Introduction

Support Vector Machines are one of the most popular and used machine learning algorithms. Exactly around the time, it was created, SVM became very popular and was applied to solve various machine learning problems.
In this post, you will learn about the support vector machine learning algorithm. Expectations after reading this post are that you get to know:

- What SVM.
- Different Areas in SVM are applicable to in real life.
- How SVM works.
- SVM kernels and how they are used.
- How to build a simple SVM model with the different kernels
- Hyperparameters that can be tuned for better performance.

SUPPORT VECTOR MACHINE

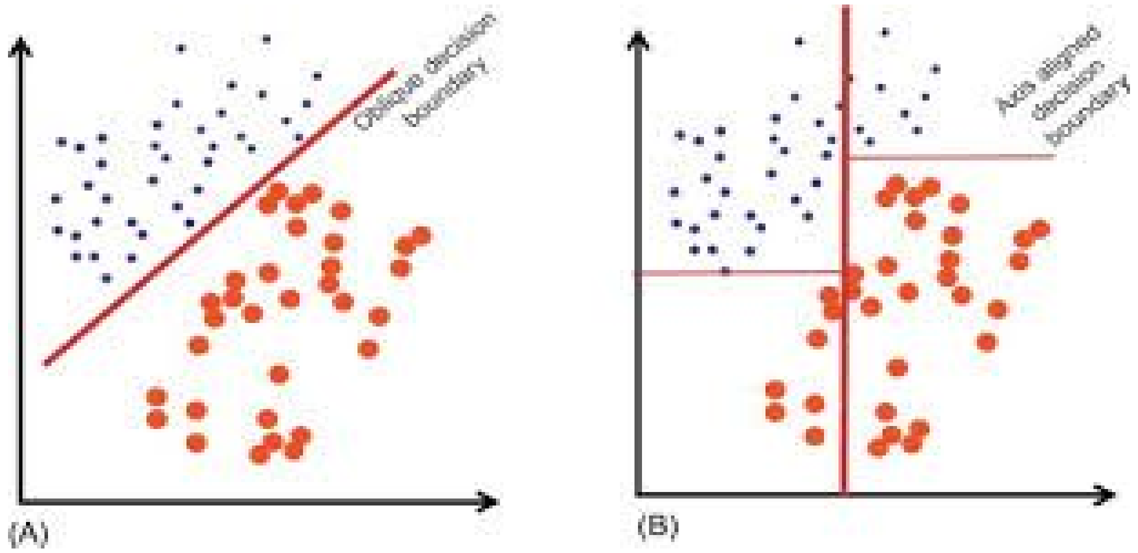
Support vector machine is also known as SVM, is a supervised machine learning algorithm used in making decisions in both classification and regression problems although it is mostly used for classification problems.
SVM main objective is to find the hyperplane or margin in an N-dimensional space that distinctly classifies the data point. This means it aims to find the best margin that makes our prediction both correct and confident.

How SVM works

To tell the story of how SVM works we will look into margins and decision boundaries.

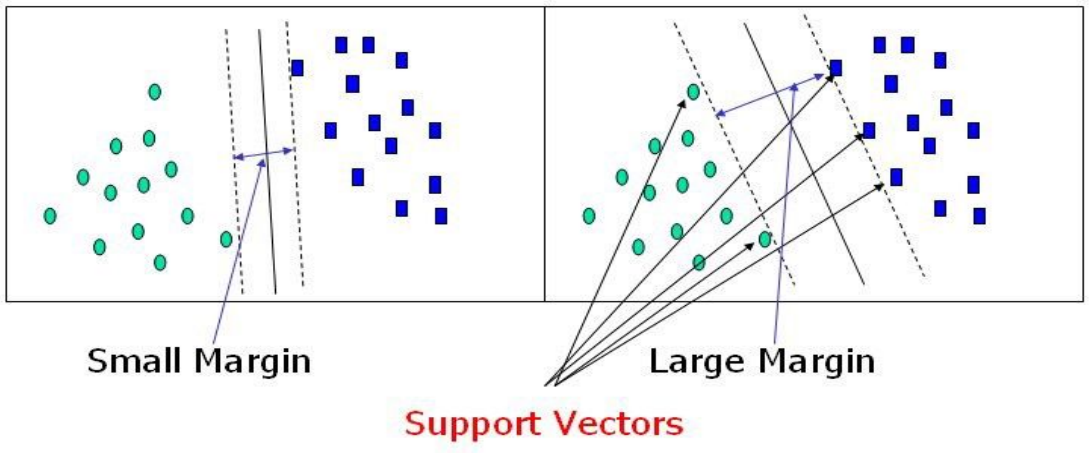
Decision Boundaries

A decision boundary is a hypersurface that partitions the underlying vector space into two sets, one for each class. The classifier will classify all the points on one side of the decision boundary as belonging to one class and all those on the other side as belonging to the other class.

