# chapter-4-4

January 23, 2022

Run in Google Colab

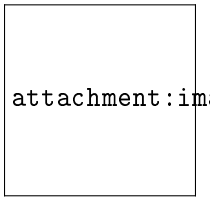
## 1 Supervised Models: Support Vector Machines (SVM)

### 1.1 Linear Classification

Machine learning involves predicting and classifying data and to do so we employ various machine learning algorithms according to the dataset. SVM or Support Vector Machine is a linear model for classification and regression problems. It can solve linear and non-linear problems and work well for many practical problems. This chapter will primarily focus on how SVM works as a classifier. It classifies data points into two classes at a time(this does not mean it is only a binary classifier, it only separates data points into two classes at a time), using a decision boundary (a hyperplane in this case). The primary objective of the Support vector Classifier is finding the 'Optimal Separating Hyperplane(Decision Boundary)'.

At first approximation what SVMs do is to find a separating line(or hyperplane) between data of two classes. SVM is an algorithm that takes the data as an input and outputs a line that separates those classes if possible but in this case our optimization objective is to maximize the margin.

The margin is defined as the distance between the separating hyperplane (decision boundary) and the training examples that are closest to this hyperplane, which are the so-called **support vectors**. This is illustrated in the following figure:




Thus SVM tries to make a decision boundary in such a way that the separation between the two classes (the margin) is as wide as possible.

#### 1.1.1 What is a Hyperplane?

In geometry, it is an n-dimensional generalization of a plane, a subspace with one less dimension $(n-1)$ than its origin space. In one-dimensional space, it is a point, In two-dimensional space it is a line, In three-dimensional space, it is an ordinary plane, in four or more dimensional spaces, it is then called a 'Hyperplane'. Take note of this, it is really how the Support Vector Machine works

behind the scenes, the dimensions are the features represented in the data. For example, say we want to carry out a classification problem and we want to be able to tell if a product gets purchased or not(a binary classification), if there is just one feature (say Gender) available as a feature in the dataset, then it is in one-dimensional space and the subspace (the separating/decision boundary) representation will be $(n-1=0)$ a 0-dimensional space, represented with just a point showing the separation of classes (Purchased or not). If there are two features(Age and Gender), it is a two-dimensional space (2D), with either of Age and Gender on the $X$ and $Y$-axis, the decision boundary will be represented as a simple line. Similarly, if the features are three(Age, Gender, Income), the decision boundary will be a two-dimensional plane in a three-dimensional space. Furthermore, if we have a four or more dimensional space data points, then it is called a 'Hyperplane' with $n-1$ dimension.

The Hyperplane is simply a concept that separates an n-dimensional space into two groups/halves. In machine learning terms, it is a form of a decision boundary that algorithms like the Support Vector Machine uses to classify or separate data points. There are two parts to it, the negative side hyperplane and the positive part hyperplane, where data points/instances can lie on either part, signifying the group/class they belong to.

### 1.1.2 Hard Margins

The rationale behind having decision boundaries with large margins is that they tend to have a lower generalization error, whereas models with small margins are more prone to overfitting. To get an idea of the margin maximization, let's take a closer look at those positive and negative hyperplanes that are parallel to the decision boundary.

When the data is linearly separable, and we don't want to have any misclassifications, we use SVM with a **hard margin**. However, when a linear boundary is not feasible, or we want to allow some misclassifications in the hope of achieving better generality, we can opt for a **soft margin** for our classifier.

...

attachment:chapter-4-4_pic_10.png

We can scale $w_1$, $w_2$, $b_u$, and $b_d$ by the same constant without changing the model. We can therefore set $b_u = b + 1$ and $b_d = b - 1$ so that the width of the pathway is

$$P = \frac{2}{\sqrt{w_1^2 + w_2^2}} = \frac{2}{||\mathbf{w}||^2}$$

In the **hard margin** case the algorithm minimizes $w_1^2 + w_2^2$ subject to **perfect separation** being achieved

Now, the objective function of the SVM becomes the maximization of this margin by maximizing $P$ under the constraint that the examples are classified correctly, which can be written as:

$$b + \mathbf{w}\cdot\mathbf{x}^{()} \geq 1 \quad \text{if } ^{()}=1$$

$$b + \mathbf{w}\cdot\mathbf{x}^{()} \leq 1 \quad \text{if } ^{()}=-1$$

for $= 1, \ldots,$. Here, $N$ is the number of examples in our dataset.

