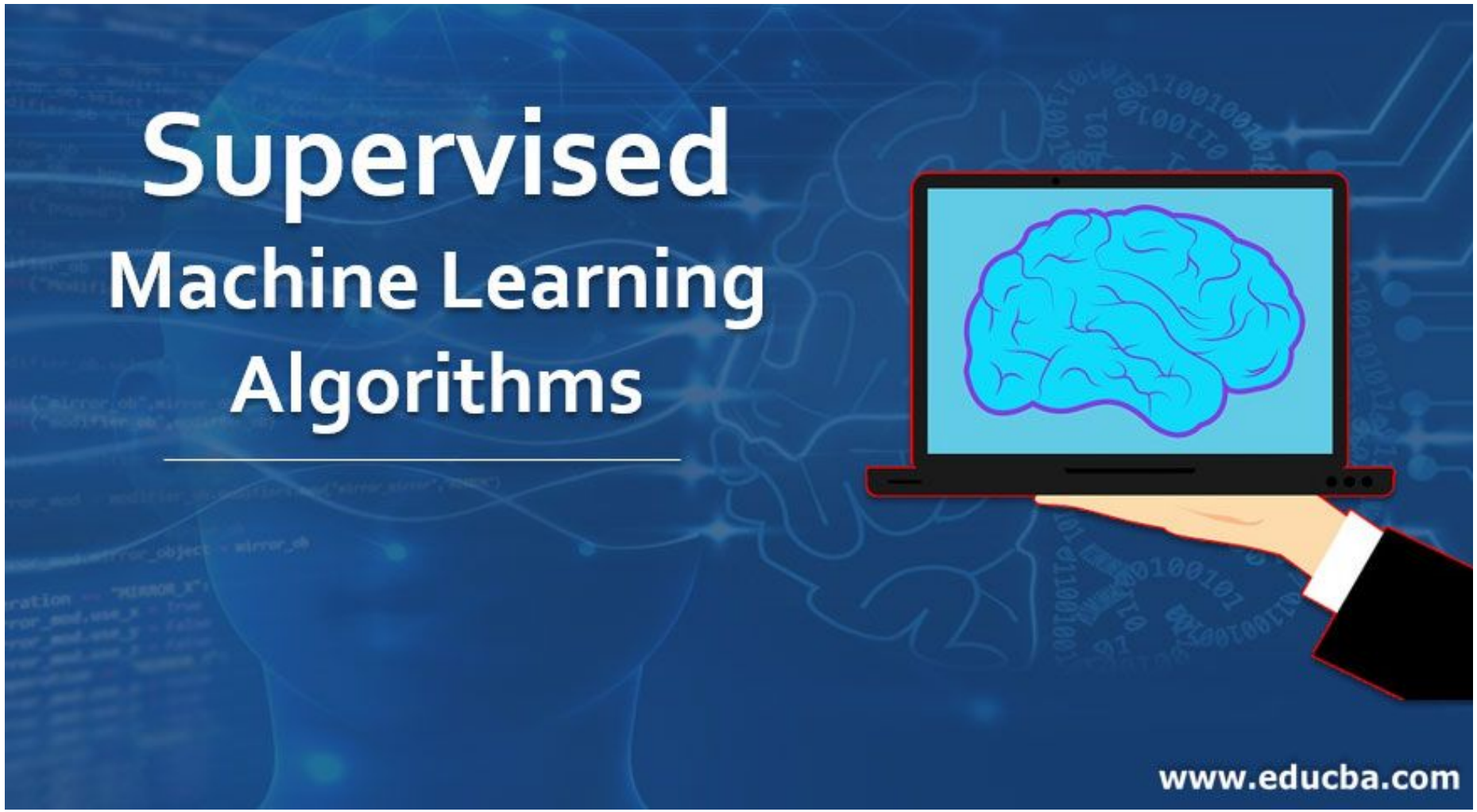

Supervised
Machine Learning
Algorithms
www.educba.com

Last lecture reminder

We learned about:

- KNN (K-Nearest Neighbors) - Choosing K value
- KNN - Elbow method
- KNN - Cross validation
- SVM (Support Vector Machines) - Introduction
- SVM - Theory
- SVM - Hard margin & Soft margin
- SVM - The kernel trick

SVM - Kernel Trick Computation

As we learned in the previous lecture, when it's hard for us to find a hyperplane that separate the data points into the different classes, we can use the kernel trick to project those points in the higher dimension.
The problem with that is that for large data sets we will need to perform a lot of computations in order to create those data points projections in the higher space.

