Support Vector Machine (SVM)

On below plot, we have two features, petal length and petal width and based on that you can determine the species.

 Image Alt Text

Now, when your draw a boundry, you will notice that there are many possible ways of drawing this boundry as shown below and all these 3 are valid boundaries
 Image Alt Text

So how do you decide which boundry is the best for my classifiction problem?
Well,one way of looking at it is that you can take nearby data points and measure the distance from the line

 Image Alt Text

So, this distance is called 'Margin',
so which line is better? the one with the lower margin or higher margin?, well, the line with the higher margin is better cuz it classify these two groups in a better way

So this is what SVM try to do, it will try to maximize these margin between the nearby data points

 Image Alt Text

So these nearby data points is what we called 'Support Vectors' hence the name Support Vector Machine

 Image Alt Text

So, in case of a 2D space where you have two features, the boundry is a line

in case of 3D, the boundry is a plane

now what will it be if we have n dimensional space?

 Image Alt Text

usually we have n numbers of features, so if you think how the boundry will look like in n dimension space then you will realize that it is impossible to visualize it but mathemetically its possible and that boundry is called 'Hyperplane'.
So, Hyperplane is a plane in n dimensions that tried to seperate our different classification and thats what SVM tried to do

 t

Gamma


Image Alt Text

So as you can see above, the boundry is only considering the nearest data points, so this is one way of drawing a decision boundry, you can see that we excluded the far away points and only considered the nearest ones.
Now the other approach to solve the same problem as shown below could be that you consider the far away points as well


Image Alt Text

So on the left side we have a High Gamma and on the right side we have a Low Gamma and both the approaches are valid, its just that we low gamma sometimes you might get problem with accuracy but that might be okay, so both approaches are right, it depends on individual situation

Regularization

The other example, this is another daatset where we try to draw our boundry very carefully to avoid any classification error but you can see that it is almost overfitting the model, so if you have a complex dataset, this line might be like zigzag and it might try to overfit the model


Image Alt Text

On the other hand, we can take some errors which might be okay and our line would look more smoother

Image Alt Text

So on the left side we have a High Regularization and on right Low Regularization
With low regularization you might get some errors but that might be okay and even better for your model

When we use SKLearn to train our model, you will see a parameter called 'c' and c is Regularization basically

Kernel

you might have a complex dataset like below


Image Alt Text

So what do you do since here its very hard to draw a boundry.
 Image Alt Text

One approach is create a Third dimension The we doing it is that we are Doing some transformation on these features and creating a new feature 'z', and with that you will be able to draw this Decision boundry,
The Y plane is perpendicular to your monitor right now, thats why we are not able to see it, so the 'z' here is called the Kernel

 Image Alt Text

By Kernel what we mean is we are creating a transformation on your existing features so that we can draw a decision boundry easily

Coding Part

In [131…
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
```

In [132…
```
data = load_iris()
```

In [133…
```
dir(data)
```

Out[133]:
```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

In [134…
```
data.feature_names
```

Out[134]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

We will create a Data frame for this cuz its easier to visualize with a dataframe

In [135…
```
df = pd.DataFrame(data.data, columns=data.feature_names)
df.head()
```

Out[135]:

	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

In [136…
```
df['target'] = data.target
df.head()
```

Out[136]:

	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 [137…
```
data.target_names
```

Out[137]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [138…
```
df[df['target'] == 0].head()
```

Out[138]:

	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 [139…
```
df[df['target'] == 1].head()
```

Out[139]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

In [140…
```
df[df['target'] == 2].head()
```

Out[140]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

Adding new column flower name

so from one column, we are trying to generate another column, the way you do that in pandas is by using the apply function and here lambda is just a small function for the transformation that you can apply on target column and generate this new column, which basically mean for each value in target column, each value is x, so it will become index of value in target_names, so for example if its 2 then on index 2 in target_names is 'Virginica'

In [141…
```
df['flower name'] = df.target.apply(lambda x: data.target_names[x])
df
```

Out[141]:

	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
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2	virginica
146	6.3	2.5	5.0	1.9	2	virginica
147	6.5	3.0	5.2	2.0	2	virginica
148	6.2	3.4	5.4	2.3	2	virginica
149	5.9	3.0	5.1	1.8	2	virginica