These two equations basically say that all negative-class examples should fall on one side of the negative hyperplane, whereas all the positive-class examples should fall behind the positive hyperplane, which can also be written more compactly as follows:

$$^{()}\left(b + \mathbf{w}\cdot\mathbf{x}^{()}\right) \geq 1 \quad \forall i \tag{1}$$

In practice, though, it is easier to minimize the reciprocal term, $\frac{1}{2}||\mathbf{w}||^2$ , which can be solved by quadratic programming.

### 1.1.3 Lagrange Multiplier

Because we have a contrained optimization problem (with inequality contraints) we can recast the problem using Lagrange multipliers, $\alpha \geq 0$ and search for the critical points of

$$L = \frac{1}{2}|\theta|^2 - \sum_{n=1}^{N} \alpha_n \left(y^{(n)}\left(\grave{}^T\mathbf{x}^{(n)} + \theta_0\right) - 1\right) \tag{2}$$

To find the critical points we differentiate with respect to the scalar $\theta_0$ and with respect to the vector $\grave{}$. Remember that differentiation with respect to a vector just means differentiating with respect to each of its entries. Setting these derivatives to zero you end up with:

$$\sum_{n=1}^{N} \alpha_n y^{(n)} = 0 \tag{3}$$

and

$$\grave{} = \sum_{n=1}^{N} \alpha_n y^{(n)} \mathbf{x}^{(n)} \tag{4}$$

which means that our vector orthogonal to the hyperplane is just a linear combination of sample vectors. Substituting (**??**) into (**??**) and using (**??**) results in

$$L = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} {\mathbf{x}^{(i)}}^T \mathbf{x}^{(j)} \tag{5}$$

We want to maximize $L$ over the $\alpha$s, all greater than or equal to zero, subject to (**??**). This is known as the dual problem.

Once we have found the $\alpha$s the dual version of the classifier for a new point $\mathbf{u}$ is the just

$$\sum_{n=1}^{N} \alpha_n y^{(n)} \mathbf{x}^{(n)^T} \mathbf{u} + \theta_0 \tag{6}$$

whether this is greater or less than zero.

**TODO Chiarire bene la seguente frase:** The maximization of $L$ whill return almost all $\alpha$s as zero. Those that aren't zero correspond to the support vectors, the cases that define the margins.

### 1.1.4 Soft Margins

Let's briefly mention the slack variable, , which was introduced by Vladimir Vapnik in 1995 and led to the socalled soft-margin classification. The motivation for introducing the slack variable, $\xi$ , was that the linear constraints need to be relaxed for nonlinearly separable data to allow the convergence of the optimization in the presence of misclassifications, under appropriate cost penalization.

The positive-valued slack variable is simply added to the linear constraints:

$$b + \mathbf{w} \cdot \mathbf{x}^{()} \geq 1 - \xi^{(i)} \quad \text{if } {}^{()} = 1$$

$$b + \mathbf{w} \cdot \mathbf{x}^{()} \leq -1 + \xi^{(i)} \quad \text{if } {}^{()} = -1$$

So, the new objective to be minimized (subject to the contraints) becomes

$$\frac{1}{2}||\mathbf{w}||^2 + C\left(\sum_i \xi^{(i)}\right) \tag{7}$$

Via the variable, C, we can then control the penalty for misclassification. Large values of C correspond to large error penalties, whereas we are less strict about misclassification errors if we choose smaller values for C. We can then use the C parameter to control the width of the margin and therefore tune the bias-variance tradeoff.