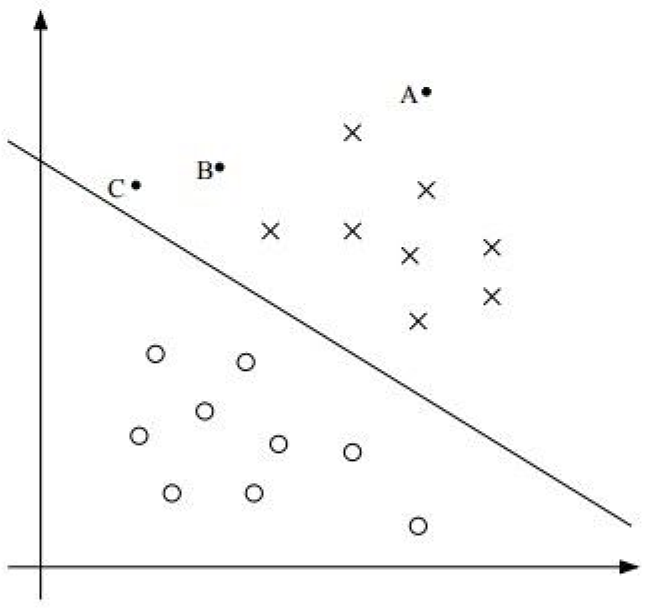


Margins

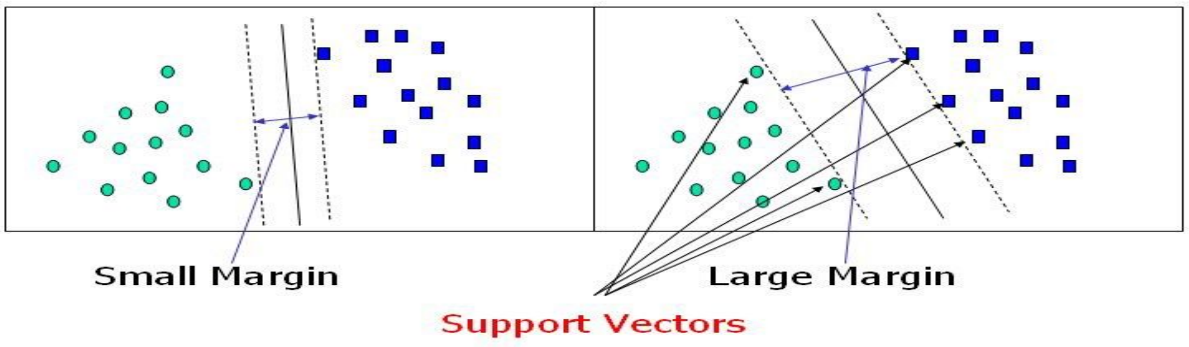
Margin is the distance between data points and a decision boundary, data points fall on either part of the margin to represent separate classes and the distance between a data point and a decision boundary indicates how confident the model is in predicting the class of that data point.



Let’s consider the figure below, where the x’s represents the positive training example and the o’s represents the negative training examples, there are also three points labelled A, B and C located at the side of the positive training examples.

