The Support Vector Machine Algorithm

StartOnAI Machine Learning Tutorial 4

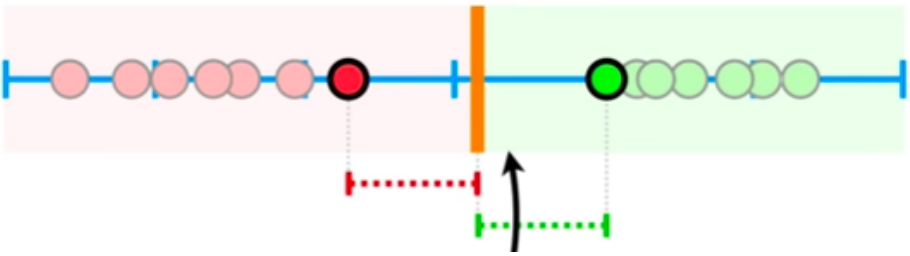
This is a tutorial which helps to break down the Support Vector Machine Algorithm.

What is the Support Vector Machine Algorithm?

An SVM or Support Vector Machine is a supervised machine learning algorithm that can be used for regression or classification problems. However, they are most commonly used for classification purposes. They can be quite useful, because they tend to be quite precise without using up too much computing power. Essentially, the goal of a Support Vector Machine is to find a hyperplane that lies in a space with "N" dimensions that can correctly classify a set of data points. When there is a linearly separable pattern within the data, an optimal hyperplane can be found without transforming it. However, patterns that are not linearly separable would require the data to be transformed into a space of higher dimensions. This can be done by a manipulation technique known as a Kernel Function.

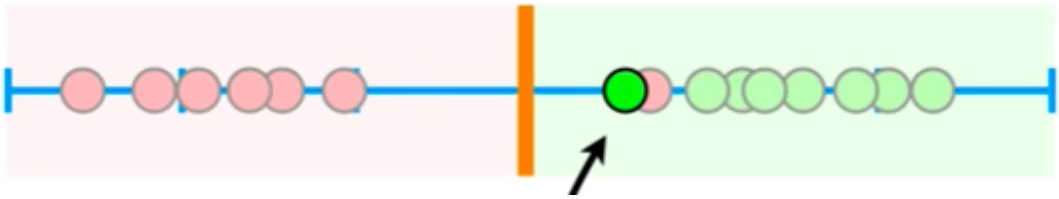
How do Support Vector Machines Work?

To understand why Support Vector Machines can be useful, it is practical to think about a maximal margin classifier. These classifiers use the edges of each cluster to make a threshold that becomes its basis of classification. This threshold is in the middle of each of the edges for the purpose of maximizing the margin, which would be the shortest distance between the observations and the threshold.



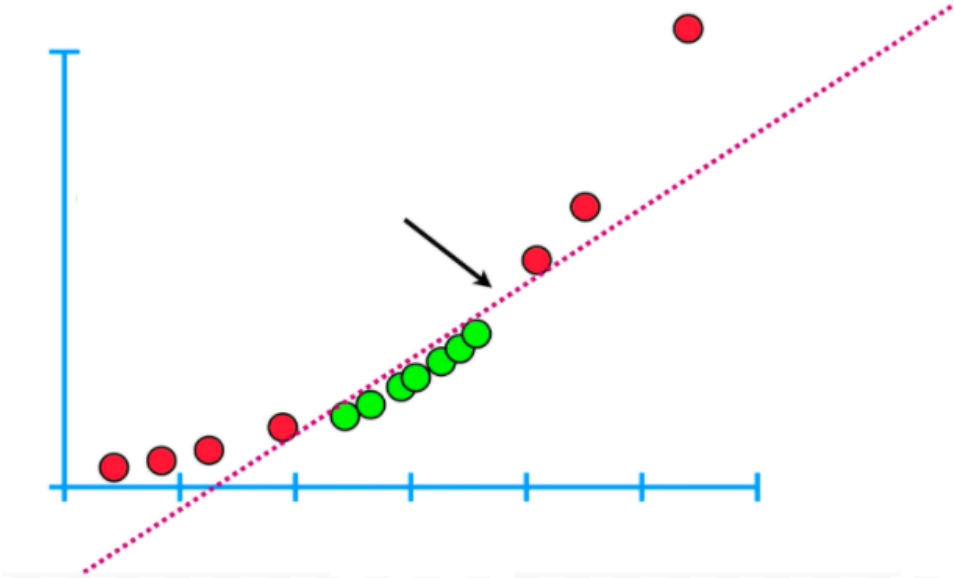
However, because this threshold is created by using the edges of each cluster, it can be very sensitive to outliers in a training data set. In turn, they are not very useful. In order to solve this issue, we must allow misclassifications. This technique is crucial to creating a good machine learning model because every good model has low bias. When choosing a threshold that is heavily influenced by an outlier, there can be a high risk for introducing bias. The term used for the distance between observations and the threshold when allowing misclassifications is known