attachment:image.png

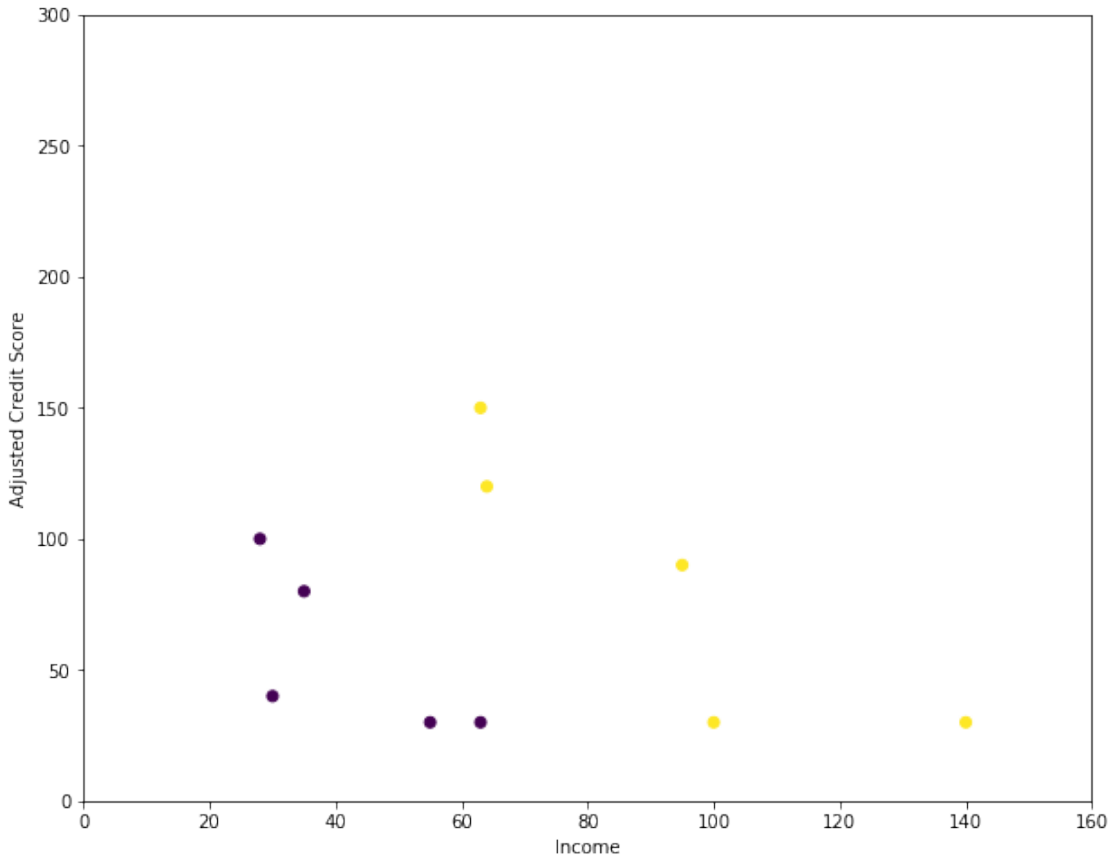
## 1.2 Example

[1]:
```
# This section carries out calculations for the example in Table 5.2 of Chapter␣
↪5 Hull J. C.
```

[2]:
```
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[3]:
```
Income = pd.DataFrame([30, 55, 63, 35, 28, 140, 100, 95, 64, 63])
Credit = pd.DataFrame([40, 30, 30, 80, 100, 30, 30, 90, 120, 150])
Loan = pd.DataFrame([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
```

[4]:
```
plt.figure(figsize=(10,8))
plt.scatter(Income, Credit, c=Loan)
plt.xlabel('Income')
plt.ylabel('Adjusted Credit Score')
axes = plt.gca()
axes.set_xlim([0,160])
axes.set_ylim([0,300])
plt.show()
```



### 1.2.1 Hard Margin Calculation

[5]:
```
X = np.asarray(pd.concat([Income, Credit],axis=1))
y = np.asarray(Loan).ravel()
clf = SVC(kernel='linear',tol=1e-5)
clf.fit(X,y)
```

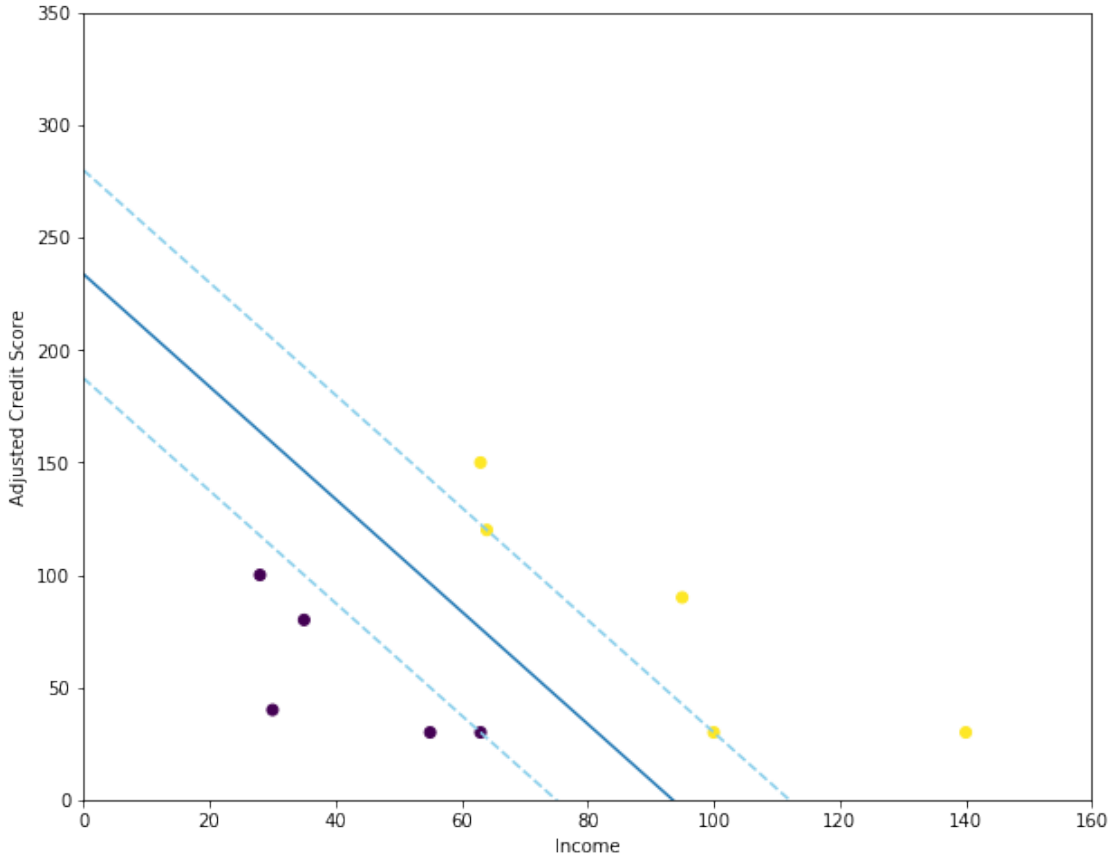
[5]:
```
SVC(kernel='linear', tol=1e-05)
```

[6]:
```
w = clf.coef_[0]
b = -clf.intercept_[0]
print(w)
print(b)
```