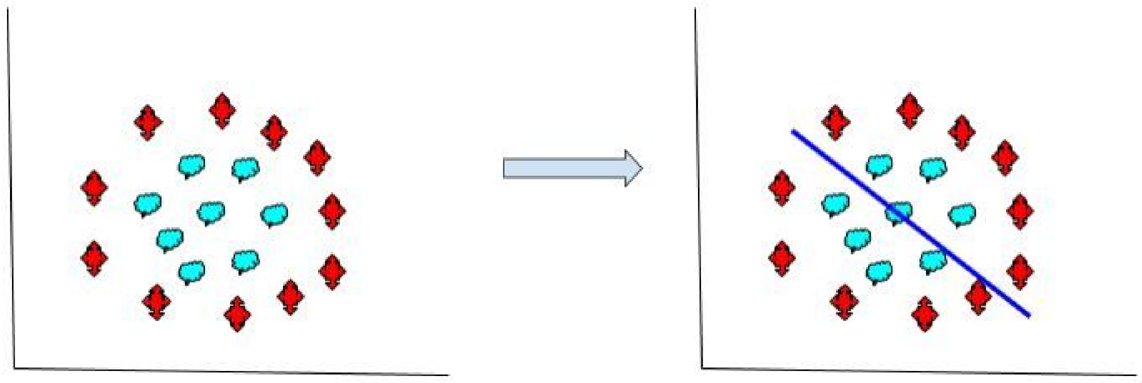


Notice the positions of the labelled points, point A has the largest distance from the decision boundary, C is very close to the line and B is close to the line and very far off from A. If we are to make a prediction for y at A, we would be very confident that y = 1 and not so confident for C, this is because point C can be easily influenced by any change to the decision boundary and see it’s prediction change from y = 1 to y = 0. If a point is far off from the decision boundary we are more confident at that prediction.
This is basically how SVM works, it tries to find the largest margin between the data points closest to the decision boundary, so that little changes to the decision boundary will not affect the prediction on that data point.

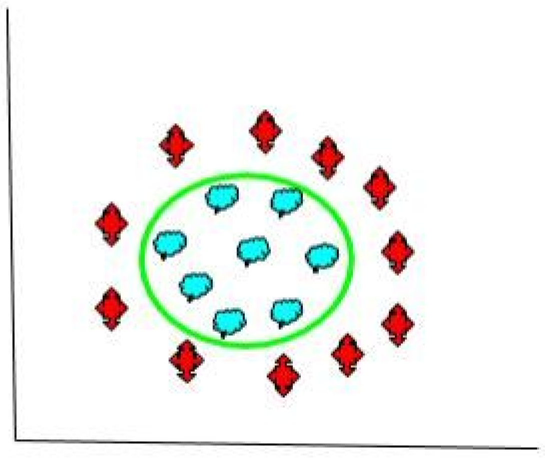


SVM Kernels

Kernels are set of mathematical functions that take in data and transform them into it's required form. Kernels are used in a non-linear data set that cannot be separated by the linear SVM decision boundary. Kernel maps the data to a higher dimension creating a non-linear decision boundary, allowing it to learn concepts not measured in the original data.



Applying linear SVM to a non-linear data



Applying a kernel to the linear SVM

Kernel functions are generally denoted by

Types of Kernel Functions

1. **Linear Kernel:** This kernel function does not transform the data, it is used when the data can be linearly separated, It can also work well in the case where we have a lot of features to deal with.

$$k(x, y) = x^T y + c$$

2. **Polynomial Kernel:** This function adds a simple non-linear transformation to the data. The polynomial kernel looks at the given input features to determine their similarities and combines these features together.

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

3. **RBF Kernel:** Radial basis function kernel (RBF) uses normal curves around the data points and sums them so the decision boundary can be defined by a type of topology condition.

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

4. **Sigmoid Kernel:** The sigmoid kernel comes from the neural network where the bipolar sigmoid function is often used as an activation function. An SVM using the sigmoid function is equivalent to a 2-layer perceptron neural network.

$$k(x, y) = \tanh(\alpha x^T y + c)$$

Applications of Support Vector Machine in Real World

SVM is used to correctly classify labelled unseen data, it is a supervised learning algorithm that is used to solve classification problems and has been applied to solve various problems, some listed below.

1. **Face Detection:** Place a relevant role in facial detection, It looks at an image and put a bounding box around the face indicating the facial part of the human body.
2. **Image Classification:** SVM is generally good for classification problems and it provides better accuracy for image classification problems.
3. **Bioinformatics:** Used in performing classification tasks on biological problems such as genes and cancer detection.
4. **Handwriting Recognition:** It has helped in identifying handwritings in documents and raw written papers.

There are many more use cases where you can find SVM has been implemented to help solve the problems. And also new problems come up every day so you can find one and solve with SVM.

Implementation of SVM

Now, I will show you how to implement a simple support vector machine algorithm on a simple dataset. Our implementation will cover all the things discussed above and we will see how this kernel changes will affect our predictions.

SVM is simple to implement and also straightforward. I will be using sklearn machine learning library to implement it.