The kernel trick also provide a much more efficient way for calculating those projections, so it will not damage our algorithm performance and resource consumption.

What the kernel trick is basically doing, Instead of explicitly computing and storing the coordinates of each point in the higher-dimensional space (which can be computationally expensive or even not feasible), the kernel trick computes only the scalar value (or dot products) between these transformed points (known as "the kernel function").

SVM Classification - Python Example

Let's see how to execute simple SVM model using Python and Scikit-Learn.

For the example we will use the '**mouse_viral_study.csv**' file that contain data about patients that takes 2 different medicine (features) and for each patient the dataset shows if a virus is present or not (categorical label).

```
In [5]: df = pd.read_csv('/Users/ben.meir/Downloads/mouse_viral_study.csv')
        df.head()
```

Out[5]:

	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

SVM Classification - Python Example

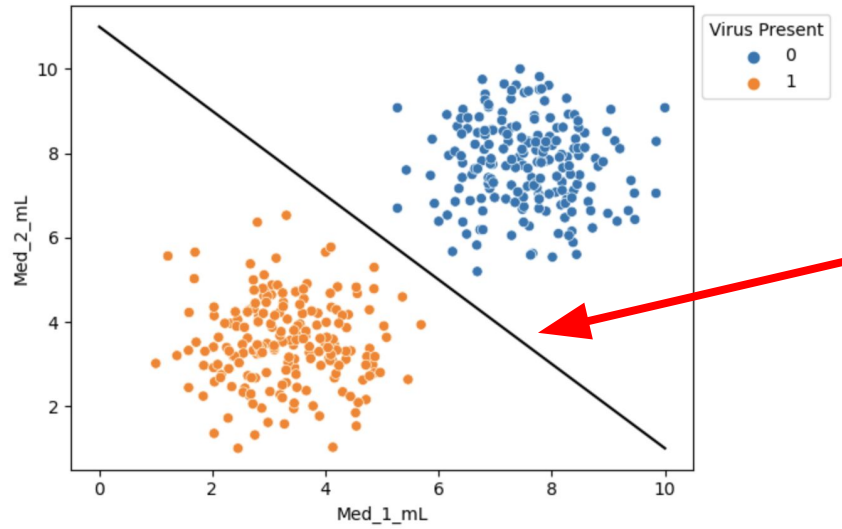
First, let's explore the data set and see if we can extract any insights by applying simple data analysis.

By generating a simple scatterplot on the data points, we can see that the data is perfectly separated between the two classes (virus present & virus not present).

We can also add a line that will separate the data points into those two class.

```
In [6]: sns.scatterplot(data=df, x='Med_1_mL', y='Med_2_mL', hue='Virus Present')
        plt.legend(title='Virus Present', bbox_to_anchor=(1,1))
        m = -1
        b = 11
        x = np.linspace(0,10,100)
        y = m * x + b

        plt.plot(x, y, 'black')
        plt.show()
```



This is not the SVM hyperplane separator. This is just a line we manually added to the scatterplot to demonstrate a hyperplane line that can separate the dataset into the different classes.

SVM Classification - Python Example

Now, let's configure our SVM model and try to classify new unseen points based on the dataset we have.

```
In [9]: from sklearn.svm import SVC

        df = pd.read_csv('/Users/ben.meir/Downloads/mouse_viral_study.csv')
        X = df.drop('Virus Present', axis=1)
        y = df['Virus Present']

        model = SVC(kernel='linear', C=1000)
        model.fit(X, y)

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

The SVM model parameters values

For the example we are training the model on the entire dataset without using the train / test split

Model parameters:

'kernel' → Represents the type of kernel that's used by the SVM model classifier. By specifying "linear", our model will use a linear kernel, which means that the SVM attempts to separate the classes by a linear hyperplane.

'C' → The penalty our model will add to its cost function when calculating the best fit hyperplane.

The larger the 'C' value, the higher the penalty on misclassification data points.

SVM Classification - Python Example

Once we finish to train our model we can use the **'plot_svm_boundary'** function for better visualization of the model hyperplane separator, the margins and the support vector points.

Simply copy the function code from the **'plot_svm_boundary.ipynb'** file to your jupyter notebook file.

In [2]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt


def plot_svm_boundary(model,X,y):

    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=100,
               linewidth=1, facecolors='none', edgecolors='k')
    plt.show()
```

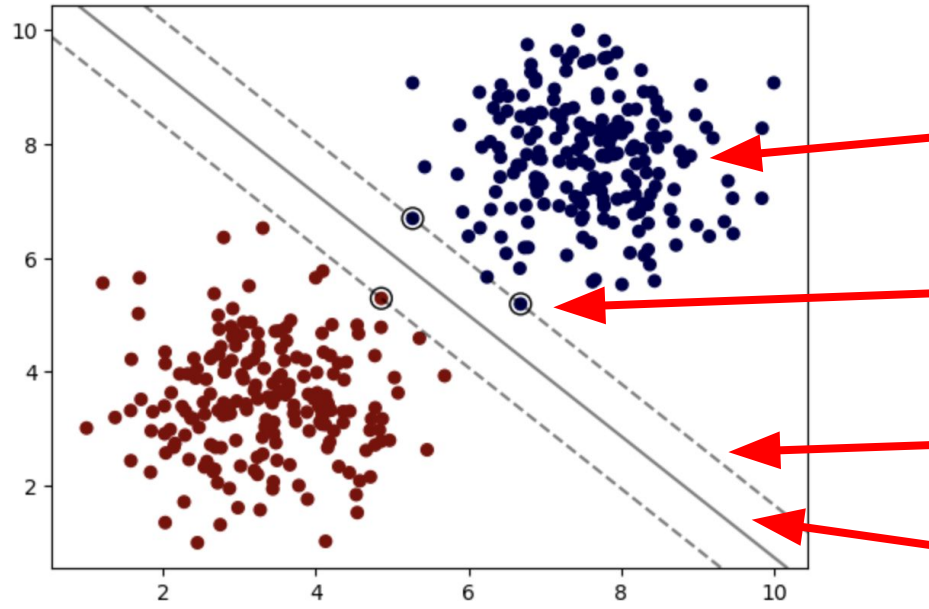
The 'plot_svm_boundary' function implementation code

SVM Classification - Python Example

Now that we have the **'plot_svm_boundary'** function code in our file, we can use this function to generate a scatterplot that will show our data points classification and the model best fit hyperplane. The function receive as parameters the SVM trained model and the features & label dataframes.

```
In [11]: plot_svm_boundary(model, X,y)
```

```
/Users/ben.meir/anaconda3/lib/python3.10/site-packages/sklearn/base.py:4
ure names, but SVC was fitted with feature names
  warnings.warn(
```



Classified data points from the dataset provided

The support vector points

The hyperplane margins

The trained SVM model best fit hyperplane

SVM Classification - Python Example

In the previous example we choosed a very high ‘C’ value, meaning the penalty we are giving for every misclassified point is very large. This caused our model to be with hard margin.

Let’s see what will happen if we will apply a very low ‘C’ value, meaning we allowing our model to misclassified some data points in order to broader the hyperplane margins (soft margin).

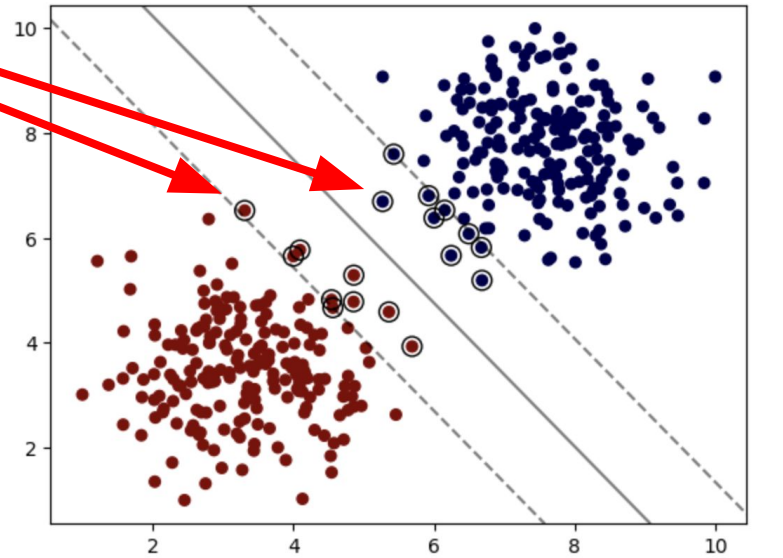
Points that we allow to be inside the margin and potentially be misclassified

```
In [12]: model = SVC(kernel='linear', C=0.05)
         model.fit(X,y)

Out[12]: ▾ SVC
         SVC(C=0.05, kernel='linear')
```

```
In [13]: plot_svm_boundary(model, X,y)
```

```
/Users/ben.meir/anaconda3/lib/python3.10/site-packages/sklearn/base
ure names, but SVC was fitted with feature names
  warnings.warn(
```



SVM Classification - RBF Kernel

In our previous example we choose the ‘linear’ kernel as the kernel function parameter and we saw that it performed well in creating a separator hyperplane. However, we need to remember that ‘linear’ kernel does not have the option to use the ‘kernel trick’ and project the point in a higher dimensional space.
The ‘linear’ kernel is trying to find the best fit hyperplane within the current dimension.

When we have data set that is not fit for ‘linear’ kernel, or in case we want to use the ‘kernel trick’ we need to select a different ‘kernel’ value.

Radial Basis Function (RBF) kernel → The Radial Basis Function (RBF) kernel is a popular choice of kernel function used in Support Vector Machine (SVM) and other kernelized models.
This is also the default kernel parameter value in the SVM models.
This kernel function perform the ‘kernel trick’ and allowing the model to project the data points in higher dimension in order to find the best fit separator hyperplane.

SVM Classification - RBF Kernel

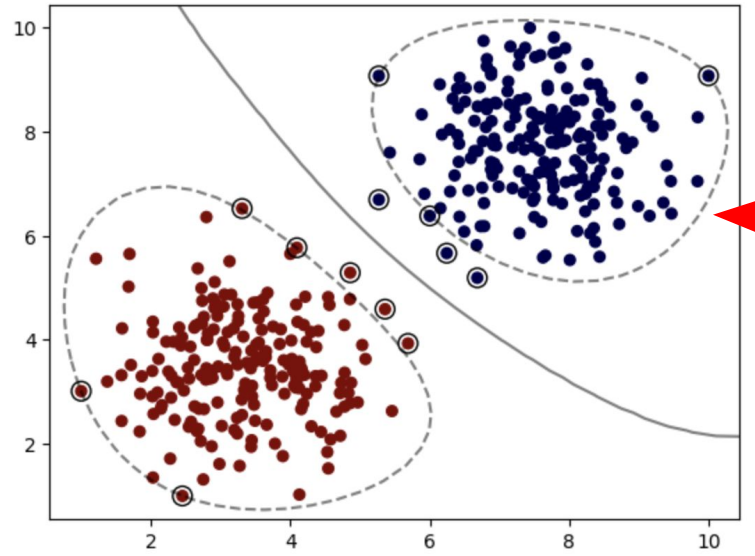
Let’s use our previous data set and apply the SVM model with the RBF kernel parameter.

```
In [22]: model = SVC(kernel='rbf', C=1)
         model.fit(X,y)

Out[22]: ▾ SVC
         SVC(C=1)
```

```
In [23]: plot_svm_boundary(model, X, y)

/Users/ben.meir/anaconda3/lib/python3.10/site-packages/sklearn/base
ure names, but SVC was fitted with feature names
  warnings.warn(
```



We can see that when using the ‘rbf’ kernel parameter we are getting margin and hyperplane separator that much more curve. This allowing us to get a hyperplane separator that is much more specific and a better fit to the data we provided

SVM Classification - Polynomial Kernel

Polynomial kernel → The polynomial kernel is another popular choice of kernel function in SVM model. Just like the Radial Basis Function (RBF) kernel, it helps create a non-linear decision boundary by projecting data points into a higher-dimensional space where they are separable.

Similar to RBF kernel, this is done more efficiently with the 'kernel trick' without the need for explicit mapping.

For example →

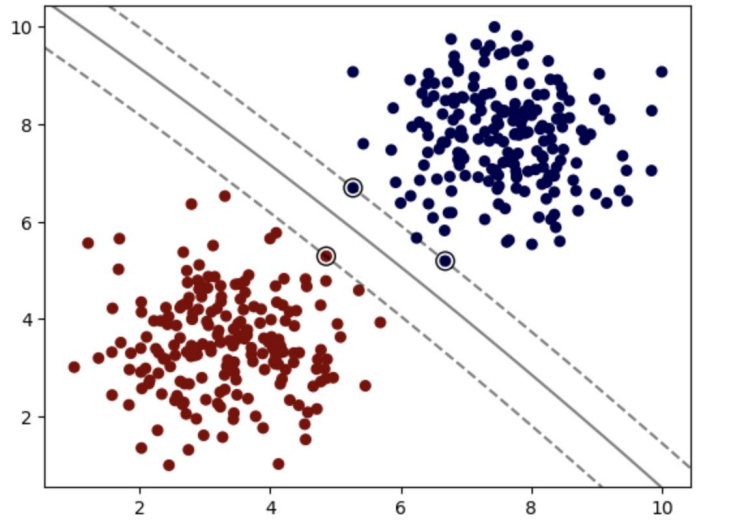
```
In [26]: model = SVC(kernel='poly', degree=3, C=1)
         model.fit(X,y)

Out[26]: SVC
         SVC(C=1, kernel='poly')
```

When using a ‘poly’ kernel type we need to also specify the degree we want the polynomial projection

```
In [25]: plot_svm_boundary(model, X, y)
```

```
/Users/ben.meir/anaconda3/lib/python3.10/site-packages/sklearn/base
ure names, but SVC was fitted with feature names
  warnings.warn(
```



SVM Classification - Grid Search

As we saw, there are multiple different parameters we can choose from when applying SVM model.
In most cases it will be hard for us to have any good intuition regarding which kernel type or what C value will be best fit for us.
In cases like this the best way is to perform grid search and let Python choose the optimal parameter values for us.

Grid search → Grid Search is a commonly used method for hyperparameter optimization.
Hyperparameters, in the context of Machine Learning, are parameters whose values are set before the learning process starts. For example, these could be the type of regularization (L1, L2) in logistic regression, or the C and gamma parameters in a Support Vector Machine (SVM).

SVM Classification - Grid Search

How grid search is working?

- First we define a set of possible values for each hyperparameter we want to tune. These sets of values define a "grid" over the hyperparameter space.
- The Grid Search then trains a separate model for each point in this hyperparameter space and evaluates it using cross-validation.
- Finally, it selects the hyperparameters that produced the model with the best cross-validation score (or another user-specified metric).

In conclusion, Grid Search allows us to try out multiple combinations of hyperparameters and find the combination that works best for our specific case.

Note: Grid Search can be computationally expensive, especially if the number of hyperparameters to tune and their potential values are large.

SVM Grid Search - Python Example

In order to find what should be the best kernel function and the best ‘C’ value let’s generate a grid search with those parameters.

```
In [29]: from sklearn.model_selection import GridSearchCV
         svm = SVC()

         param_grid = {'C': [0.01,0.1,1], 'kernel':['linear', 'rbf']}

         grid = GridSearchCV(svm, param_grid)
         grid.fit(X,y)

         grid.best_params_

Out[29]: {'C': 0.01, 'kernel': 'linear'}
```

We build a dictionary with the model parameters names and a list of values we want the grid to test for us.
In this example we want to check the kernel function and the optimal ‘C’ value

best_params_ attribute → The best_params_ attribute will provide a dictionary with each parameter we passed to the grid search and the optimal value the grid found.

ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

SVM Grid Search - Python Example

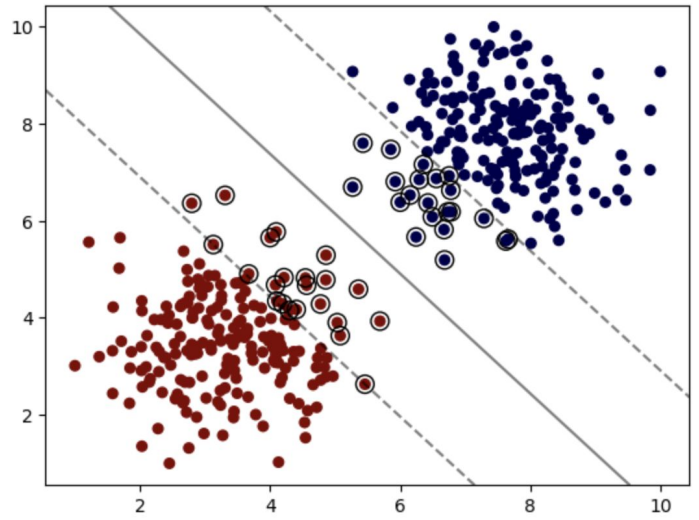
When generating the scatterplot according to the best fit params our grid search found we get the following plot:

```
In [31]: model = SVC(kernel='linear', C=0.01)
         model.fit(X,y)

Out[31]:  SVC
          SVC(C=0.01, kernel='linear')
```

```
In [32]: plot_svm_boundary(model, X, y)

/Users/ben.meir/anaconda3/lib/python3.10/site-packages/sklearn/base.py:4
ure names, but SVC was fitted with feature names
  warnings.warn(
```



SVM Grid Search - Python Example

Finally, let's create a full Python example for using the SVM model with the Grid Search and see the accuracy of our model.

In [39]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV


df = pd.read_csv('/Users/ben.meir/Downloads/mouse_viral_study.csv')
X = df.drop('Virus Present', axis=1)
y = df['Virus Present']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=101)

scaler = StandardScaler()
scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.transform(X_test)

svm = SVC()

param_grid = {'C': [0.01,0.1,1], 'kernel':['linear', 'rbf']}
grid = GridSearchCV(svm, param_grid)
grid.fit(scaled_X_train, y_train)

y_pred = grid.predict(scaled_X_test)
print("Optimal Params: ", grid.best_estimator_)
print("Accuracy: ", accuracy_score(y_test, y_pred))
```

```
Optimal Params:  SVC(C=0.01, kernel='linear')
Accuracy:  1.0
```

Applying Grid Search to find the best params for our SVM model

Print the optimal params values

Print the accuracy score of our model predictions.
In this case we got 1 meaning no misclassification were made

SVM - Continuous Data Prediction

SVM model can also be useful when trying to predict a continuous data value (and not predict data classification).

Support Vector Regression (SVR) → For continuous data the SVM model also known as SVR meaning support vector regression.
The reason the model works well also on continuous data is because instead of finding the best separator hyperplane, the model will try to find the best regression hyperplane that fits the data (similar to linear regression). SVR also using the 'kernel trick' allowing it to efficiently project the data points on a higher dimensional space.

So despite those differences, the underlying principles are very similar to the original classification problem. Both SVM and SVR aim to solve an optimization problem, using the same mathematical techniques, to find the best hyperplane that either separates classes or fits the data for regression.

Class Exercise - SVM

Instructions:

For this exercise use the **'diabetes_data.csv'** dataset. This dataset provides data about patients with their Body Mass Index (bmi), Blood Pressure (bp), and S1 (a type of serum measurement). It also contains a 'progression' column which is a quantitative measure of the disease progression in the patient. Your mission is to predict for future patients their disease progression based on the bmi, bp and S1 features.

Use the SVM model in order to predict the disease progression.

- Use the SVM model to perform continuous data prediction
- Use standardization on the data
- Use grid search to find the best parameters for ‘C’ and ‘kernel’
- Use MSE and RMSE metrics to evaluate your model
- Provide your model unseen data and predict the patient disease progression

Class Exercise Solution - SVM