```
[0.05405405 0.02162159]
5.054052228526239
```

The outer lines are $w_1x_1 + w_2x_2 = b_u$ and $w_1x_1 + w_2x_2 = b_d$. The middle line is $w_1x_1 + w_2x_2 = b$, where $b_u = b + 1$ and $b_d = b - 1$. The width of the path is $\frac{2}{\sqrt{w_1^2+w_2^2}}$.

[7]:
```
x1 = np.linspace(0,160,300)
w1 = w[0]
w2 = w[1]
bu = b+1
bd = b-1
y1 = (bu-w1*x1)/w2
y2 = (bd-w1*x1)/w2
y0 = (b-w1*x1)/w2
plt.figure(figsize=(10,8))
plt.scatter(Income, Credit, c=Loan)
plt.plot(x1,y1,'--',color='skyblue')
plt.plot(x1,y2,'--',color='skyblue')
plt.plot(x1,y0,'-')
plt.xlabel('Income')
plt.ylabel('Adjusted Credit Score')
axes = plt.gca()
axes.set_xlim([0,160])
axes.set_ylim([0, 350])
plt.show()
```




### 1.2.2 Modification for Soft Margin

Note that the objective function of SVC is $C\sum_{j=1}^{n} z_j + \frac{1}{2}\sum_{j=1}^{n} w_j^2$. If our objective function is $C\sum_{j=1}^{n} z_j + \sum_{j=1}^{n} w_j^2$, then we to set $C = C/2$ in SVC.

[8]:
```
#We carry out calculations for C=0.0005 which corresponds to C=0.001 in Hull's␣
 ↪book.
X = np.asarray(pd.concat([Income, Credit],axis=1))
y = np.asarray(Loan).ravel()
clf = SVC(kernel='linear',C=0.0005,tol=1e-5)
clf.fit(X,y)
```

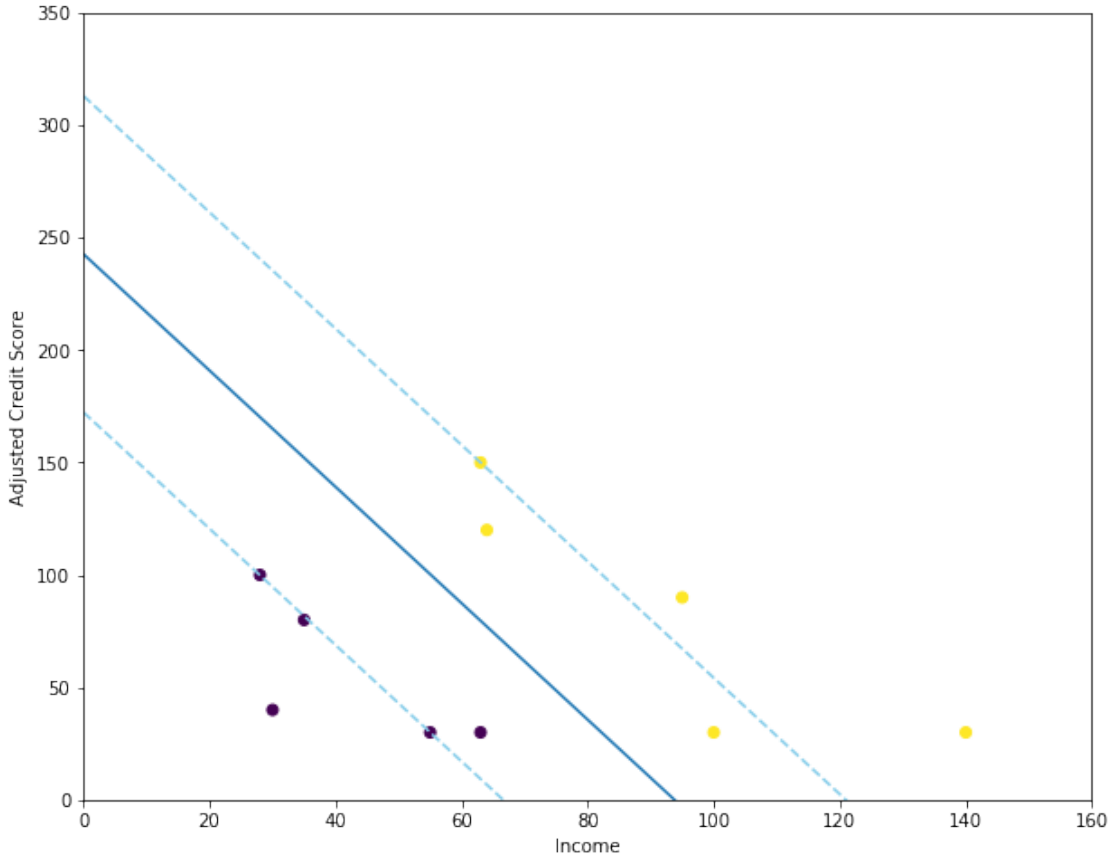
[8]:
```
SVC(C=0.0005, kernel='linear', tol=1e-05)
```

[9]:
```
w = clf.coef_[0]
b = -clf.intercept_[0]
print(w)
print(b)
```