150 rows × 6 columns

now, lets seperate these 3 species into three different dataframes as follows

In [142…
```
df0 = df[df.target==0]
df1 = df[df.target==1]
```

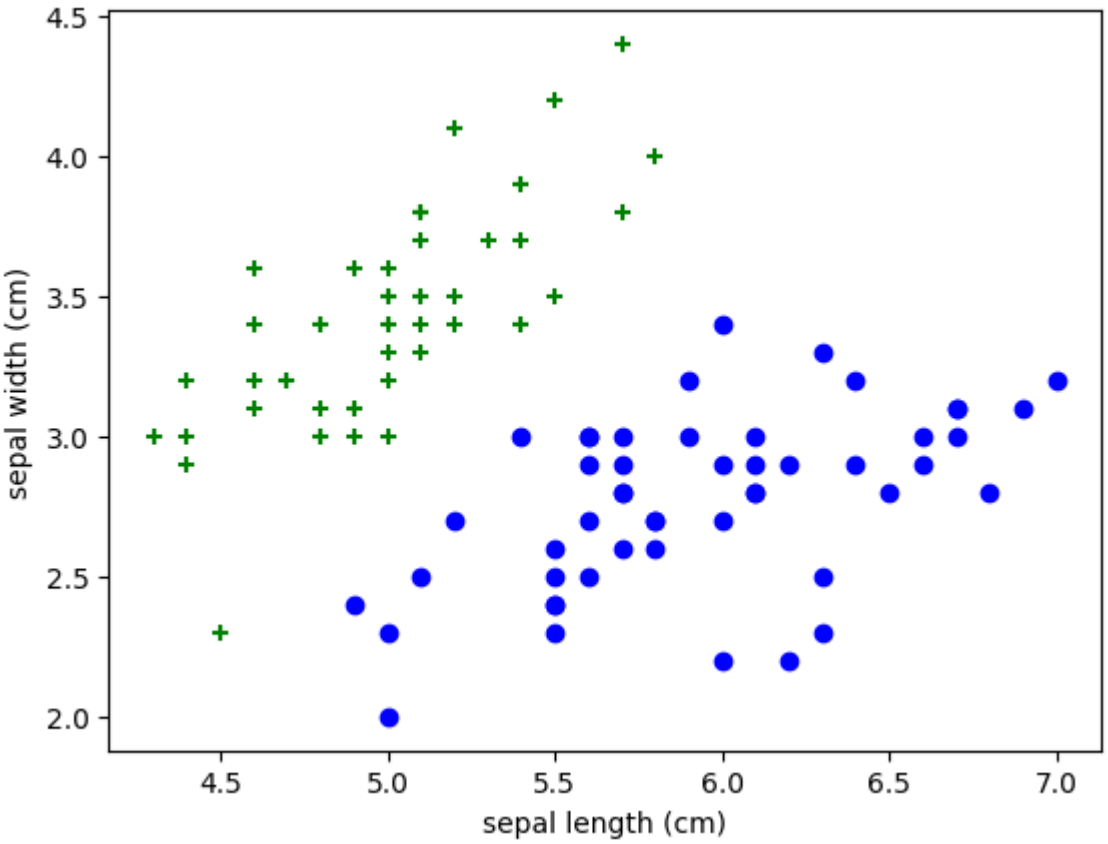
```
df2 = df[df.target==2]
```

Visualization

Drawing sepal length and width for df0 means setosa Drawing sepal length and width for df1 means versicolor

In [143…
```
plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'], color="green"
plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'], color="blue")
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
```