Get the Data

Firstly is to get data. This is always the case for any machine learning problem. I will be using a staff promotion dataset gotten from a kaggle competition. The problem was to determine if an employee was fit for promotion or not, the dataset contained a class label indicating if an employee was promoted or not. This indicates a binary classification problem.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

The above code snippets import the packages we need, the NumPy and pandas packages for handling mathematical calculations and data processing, matplotlib for visualization, use train_test_split for train and test split then the SVM library from sklearn and accuracy score from sklearn metrics to calculate the accuracy of our model.

```
# read the data
data = pd.read_csv('train.csv')

# select only numeric datatypes
data = data.select_dtypes(['float64', 'int64'])

# separate the class and train data
class_label = data['Promoted_or_Not']
train_data = data.drop(data[['Promoted_or_Not']], axis=1)
)
#split the training data into train and test
X_train, X_test, y_train, y_test = train_test_split(train_data, class_label, test_size=0.3,
random_state=42)
```

In the above code snippets, we first read in the data, then select all the numeric features because machine learning models only work with numeric features. After, we get the column holding the class label called “Promoted_or_Not” and drop it from the overall data. Lastly, we split the data into train and test, we do this because we want to test the performance of our model on an unseen dataset.

Linear SVM

To create a linear SVM model, we just need to call the linear argument to the kernel parameter of the SVM model as seen below.

```
# instantiate the model
linear_svm = svm.SVC(kernel='linear')

# fit the model on the training data
linear_svm.fit(X_train, y_train)

# make predictions on train data
train_pred = linear_svm.predict(X_train)
# make prediction on test data
test_pred = linear_svm.predict(X_test)
```

We first need to instantiate the model then call the fit method on the train data and label to train the model on the data. To make predictions we use the svm.predict method on the train or test data as we deem fit.

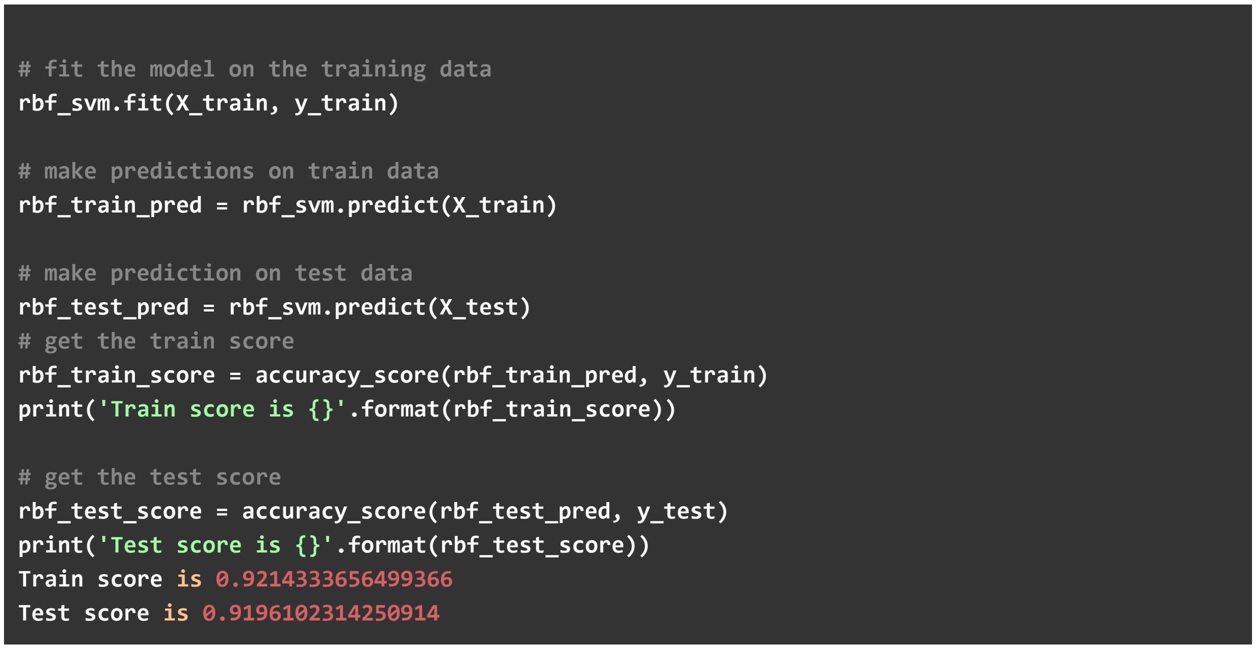
```
# get the train score
train_score = accuracy_score(prediction, y_train)
print('Train score is {}'.format(train_score))

# get the test score
test_score = accuracy_score(test_pred, y_test)
print('Test score is {}'.format(test_score))

Train score is 0.9155045118949959
Test score is 0.9151731338089438
```

The accuracy_score method is used to get the scores of the model on the predictions, in this case, the score on the train and test model are both at approximately 91%. There is no indication of overfitting as the train and test score are the same but we can use other metrics to determine how good our model is.

RBF SVM

Creating a SVM model with the RBF kernel is similar to the linear model, we just need to change the model parameter.

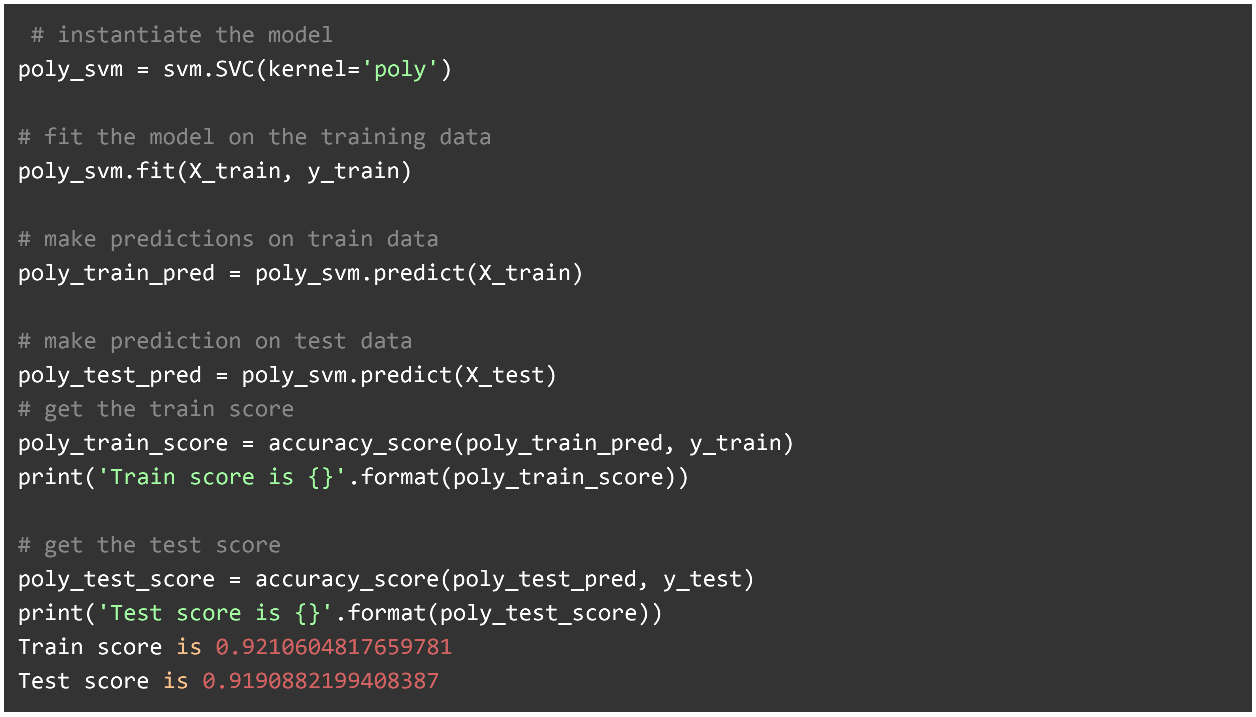
```
# instantiate the model
rbf_svm = svm.SVC(kernel='rbf')
```



```
# fit the model on the training data
rbf_svm.fit(X_train, y_train)

# make predictions on train data
rbf_train_pred = rbf_svm.predict(X_train)

# make prediction on test data
rbf_test_pred = rbf_svm.predict(X_test)
# get the train score
rbf_train_score = accuracy_score(rbf_train_pred, y_train)
print('Train score is {}'.format(rbf_train_score))

# get the test score
rbf_test_score = accuracy_score(rbf_test_pred, y_test)
print('Test score is {}'.format(rbf_test_score))
Train score is 0.9214333656499366
Test score is 0.9196102314250914
```

As we can see from the image above, we just changed the model kernel parameter and passed it the RBF kernel. Then we trained the model on the data and made predictions on the train and test data set.
The RBF model performed better than the linear model, it got approximately 92% on both the train and test score.

Poly SVM



```
 # instantiate the model
poly_svm = svm.SVC(kernel='poly')

# fit the model on the training data
poly_svm.fit(X_train, y_train)

# make predictions on train data
poly_train_pred = poly_svm.predict(X_train)

# make prediction on test data
poly_test_pred = poly_svm.predict(X_test)
# get the train score
poly_train_score = accuracy_score(poly_train_pred, y_train)
print('Train score is {}'.format(poly_train_score))

# get the test score
poly_test_score = accuracy_score(poly_test_pred, y_test)
print('Test score is {}'.format(poly_test_score))
Train score is 0.9210604817659781
Test score is 0.9190882199408387
```

Poly kernel gave a similar score to the RBF model. Train and test both got approximate accuracy scores of 92%.

Sigmoid SVM

```
# instantiate the model
sig_svm = svm.SVC(kernel='sigmoid')

# fit the model on the training data
sig_svm.fit(X_train, y_train)

# make predictions on train data
sig_train_pred = sig_svm.predict(X_train)

# make prediction on test data
sig_test_pred = sig_svm.predict(X_test)
# get the train score
sig_train_score = accuracy_score(sig_train_pred, y_train)
print('Train score is {}'.format(sig_train_score))

# get the test score
sig_test_score = accuracy_score(sig_test_pred, y_test)
print('Test score is {}'.format(sig_test_score))
Train score is 0.8583041240957566
Test score is 0.8547938054637202
```

The sigmoid kernel had the worst performance from all the kernels. It had an accuracy score of 85% on both the train and test data.

Hyperparameter Tuning

SVM has some parameters that can be tuned to provide better performance for your model.
Gamma: This is a hyperparameter used in non-linear hyperplanes, It is advisable to start with a low gamma value and move your way up and also keep it in mind that a high gamma value can cause your model to overfit.

```
gammas = [0.1, 1, 10, 100]
for gamma in gammas:
   tuned_linear_svc = svm.SVC(kernel='rbf', gamma=gamma)
   tuned_linear_svc.fit(X_train, y_train)
   pred = tuned_linear_svc.predict(X_test)
   print(accuracy_score(pred, y_test))

0.9191752218548808
0.9219592831042283
0.9166521663476597
0.9117800591613016
```

The above code snippet takes in a list of values and loops over the values. In the for loop, a

new SVM model is instantiated and passed the RBF and list of gamma parameters. The model is later fitted on the training set and prediction is made. From the scores, we see that gamma=1 produced the best score.
C: Is the parameter that penalizes the error, it controls the trade off between our decision boundary and correctly classified data points.

```
cs = [0.1, 1, 10, 100, 1000]
for c in cs:
  tuned_linear_c = svm.SVC(kernel='rbf', C=c)
  tuned_linear_c.fit(X_train, y_train)
  pred = tuned_linear_c.predict(X_test)
  print(accuracy_score(pred, y_test))
0.9189142161127545
0.9196102314250914
0.922568296502523
0.9215242735340178
0.9208282582216809
```

For the C parameter, C=10 had the best test score of 92% , and C=1000 took the longest time to run but performance was not up to C=10.

Degree: This parameter is specifically targeted to the poly kernel, it is the degree of polynomial used to find the decision boundary to split the data. Using a degree =1 is the same as using a linear kernel and the higher the degree the linger the model will train.

```
degrees = [0, 1, 2, 3, 4, 5, 6]
for degree in degrees:
  tuned_linear_d = svm.SVC(kernel='poly', degree=degree)
  tuned_linear_d.fit(X_train, y_train)
  pred = tuned_linear_d.predict(X_test)
  print(accuracy_score(pred, y_test))

0.9151731338089438
0.9151731338089438
0.919262223768923
0.9190882199408387
0.9194362275970072
0.9222202888463547
0.9213502697059335
```

Passed the model a list of degrees from 0 to 6, degree = 5 achieved the best accuracy score.

Further Reading

Obviously, I only touched on some parts of SVM, if you are interested in learning more about SVM there are numerous resources online in which you can learn from. I will list some useful resources I got from a blog post that might be of help.
Link to the notebook can be found here SVM implementation

- Support Vector Machine on Wikipedia
- Wikibook on Support Vector Machines
- What does the support vector machine (SVM) mean in layman's terms?
- Please explain Support Vector Machines (SVM) like I am a 5-year-old
- An Introduction to Statistical Learning: with Applications in R, Chapter 8
- Applied Predictive Modeling, Chapter 13
- The Elements of Statistical Learning: Data Mining, Inference, and Prediction Chapter 12.
- SVM Implementation Details
- SVM Documentation Sklearn

Summary

In this post, you learned about Support Vector Machine Algorithm for machine learning. You learned about:

- Support Vector Machines and how they work.
- Different Support Vector Machine Kernel types and when they are used.
- Implementation of Support Vector Machines and the parameters to tune.