```
[0.03678097 0.0142146 ]
3.449391592920354
```

The outer lines are $w_1x_1 + w_2x_2 = b_u$ and $w_1x_1 + w_2x_2 = b_d$. The middle line is $w_1x_1 + w_2x_2 = b$, where $b_u = b + 1$ and $b_d = b - 1$. The width of the path is $\frac{2}{\sqrt{w_1^2+w_2^2}}$.

[10]:
```
x1 = np.linspace(0,160,100)
w1 = w[0]
w2 = w[1]
bu = b+1
bd = b-1
y1 = (bu-w1*x1)/w2
y2 = (bd-w1*x1)/w2
y0 = (b-w1*x1)/w2
plt.figure(figsize=(10,8))
plt.scatter(Income, Credit, c=Loan)
plt.plot(x1,y1,'--',color='skyblue')
plt.plot(x1,y2,'--',color='skyblue')
plt.plot(x1,y0,'-')
plt.xlabel('Income')
plt.ylabel('Adjusted Credit Score')
axes = plt.gca()
axes.set_xlim([0,160])
axes.set_ylim([0, 350])
plt.show()
```




For different values of $C$ and find out the loans misclassified as well as width of pathway

[11]:
```
# The values of C are those in Hull's book. We divide them by 2 to get␣
 ↪corresponding values for Sklearn
for C in [0.01, 0.001, 0.0005, 0.0003, 0.0002]:
    clf = SVC(kernel='linear',C=C/2,tol=1e-6)
    clf.fit(X,y)
    S = clf.score(X,y)
    w = clf.coef_[0]
    b = -clf.intercept_[0]
    P = 2/np.sqrt(w[0]**2+w[1]**2)
    sum = 0
    for i in range(len(Income)):
        if Loan.iloc[i,0] == 0:
            sum += max(w[0] * Income.iloc[i,0] + w[1] * Credit.iloc[i,0] - b,0)
        else:
            sum += max(b- w[0] * Income.iloc[i,0] - w[1] * Credit.iloc[i,0],0)
    print("C = %6.4f, w1 = %6.4f, w2 = %6.4f, b = %5.2f, Loan Misclassified = %3.
 ↪0f%%, Width = %5.1f, Cent Error = %6.4f" %(C,w[0],w[1],b,100*(1-S),P,sum))
```

```
C = 0.0100, w1 = 0.0541, w2 = 0.0216, b =  5.05, Loan Misclassified =   0%,
```

