Support Vector Machines (SVM)

NOTE: For this example, we will explore the algorithm, so we'll skip scaling and also skip a train\test split and instead see how the various parameters can change an SVM (easiest to visualize the effects in classification)

In [1]:
```
## Importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Data

The data shown here simulates a medical study in which mice infected with a virus were given various doses of two medicines and then checked 2 weeks later to see if they were still infected. Given this data, our goal is to create a classifcation model than predict (given two dosage measurements) if they mouse will still be infected with the virus.

You will notice the groups are very separable, this is on purpose, to explore how the various parameters of an SVM model behave.

In [2]:
```
## Importing the dataset
df = pd.read_csv("mouse_viral_study.csv")
df.head()
```

Out[2]:

	Med_1_mL	Med_2_mL	Virus Present
0	6.508231	8.582531	0
1	4.126116	3.073459	1
2	6.427870	6.369758	0
3	3.672953	4.905215	1
4	1.580321	2.440562	1

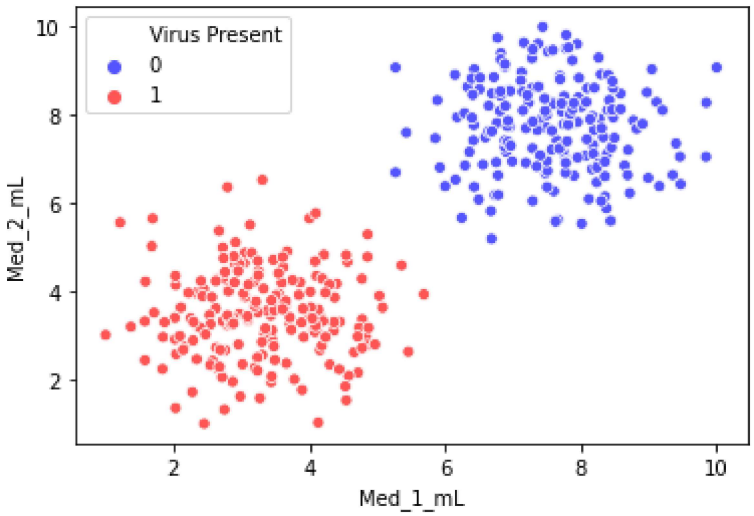
In [3]:
```
df.columns
```

Out[3]:
```
Index(['Med_1_mL', 'Med_2_mL', 'Virus Present'], dtype='object')
```

Classes

In [4]:
```
sns.scatterplot(x='Med_1_mL',y='Med_2_mL',hue='Virus Present',data=df,palette=
'seismic')
```

Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x1e63f5ff730>

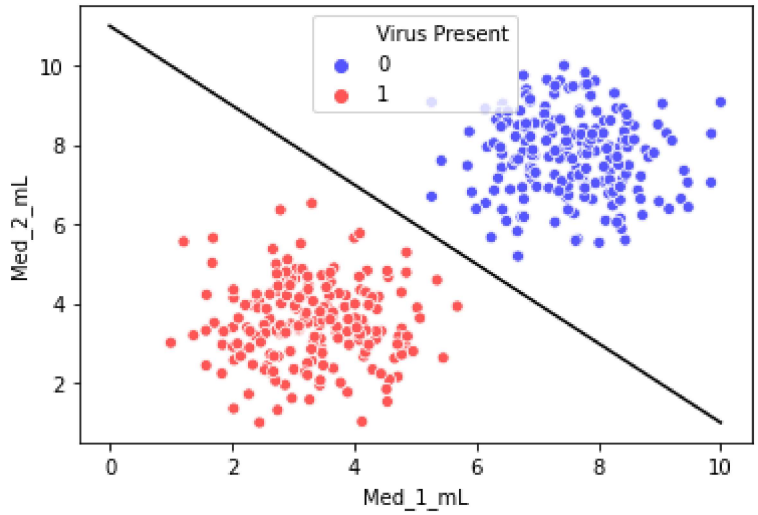


Separating Hyperplane

Our goal with SVM is to create the best separating hyperplane. In 2 dimensions, this is simply a line.

In [5]:
```
sns.scatterplot(x='Med_1_mL',y='Med_2_mL',hue='Virus Present',palette='seismic',data=df)

