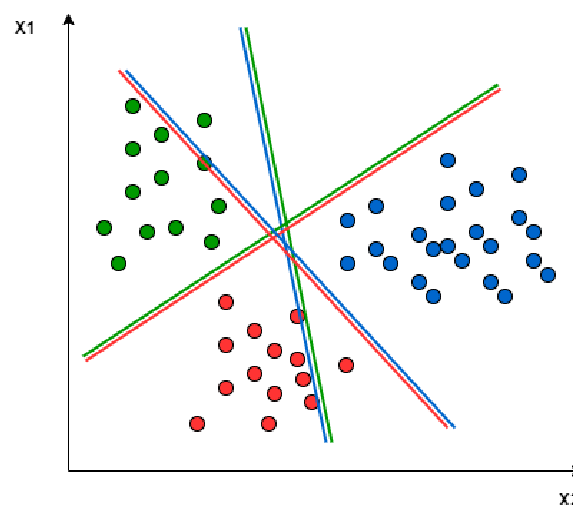
- In the One-to-Rest approach, the classifier can use $m$ SVMs. Each SVM would predict membership in one of the $m$ classes.
- In the One-to-One approach, the classifier can use $\frac{m(m-1)}{2}$ SVMs.

Let's take an example of 3 classes classification problem; green, red, and blue, as the following image:

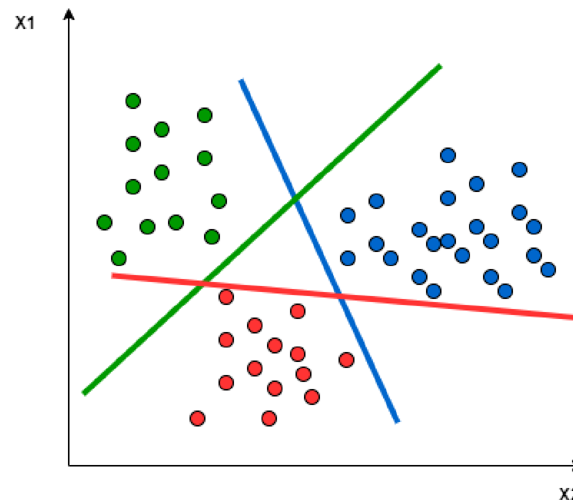



Applying the two approaches to this data set results in the followings:

- In the One-to-One approach, we need a hyperplane to separate between every two classes, neglecting the points of the third class. This means the separation takes into account only the points of the two classes in the current split. For example, the red-blue line tries to maximize the separation only between blue and red points. It has nothing to do with green points:



- In the One-to-Rest approach, we need a hyperplane to separate between a class and all others at once. This means the separation takes all points into account, dividing them into two groups; a group for the class points and a group for all other points. For example, the green line tries to maximize the separation between green points and all other points at once:




## 3.3 SVM Multi-class Classification in Python

In this subsection, the Python code shows an implementation for building (training and testing) a multiclass classifier (3 classes), using Python 3.7 and Scikitlean library. We developed two different classifiers to show the usage of two different kernel functions: polynomial and RBF. The code also calculates the accuracy and F1 scores to show the performance difference between the two selected kernel functions on the same data set.

In this code, we use the Iris flower data set. That data set contains three classes of 50 instances each, where each class refers to a type of Iris plant.

We'll start our script by importing the needed classes:

```
from sklearn import svm, datasets
import sklearn.model_selection as model_selection
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
```

Load Iris data set from Scikitlearn, no need to download it separately:

```
iris = datasets.load_iris()
```

Now we need to separate features set $X$ from the target column (class label) $y$, and divide the data set to 60% for training, and 40% for testing:

```
X = iris.data[:, :2]
y = iris.target
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
    train_size=0.60, test_size=0.40, random_state=101)
```

We'll create two objects from SVM, to create two different classifiers; one with a polynomial kernel, and another one with an RBF kernel:

```
rbf = svm.SVC(kernel='rbf', gamma=0.5, C=0.1).fit(X_train, y_train)
poly = svm.SVC(kernel='poly', degree=3, C=1).fit(X_train, y_train)
```

To calculate the efficiency of the two models, we'll test the two classifiers using the test dataset:

```
poly_pred = poly.predict(X_test)
rbf_pred = rbf.predict(X_test)
```

Finally, we'll calculate the accuracy and F1 scores for SVM with the polynomial kernel:

```
poly_accuracy = accuracy_score(y_test, poly_pred)
poly_f1 = f1_score(y_test, poly_pred, average='weighted')
print('Accuracy (Polynomial Kernel): ', "%.2f" % (poly_accuracy*100))
print('F1 (Polynomial Kernel): ', "%.2f" % (poly_f1*100))
```

In the same way, the accuracy and F1 scores for SVM with the RBF kernel:

```
rbf_accuracy = accuracy_score(y_test, rbf_pred)
rbf_f1 = f1_score(y_test, rbf_pred, average='weighted')
print('Accuracy (RBF Kernel): ', "%.2f" % (rbf_accuracy*100))
print('F1 (RBF Kernel): ', "%.2f" % (rbf_f1*100))
```

That code will print the following results:

```
Accuracy (polynomial kernel): 70.00
F1 (polynomial kernel): 69.67
Accuracy (RBF Kernel): 76.67
F1 (RBF Kernel): 76.36
```

Out of the known metrics for validating machine learning models, we choose Accuracy and F1 as they are the most used in supervised machine learning. For the accuracy score, it shows the percentage of the true positive and true negative to all data points. So, it's useful when the data set is balanced. For the F1 score, it calculates the harmonic mean between precision and recall, and both depend on the false positive and false negative. So, it's useful to calculate the F1 score when the data set isn't balanced.

Playing around with SVM hyperparameters, like C, gamma, and degree in the previous code snippet will produce different results. As we can see, in this problem, SVM with RBF kernel function outperforms SVM with the polynomial kernel function.