```
Width =  34.4, Cent Error = 0.0000
C = 0.0010, w1 = 0.0368, w2 = 0.0142, b =  3.45, Loan Misclassified =   0%,
Width =  50.7, Cent Error = 0.0000
C = 0.0005, w1 = 0.0290, w2 = 0.0155, b =  3.15, Loan Misclassified =   0%,
Width =  60.9, Cent Error = 0.0000
C = 0.0003, w1 = 0.0235, w2 = 0.0151, b =  2.75, Loan Misclassified =   0%,
Width =  71.5, Cent Error = 0.0000
C = 0.0002, w1 = 0.0193, w2 = 0.0124, b =  2.07, Loan Misclassified =   0%,
Width =  87.2, Cent Error = 0.0000
```

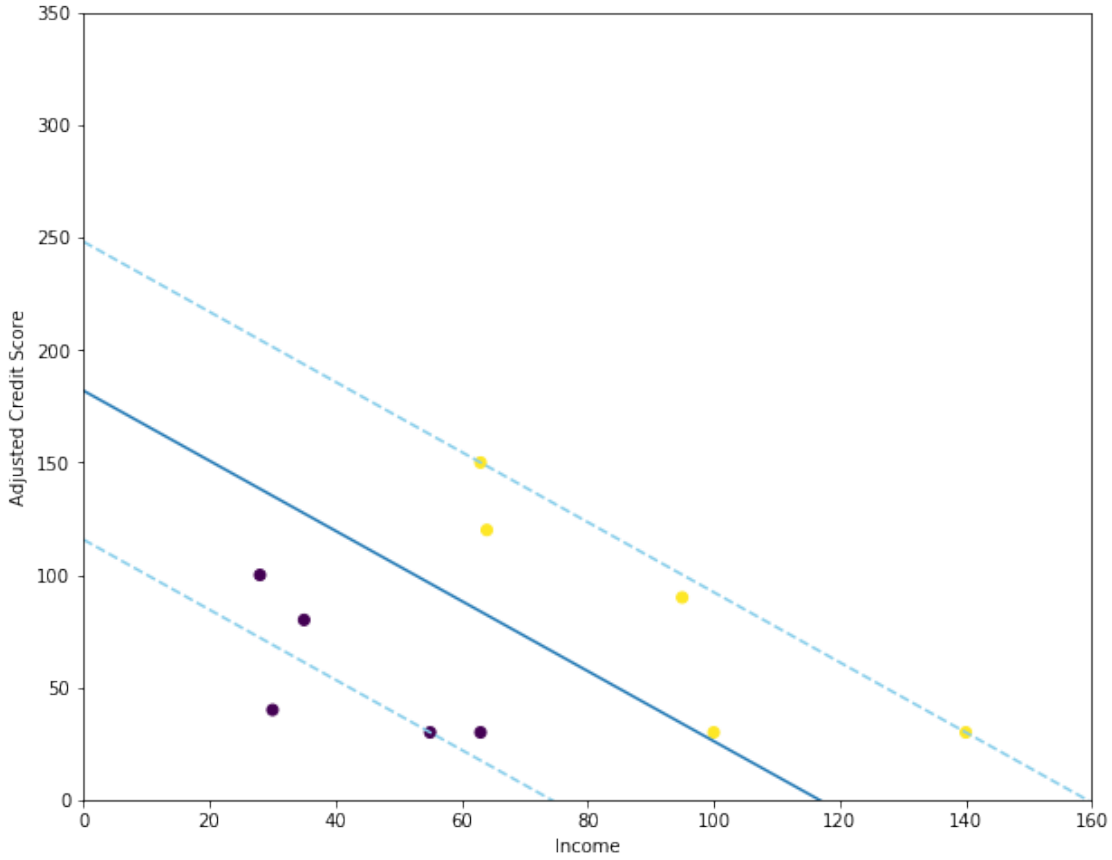
[12]:
```
#We carry out calculations for C=0.00015 which corresponds to C=0.0003 in Hull's␣
 ↪book.
X = np.asarray(pd.concat([Income, Credit],axis=1))
y = np.asarray(Loan).ravel()
clf = SVC(kernel='linear',C=0.00015,tol=1e-5)
clf.fit(X,y)
```

[12]:
```
SVC(C=0.00015, kernel='linear', tol=1e-05)
```

[13]:
```
w = clf.coef_[0]
b = -clf.intercept_[0]
print(w)
print(b)
```