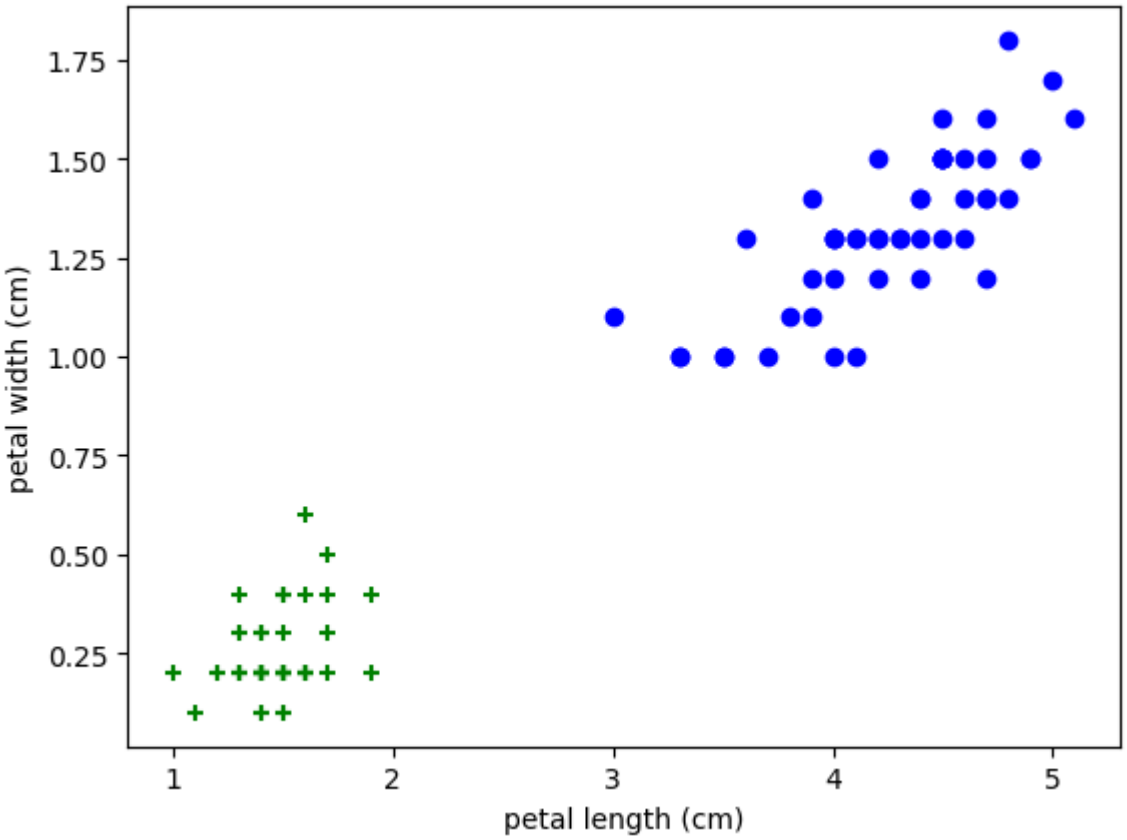
Out[143]: Text(0, 0.5, 'sepal width (cm)')



So we can see from above that theres a clear classification, so if we use a SVM algorithm model here, it will prolly draw a boundry from the middle. Now lets plot patel length and width also

In [144…
```
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'], color="green"
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'], color="blue")
plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)')
```

Out[144]: Text(0, 0.5, 'petal width (cm)')



As you can see above that these has even more clear distinction so looks like our SVM will be able to easily draw the line.

Here we only plotted on two features but in real world you will use all four features and all the species

In [145…
```
from sklearn.model_selection import train_test_split
```

In [146…
```
X = df.drop(['target', 'flower name'], axis=1)
y = df.target
```

In [147…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [148…
```
len(X_train)
```

Out[148]: 120

In [149…
```
len(X_test)
```

Out[149]: 30

In [150…
```
from sklearn.svm import SVC
```

In [151…
```
model = SVC()
```

In [152…
```
model.fit(X_train, y_train)
```

Out[152]:
```
▼ SVC
SVC()
```

In [153…
```
model.score(X_test, y_test)
```

Out[153]:
```
0.9666666666666667
```

In [158…
```
model.predict(X_test)
```

Out[158]:
```
array([0, 1, 2, 2, 2, 1, 2, 2, 2, 0, 1, 2, 0, 1, 2, 1, 1, 0, 0, 1, 2, 1,
       2, 0, 1, 1, 1, 1, 0, 1])
```

Tuning Parameters

Now you can use seperate parameters,

Use get_params() function to get all the parameters,

now, for example by default Regularization (c) is 1.0 so what if you modify the parameter, lets say you do c=10 or c=50, play with it and see which gives better accuracy

In [154…
```
model.get_params()
```

Out[154]:
```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': None,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

In [155…
```
model_C = SVC(C=10)
model_C.fit(X_train, y_train)
model_C.score(X_test, y_test)
```

Out[155]:
```
0.9666666666666667
```

So, you can use these parameters to tune your models, you can do this on couple of datasets using Cross validation technique and find what params is best suited for your problem

You can use gamma also

In [156…
```
model_g = SVC(gamma=10)
model_g.fit(X_train, y_train)
model_g.score(X_test, y_test)
```

Out[156]: 0.9333333333333333

Just tweak the value till you find highest accuracy

you can also use Kernel, so how do you know which kernel to pass? just use shift + tab and it will show all the info and the possible kernel you have available, we will just use linear

In [157…
```
model_linear_kernal = SVC(kernel='linear')
model_linear_kernal.fit(X_train, y_train)
model_linear_kernal.score(X_test, y_test)
```

Out[157]: 0.9666666666666667

Conclusion

As we can see, seems like our accuracy doesnt go beyond 0.96, so this is the highest accuracy we could get by tweaking the params