as the Soft Margin. The data points, which are most difficult to classify and within the soft margin are known as Support Vectors.



But, what if the data has a lot of overlapping and is not linearly separable? A simple threshold cannot be drawn in one dimension to separate the data. This is where **Support Vector Machines** come into play.

In order to gain intuition behind the idea of a Support Vector Machine, we can transform the data into a two-dimensional space by adding a y-axis where the values are the initial(now the x-axis) values squared. Now, something amazing happens.



After simply moving the data one dimension higher, we can now classify the data into two different sets by finding a support vector classifier. However, how would we know to square the values of the initial data as opposed to cubing them or taking their square roots? This can be done via a kernel function. One specific kernel function known as the polynomial kernel takes data and continuously increases its dimensions. Everytime the dimensions are increased, a support vector classifier is found by calculating the relationships between different pairs of the data. Then, the kernel function determines the best value for the number of dimensions by evaluating each of the machine learning models via a process called cross validation. The models are trained using a large part of the data and then tested using a smaller part of the data.

[Note: Models should never be trained and tested using the same data because this can introduce bias.]

Apply Support Vector Machines to Classifying Iris Species

We will now apply Support Vector Machines in order to classify species of iris, which is a great classification problem. The flowers that we will classify are contained in the Iris dataset, which we will be importing later on. The dataset contains information about the flowers' Petal length and width, Sepal length and width, and what species of Iris flower these observations correspond to. The data has an ample amount of separation with moderate overlap, which screams Support Vector Machines! We will be importing the Iris dataset by using the Scikit-Learn Library.

Code Walkthrough

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
```

Here, we import the pandas library which we will later use to create a DataFrame. Then, we import the iris dataset from the Scikit-Learn Library.

We will now create a DataFrame out of this data set, because this gives us a better way of visualizing the data. The column headers will be the feature names of the iris flowers such as sepal length, petal length, etc. and the values inside the columns will be the data.

Code:

```
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df.head()
```

Output:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Code:
df['target'] = iris.target
df.head()

Output:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In this step, we append a column with the header "target" and the values within the column are target values 0, 1, or 2 found within the dataset. These target values correspond to different Iris species.

Code:
df.head()
df[df.target==1].head()
df[df.target==2].head()

From this series of steps, we find that the first 50 rows of data have target values of 0, the next 50 have target values of 1, and the next 50 have target values of 3.

Code:
Iris.target_names

Output:
array(['setosa', 'versicolor', 'virginica'],
dtype='<U10')

This step shows us that a target value of 0 corresponds to "setosa", a target value of 1 corresponds to "versicolor", and a target value of 2 corresponds to "virginica".

```
df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])
df.head()
```

Output:

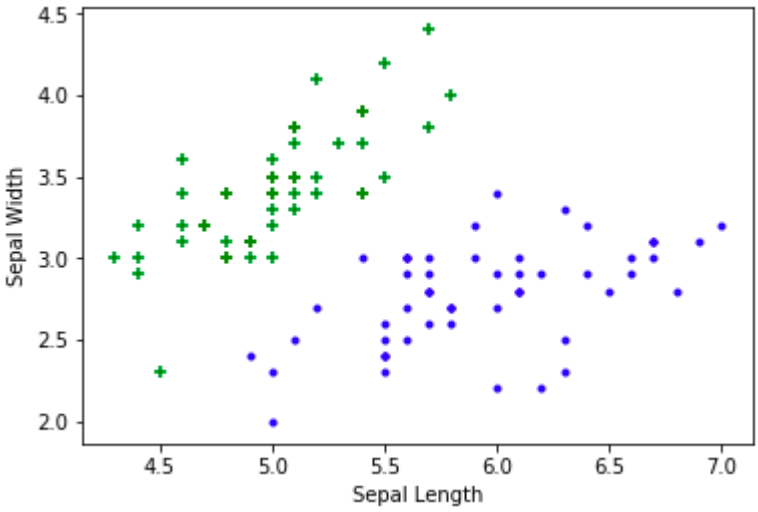
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

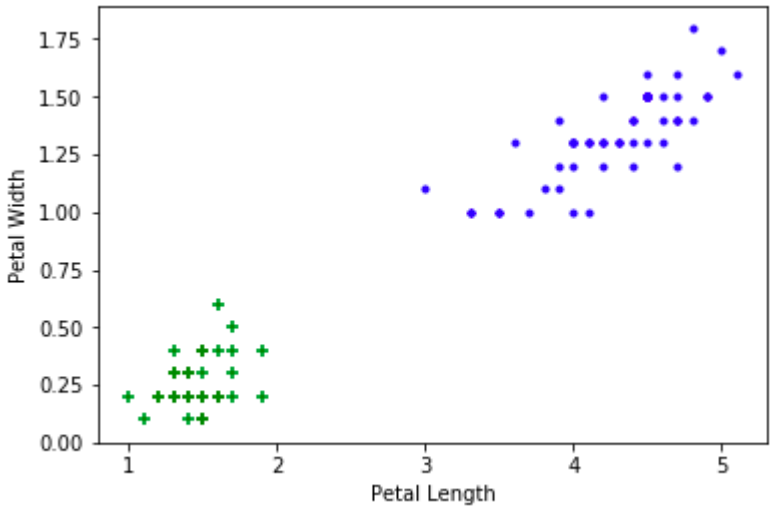
Using the "apply" function, we can generate a new column called "flower_name" from the target column. We also use the "lambda" function to transform the target column so that the values of the new column are the target values' corresponding flowers.

Code:

```
df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]
import matplotlib.pyplot as plt
%matplotlib inline
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'],color="green",marker='+')
plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'],color="blue",marker='.')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],color="green",marker='+')
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],color="blue",marker='.')
```

Output:





First, we separate the three different species into three different DataFrames. Then, we import the Matplotlib library so that we can create scatter plots. Finally, we create two scatter plots to represent data related to Sepal size and data related to Petal size.

```
from sklearn.model_selection import train_test_split
X = df.drop(['target','flower_name'], axis='columns')
y = df.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
from sklearn.svm import SVC
model = SVC(kernel='linear')
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

As always, we must split the dataset into a training dataset and a testing dataset. As stated earlier in the tutorial, we should never reuse the same data for training and testing because this may introduce bias. In this example, we use 20% of the data for testing. However, we only want to be training our model using the features so we drop the target and flower_name columns. Then, we can create a model using an SVM classifier and a linear kernel. After testing the model, adjustments can be made by tweaking the "gamma" or the "regularization" until a desirable score is achieved.

---

5. More Resources

Theory Behind Support Vector Machines
Some More Examples with Code
A Brief Explanation of SVMs
Going Into More Depth
A More Simple Explanation(In case the concept is hard to understand)

6. References

Codebasics. "Codebasics/Py." GitHub, github.com/codebasics/py/blob/master/ML/10_svm/10_svm.ipynb.

Starmer, Josh, director. *Support Vector Machines, Clearly Explained!!! Support Vector Machines, Clearly Explained!!!*, www.youtube.com/watch?v=efR1C6CvhmE.