```
[0.02352936 0.015098  ]
2.7470513941588437
```

[14]:
```
x1 = np.linspace(0,160,100)
w1 = w[0]
w2 = w[1]
bu = b+1
bd = b-1
y1 = (bu-w1*x1)/w2
y2 = (bd-w1*x1)/w2
y0 = (b-w1*x1)/w2
plt.figure(figsize=(10,8))
plt.scatter(Income, Credit, c=Loan)
plt.plot(x1,y1,'--',color='skyblue')
plt.plot(x1,y2,'--',color='skyblue')
plt.plot(x1,y0,'-')
plt.xlabel('Income')
plt.ylabel('Adjusted Credit Score')
axes = plt.gca()
axes.set_xlim([0,160])
axes.set_ylim([0, 350])
plt.show()
```



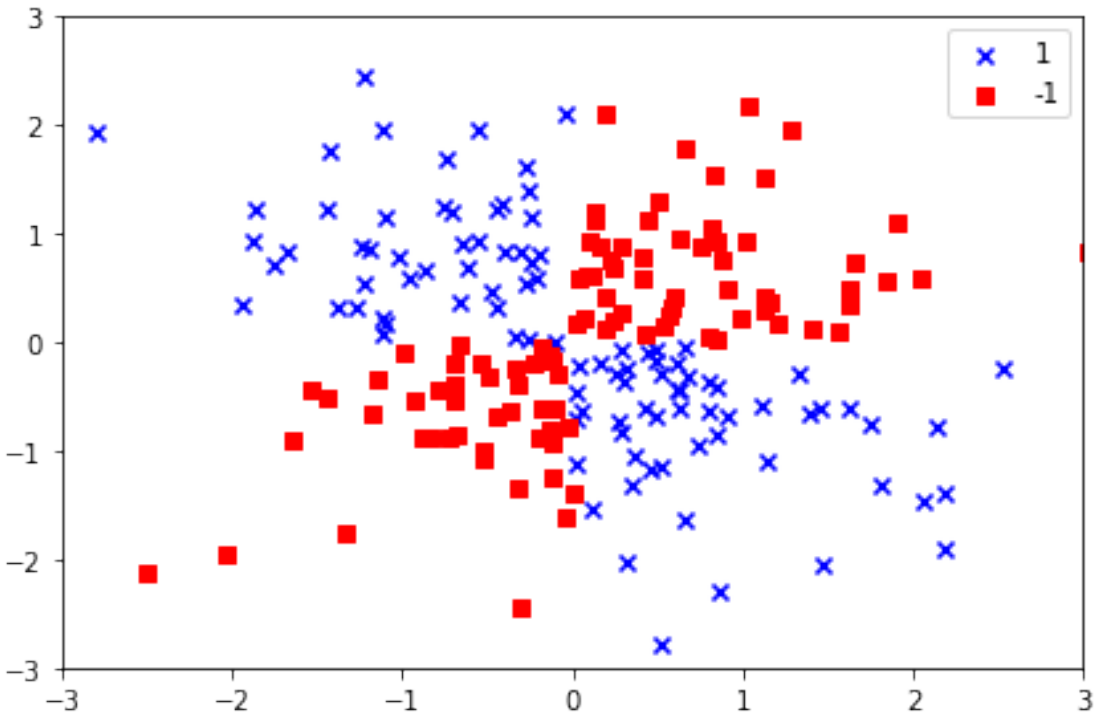

[ ]:

### 1.3 Non Linear Classification

Another reason why SVMs enjoy high popularity among machine learning practitioners is that they can be easily kernelized to solve nonlinear classification problems. Before we discuss the main concept behind the so-called kernel SVM, the most common variant of SVMs, let's first create a synthetic dataset to see what such a nonlinear classification problem may look like. Using the following code, we will create a simple dataset that has the form of an XOR gate using the logical_or function from NumPy, where 100 examples will be assigned the class label 1, and 100 examples will be assigned the class label -1:

[2]:
```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(1)
X_xor = np.random.randn(200, 2)
y_xor = np.logical_xor(X_xor[:, 0] > 0,
    X_xor[:, 1] > 0)
y_xor = np.where(y_xor, 1, -1)
plt.scatter(X_xor[y_xor == 1, 0],
```

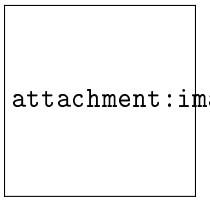
```
    X_xor[y_xor == 1, 1],
    c='b', marker='x',
    label='1')
plt.scatter(X_xor[y_xor == -1, 0],
    X_xor[y_xor == -1, 1],
    c='r',
    marker='s',
    label='-1')
plt.xlim([-3, 3])
plt.ylim([-3, 3])
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```




As we can see we can have data that is not linearly separable but which nonetheless falls neatly into two groups, just not groups that can be divided by a hyperplane. In this case the problem is not linearly separable. We cannot draw a straight line that can classify this data. But, may be that this data can be converted to linearly separable data in higher dimension. Lets add one more dimension and call it z-axis. For the data in Fig. we could go to three dimensions via the transformation

$$(x_1, x_2) \rightarrow (x_1, x_2, x_1^2 + x_2^2)$$

This allows us to separate the two classes shown in the plot via a linear hyperplane that becomes a nonlinear decision boundary if we project it back onto the original feature space.