# We want to somehow automatically create a separating hyperplane ( a line in 2D)

x = np.linspace(0,10,100)
m = -1
b = 11
y = m*x + b
plt.plot(x,y,'k')
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x1e63fdcd6a0>]
```



In [12]:
```
y = df['Virus Present']
X = df.drop('Virus Present',axis=1)
```

NOTE: For this example, we will explore the algorithm, so we'll skip any scaling or even a train\test split for now

SVM - Support Vector Machine

In [7]:
```
from sklearn.svm import SVC # Supprt Vector Classifier
```

Hyper Parameters

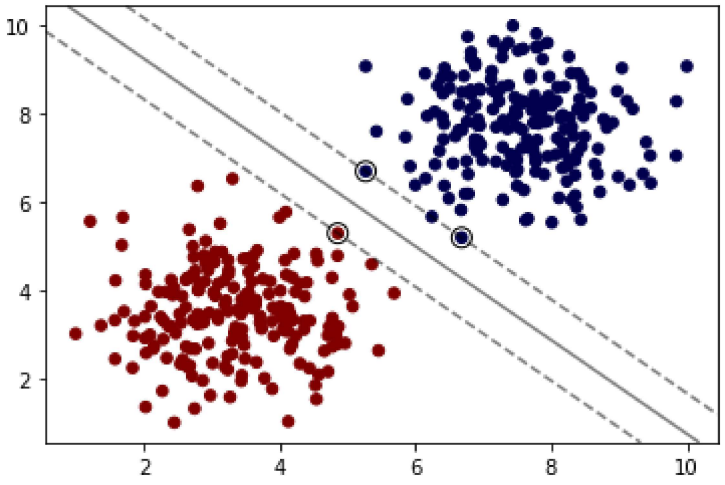
C

Regularization parameter. The strength of the regularization is **inversely** proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

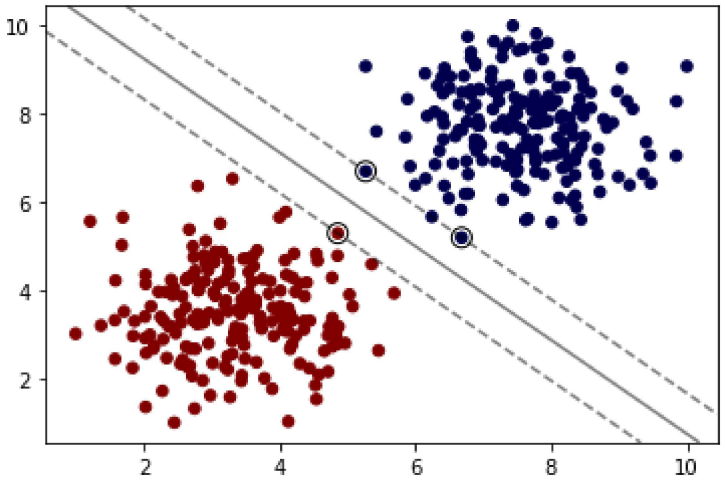
In [9]:
```
model = SVC(kernel='linear', C=1000)
```

Out[9]:
```
SVC(C=1000, kernel='linear')
```

In [10]:
```
    X = X.values
    y = y.values

    # Scatter Plot
    plt.scatter(X[:, 0], X[:, 1], c=y, s=30,cmap='seismic')


    # plot the decision function
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    xx = np.linspace(xlim[0], xlim[1], 30)
    yy = np.linspace(ylim[0], ylim[1], 30)
    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T
    Z = model.decision_function(xy).reshape(XX.shape)

    # plot decision boundary and margins
    ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])
    # plot support vectors
    ax.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1], s=1
00,
               linewidth=1, facecolors='none', edgecolors='k')
    plt.show()
```



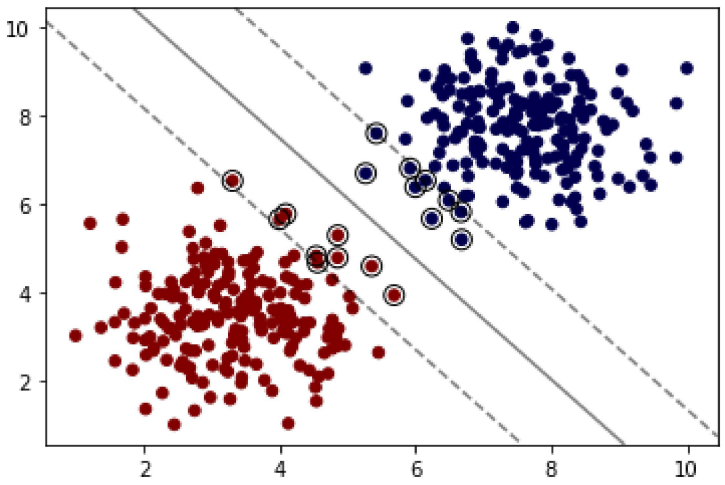
In [13]:
```
# This is imported from the supplemental .py file
from svm_margin_plot import plot_svm_boundary
plot_svm_boundary(model,X,y)
```



In [14]:
```
model = SVC(kernel='linear', C=0.05)
model.fit(X,y)
```

Out[14]:
```
SVC(C=0.05, kernel='linear')
```

In [15]:
```
plot_svm_boundary(model,X,y)
```



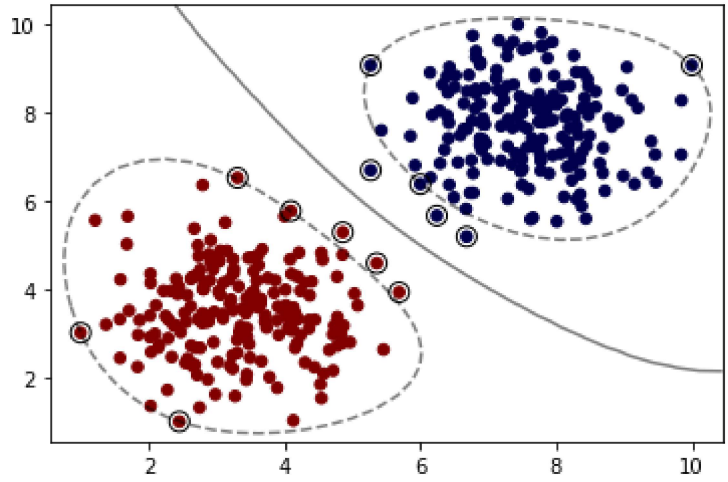
Kernel

Choosing a Kernel (https://stats.stackexchange.com/questions/18030/how-to-select-kernel-for-svm?rq=1)

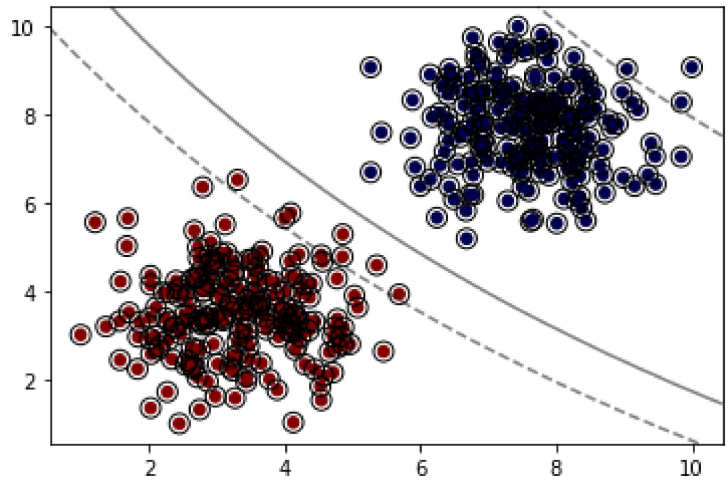
rbf - Radial Basis Function (https://en.wikipedia.org/wiki/Radial_basis_function_kernel)

When training an SVM with the Radial Basis Function (RBF) kernel, two parameters must be considered: C and gamma. The parameter C, common to all SVM kernels, trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly. gamma defines how much influence a single training example has. The larger gamma is, the closer other examples must be to be affected.

In [16]:
```
model = SVC(kernel='rbf', C=1)
model.fit(X, y)
plot_svm_boundary(model,X,y)
```



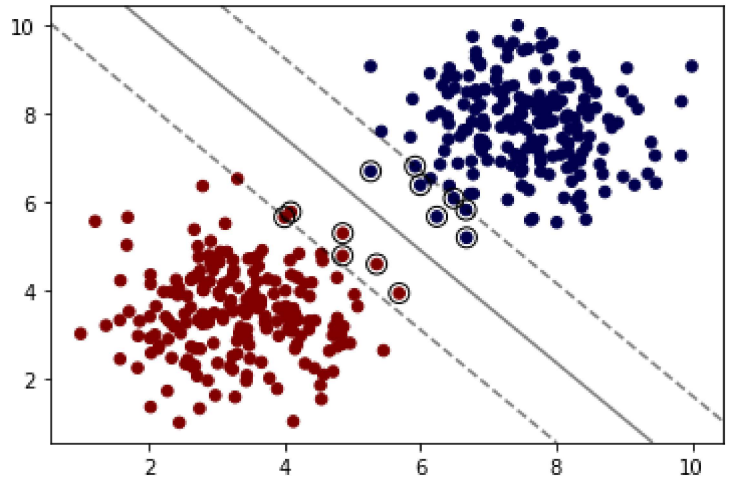
In [17]:
```
model = SVC(kernel='sigmoid')
model.fit(X, y)
plot_svm_boundary(model,X,y)
```



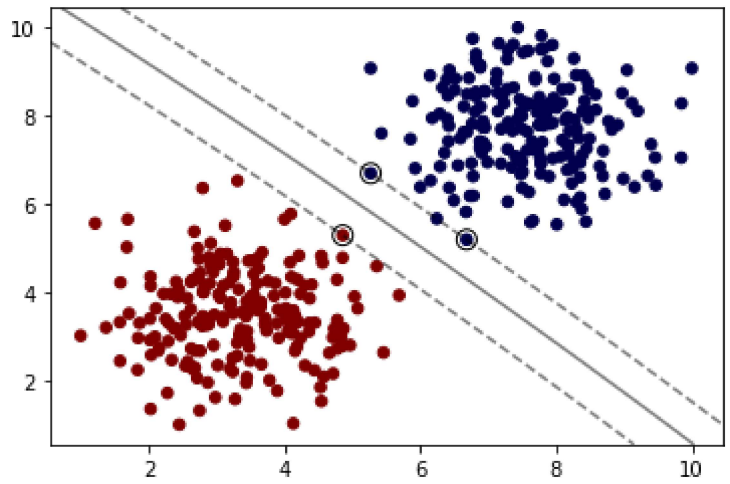
Degree (poly kernels only)

Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

In [18]:
```
model = SVC(kernel='poly', C=1,degree=1)
model.fit(X, y)
plot_svm_boundary(model,X,y)
```



In [19]:
```
model = SVC(kernel='poly', C=1,degree=2)
model.fit(X, y)
plot_svm_boundary(model,X,y)
```



gamma

gamma : {'scale', 'auto'} or float, default='scale' Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if ``gamma='scale'`` (default) is passed then it uses
  1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features.

In [ ]:
```
model = SVC(kernel='rbf', C=1,gamma=0.01)
model.fit(X, y)
plot_svm_boundary(model,X,y)
```

Grid Search

Keep in mind, for this simple example, we saw the classes were easily separated, which means each variation of model could easily get 100% accuracy, meaning a grid search is "useless".

In [20]:
```
from sklearn.model_selection import GridSearchCV
```

In [21]:
```
svm = SVC()
param_grid = {'C':[0.01,0.1,1],'kernel':['linear','rbf']}
```

In [22]:
```
# Note again we didn't split Train|Test
grid = GridSearchCV(svm,param_grid)
grid.fit(X,y)
```

Out[22]:
```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.01, 0.1, 1], 'kernel': ['linear', 'rbf']})
```

In [23]:
```
# 100% accuracy (as expected)
grid.best_score_
```

Out[23]:
```
1.0
```

In [24]:
```
grid.best_params_
```

Out[24]:
```
{'C': 0.01, 'kernel': 'linear'}
```