### 1.3.1 Kernel Trick

**TO BE FINISHED**

Thanks to the dual formulation in (-) we can see that finding the $\alpha$s depends only on the products $\mathbf{x}^{(i)^T}\mathbf{x}^{(j)}$, and it's easy to observe that also the classification of new data depends on the product $\mathbf{x}^{(i)^T}\mathbf{u}$.

If we use $\Phi(\mathbf{x})$ to denote a transformation of the sample points (into higher dimensions) then the above observation show us that all we need to know for training and for classifying in the higher dimensional space are

$$\Phi\left(\mathbf{x}^{(i)}\right)^T \Phi\left(\mathbf{x}^{(j)}\right)$$

and

$$\Phi\left(\mathbf{x}^{(i)}\right)^T \Phi(\mathbf{u})$$

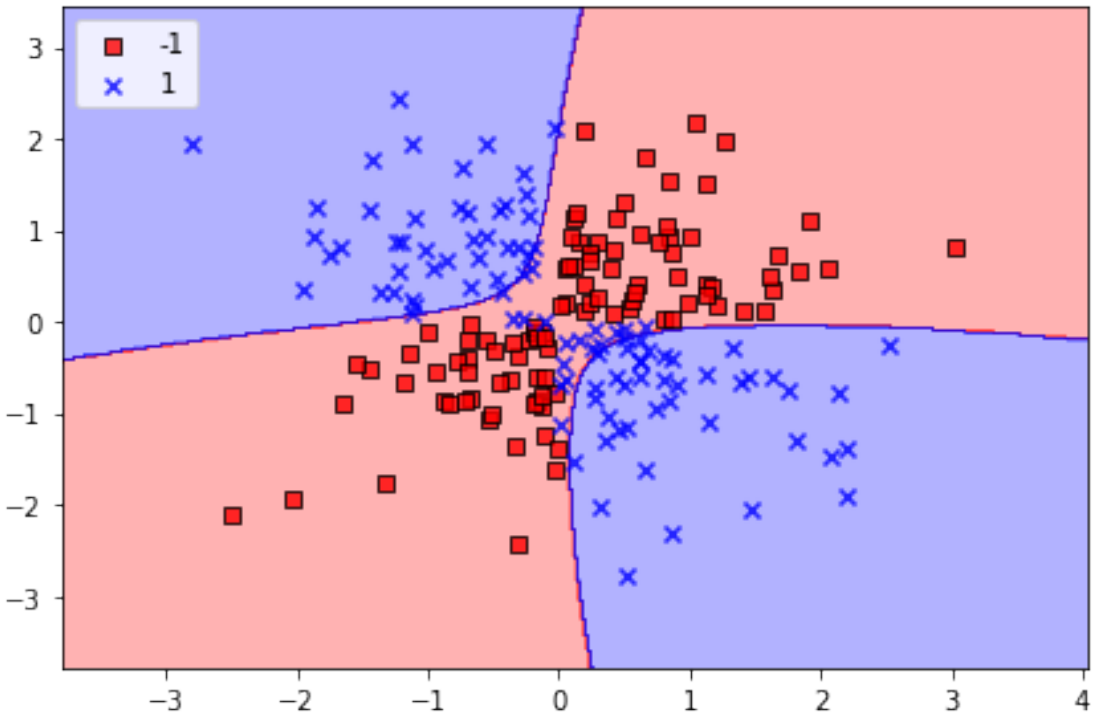
for our transformed data.

...

[13]:
```
from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),np.
 ↪arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
```

```
    plt.ylim(xx2.min(), xx2.max())
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.8,
                    c=colors[idx],
                    marker=markers[idx],
                    label=cl,
                    edgecolor='black')

    # highlight test examples
    if test_idx:
        # plot all examples
        X_test, y_test = X[test_idx, :], y[test_idx]
        plt.scatter(X_test[:, 0],
                    X_test[:, 1],
                    c='',
                    edgecolor='black',
                    alpha=1.0,
                    linewidth=1,
                    marker='o',
                    s=100,
                    label='test set')
```

[14]:
```
from sklearn.svm import SVC

svm = SVC(kernel='rbf', random_state=1, gamma=0.10, C=10.0)
svm.fit(X_xor, y_xor)
plot_decision_regions(X_xor, y_xor, classifier=svm)
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```




### 1.4 Predicting a Continuous Variable

**TO BE DONE**

[ ]: