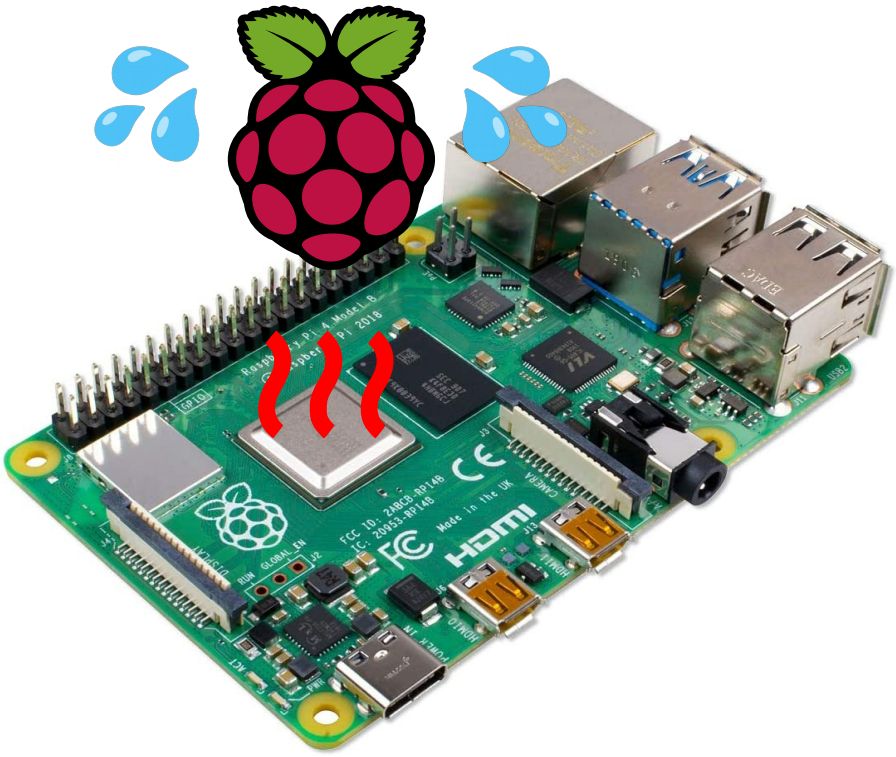
Playing with parameters of Support Vector Classifiers (SVC)

Björn Kasper (kasper.bjoern@bgetem.de)

August 2, 2022



This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

This is a test abstract.

Contents

1 Introduction 2

2 Vary kernel 7

3 Vary gamma 9

4 Vary C 11

5 degree 14

1 Introduction

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import svm
%matplotlib inline
```

[2]:
```
# import some data to play with
irisdata_df = pd.read_csv('../datasets/IRIS_flower_dataset_kaggle.csv')

# encode the class column from class strings to integer equivalents
irisdata_df_enc = irisdata_df.replace({"species":  {"Iris-setosa":
 ↪0,"Iris-versicolor":1, "Iris-virginica":2}})
irisdata_df_enc
```

[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
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
..	…	…	…	…	…
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

[150 rows x 5 columns]

[3]:
```
# copy only 2 feature columns
# and convert pandas dataframe to numpy array
X = irisdata_df_enc[['petal_length', 'petal_width']].to_numpy(copy=True)
#X = irisdata_df_enc[['sepal_length', 'sepal_width']].to_numpy(copy=True)
X
```

[3]:
```
array([[1.4, 0.2],
       [1.4, 0.2],
       [1.3, 0.2],
       [1.5, 0.2],
       [1.4, 0.2],
```

```
[1.7, 0.4],
[1.4, 0.3],
[1.5, 0.2],
[1.4, 0.2],
[1.5, 0.1],
[1.5, 0.2],
[1.6, 0.2],
[1.4, 0.1],
[1.1, 0.1],
[1.2, 0.2],
[1.5, 0.4],
[1.3, 0.4],
[1.4, 0.3],
[1.7, 0.3],
[1.5, 0.3],
[1.7, 0.2],
[1.5, 0.4],
[1. , 0.2],
[1.7, 0.5],
[1.9, 0.2],
[1.6, 0.2],
[1.6, 0.4],
[1.5, 0.2],
[1.4, 0.2],
[1.6, 0.2],
[1.6, 0.2],
[1.5, 0.4],
[1.5, 0.1],
[1.4, 0.2],
[1.5, 0.1],
[1.2, 0.2],
[1.3, 0.2],
[1.5, 0.1],
[1.3, 0.2],
[1.5, 0.2],
[1.3, 0.3],
[1.3, 0.3],
[1.3, 0.2],
[1.6, 0.6],
[1.9, 0.4],
[1.4, 0.3],
[1.6, 0.2],
[1.4, 0.2],
[1.5, 0.2],
[1.4, 0.2],
[4.7, 1.4],
[4.5, 1.5],
[4.9, 1.5],
[4. , 1.3],
[4.6, 1.5],
[4.5, 1.3],
[4.7, 1.6],
[3.3, 1. ],
[4.6, 1.3],
[3.9, 1.4],
[3.5, 1. ],
[4.2, 1.5],
```

```
[4. , 1. ],
[4.7, 1.4],
[3.6, 1.3],
[4.4, 1.4],
[4.5, 1.5],
[4.1, 1. ],
[4.5, 1.5],
[3.9, 1.1],
[4.8, 1.8],
[4. , 1.3],
[4.9, 1.5],
[4.7, 1.2],
[4.3, 1.3],
[4.4, 1.4],
[4.8, 1.4],
[5. , 1.7],
[4.5, 1.5],
[3.5, 1. ],
[3.8, 1.1],
[3.7, 1. ],
[3.9, 1.2],
[5.1, 1.6],
[4.5, 1.5],
[4.5, 1.6],
[4.7, 1.5],
[4.4, 1.3],
[4.1, 1.3],
[4. , 1.3],
[4.4, 1.2],
[4.6, 1.4],
[4. , 1.2],
[3.3, 1. ],
[4.2, 1.3],
[4.2, 1.2],
[4.2, 1.3],
[4.3, 1.3],
[3. , 1.1],
[4.1, 1.3],
[6. , 2.5],
[5.1, 1.9],
[5.9, 2.1],
[5.6, 1.8],
[5.8, 2.2],
[6.6, 2.1],
[4.5, 1.7],
[6.3, 1.8],
[5.8, 1.8],
[6.1, 2.5],
[5.1, 2. ],
[5.3, 1.9],
[5.5, 2.1],
[5. , 2. ],
[5.1, 2.4],
[5.3, 2.3],
[5.5, 1.8],
[6.7, 2.2],
[6.9, 2.3],
```

```
       [5. , 1.5],
       [5.7, 2.3],
       [4.9, 2. ],
       [6.7, 2. ],
       [4.9, 1.8],
       [5.7, 2.1],
       [6. , 1.8],
       [4.8, 1.8],
       [4.9, 1.8],
       [5.6, 2.1],
       [5.8, 1.6],
       [6.1, 1.9],
       [6.4, 2. ],
       [5.6, 2.2],
       [5.1, 1.5],
       [5.6, 1.4],
       [6.1, 2.3],
       [5.6, 2.4],
       [5.5, 1.8],
       [4.8, 1.8],
       [5.4, 2.1],
       [5.6, 2.4],
       [5.1, 2.3],
       [5.1, 1.9],
       [5.9, 2.3],
       [5.7, 2.5],
       [5.2, 2.3],
       [5. , 1.9],
       [5.2, 2. ],
       [5.4, 2.3],
       [5.1, 1.8]])
```

[4]:
```
# convert pandas dataframe to numpy array
# and get a flat 1D copy of 2D numpy array
y = irisdata_df_enc[['species']].to_numpy(copy=True).flatten()
y
```

[4]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], dtype=int64)
```

[5]:
```
svc = svm.SVC(kernel='rbf').fit(X, y)
```

[6]:
```
# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

x_min
```

[6]:
```
0.0
```

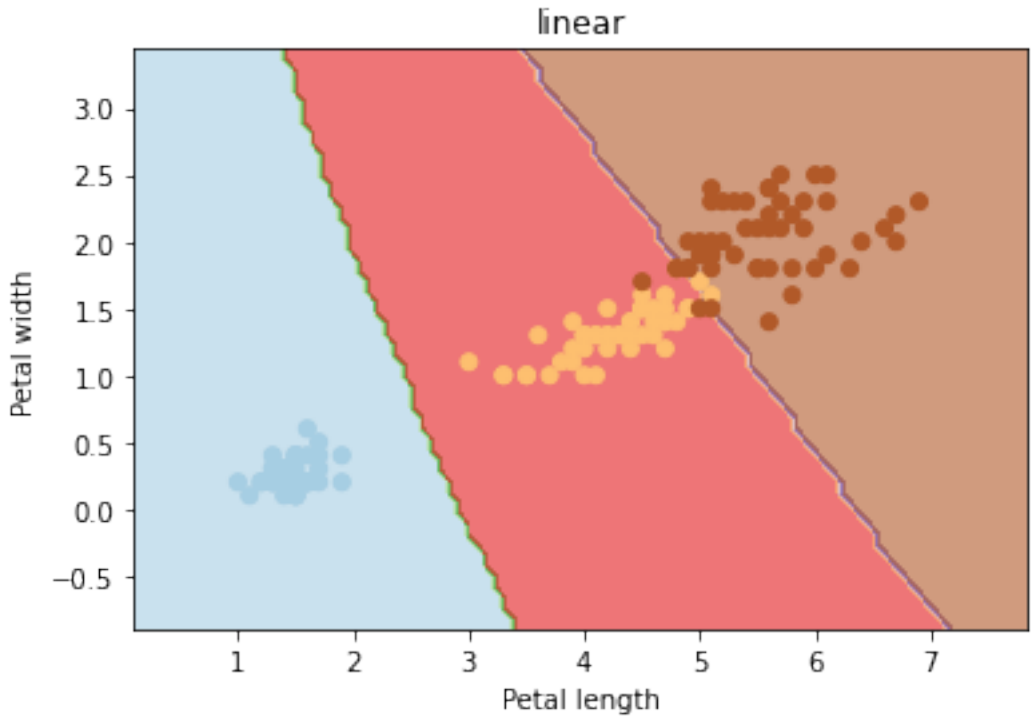
[7]:
```
# prevent division by zero
if x_min == 0.0:
```

```
    x_min = 0.1

x_min
```

[7]:
```
0.1
```

[8]:
```
h = (x_max / x_min)/1000
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
```

[9]:
```
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
#print(Z)

plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.6)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)

plt.xlabel('Petal length')
plt.ylabel('Petal width')
plt.xlim(xx.min(), xx.max())
plt.title('linear')
#plt.savefig('plot.png')
plt.show()
```



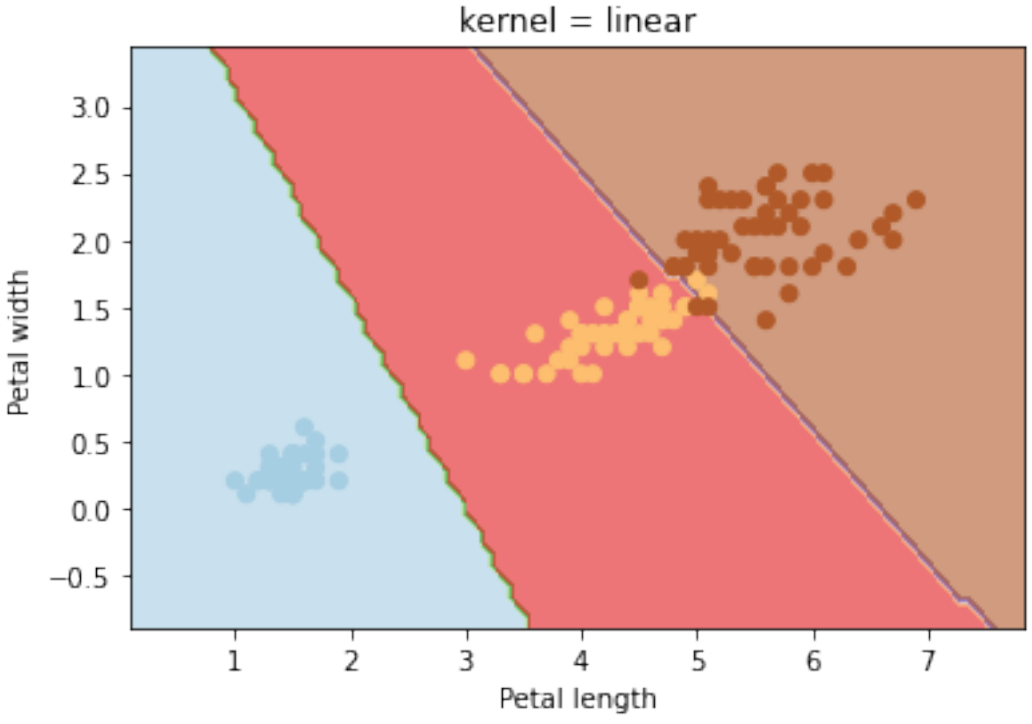
[10]:
```
def plotSVC(title, xlabel, ylabel):
    # create a mesh to plot in
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    # prevent division by zero
    if x_min == 0.0:
```

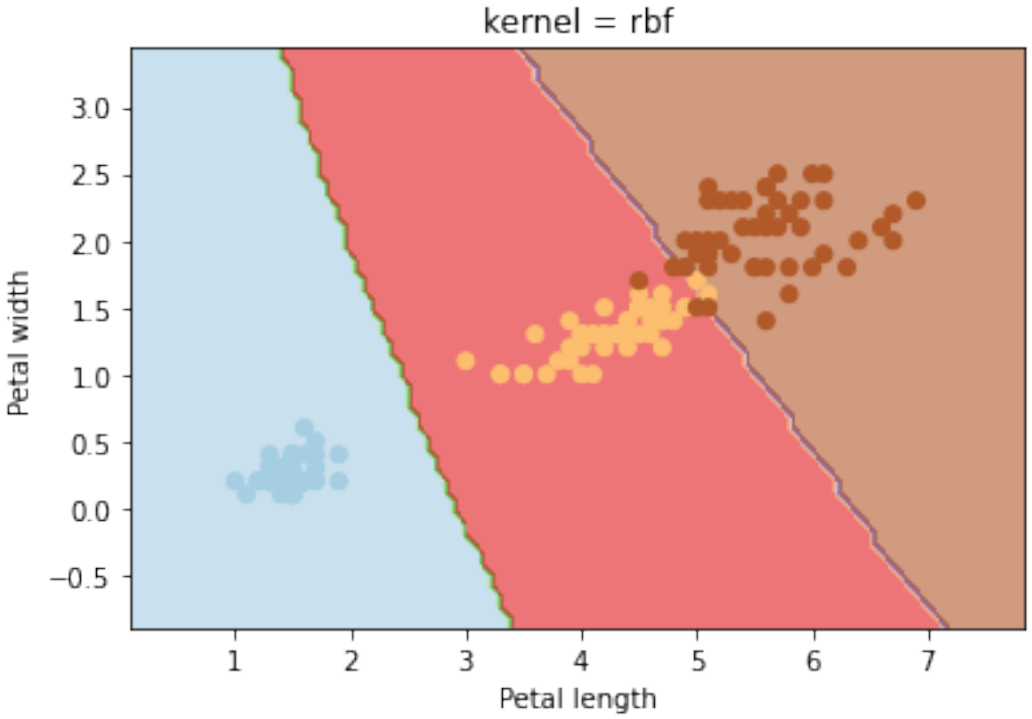
```
        x_min = 0.1

    h = (x_max / x_min)/1000
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    plt.subplot(1, 1, 1)
    Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.6)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.xlim(xx.min(), xx.max())
    plt.title(title)
    plt.show()
```

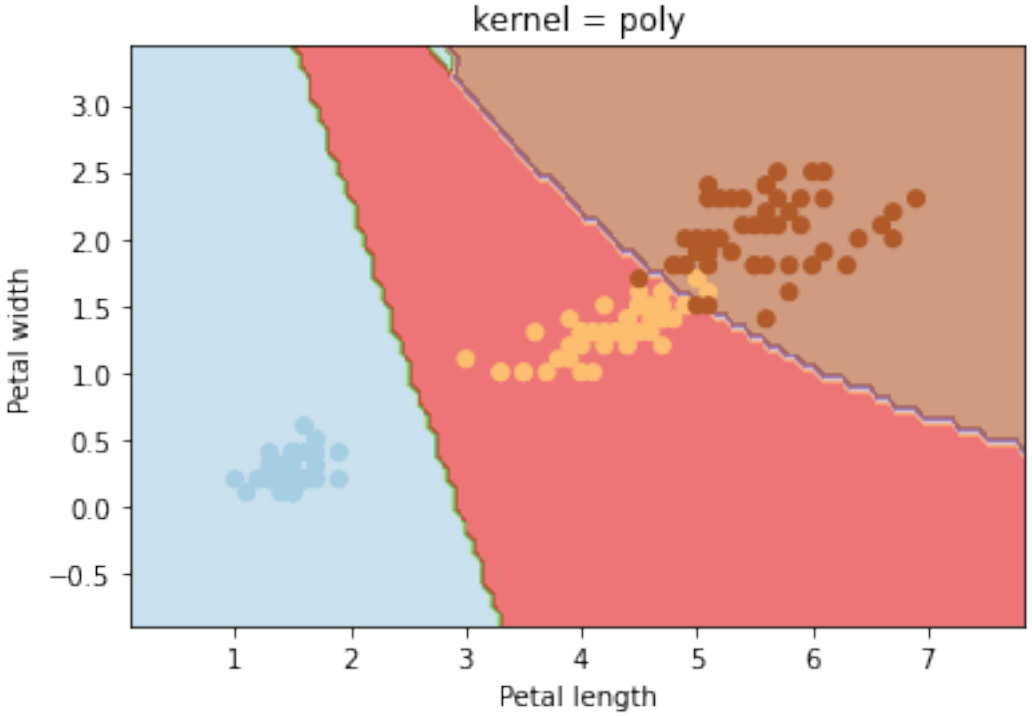
2 Vary `kernel`

The `kernel` parameter selects the type of hyperplane that is used to separate the data. Using `linear` (linear classifier) kernel will use a linear hyperplane (a line in the case of 2D data). The `rbf` (radial basis function kernel) and `poly` (polynomial kernel) kernel use non linear hyperplanes.

[11]:
```
kernels = ['linear', 'rbf', 'poly']

xlabel = 'Petal length'
ylabel = 'Petal width'

for kernel in kernels:
    svc = svm.SVC(kernel=kernel).fit(X, y)
    plotSVC('kernel = ' + str(kernel), xlabel, ylabel)
```




kernel = rbf
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7

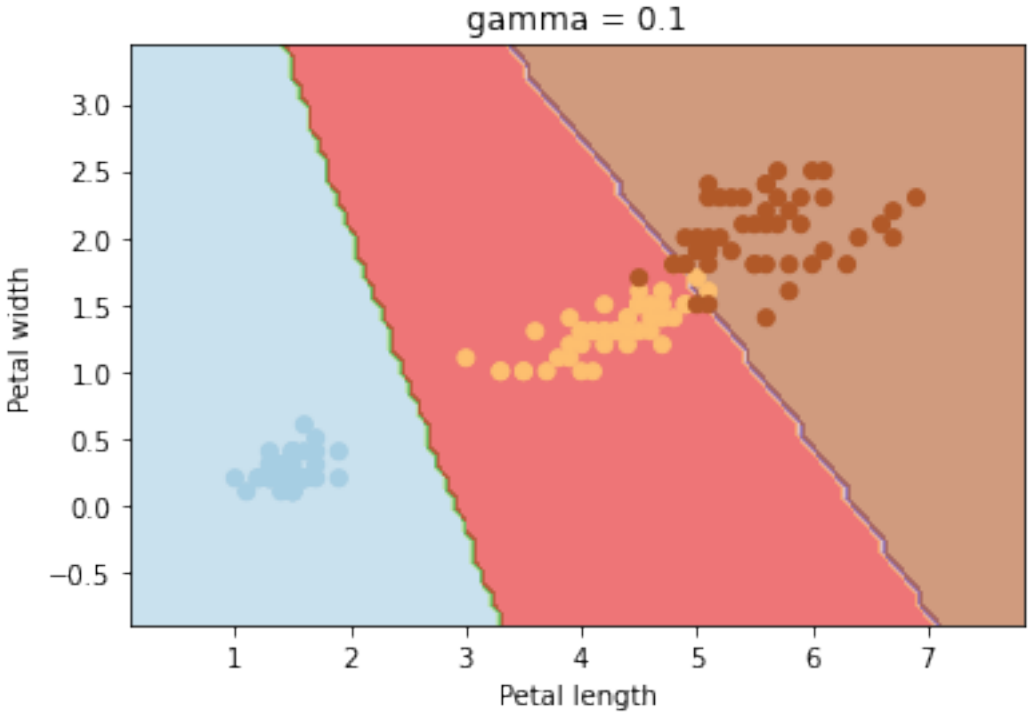

kernel = poly
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7

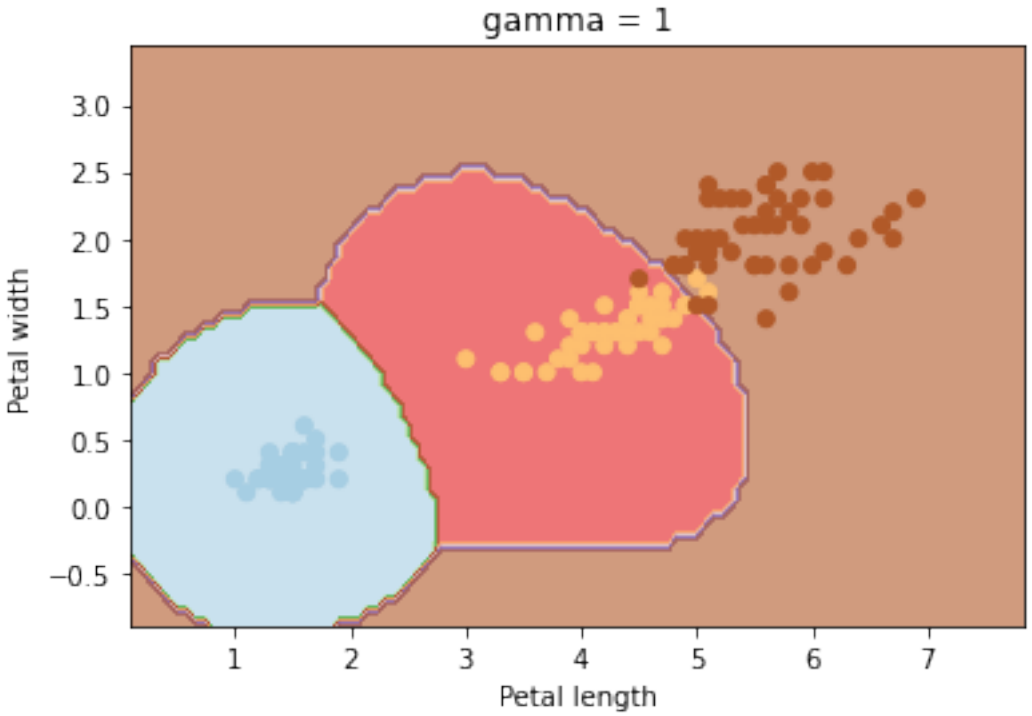
3 Vary gamma

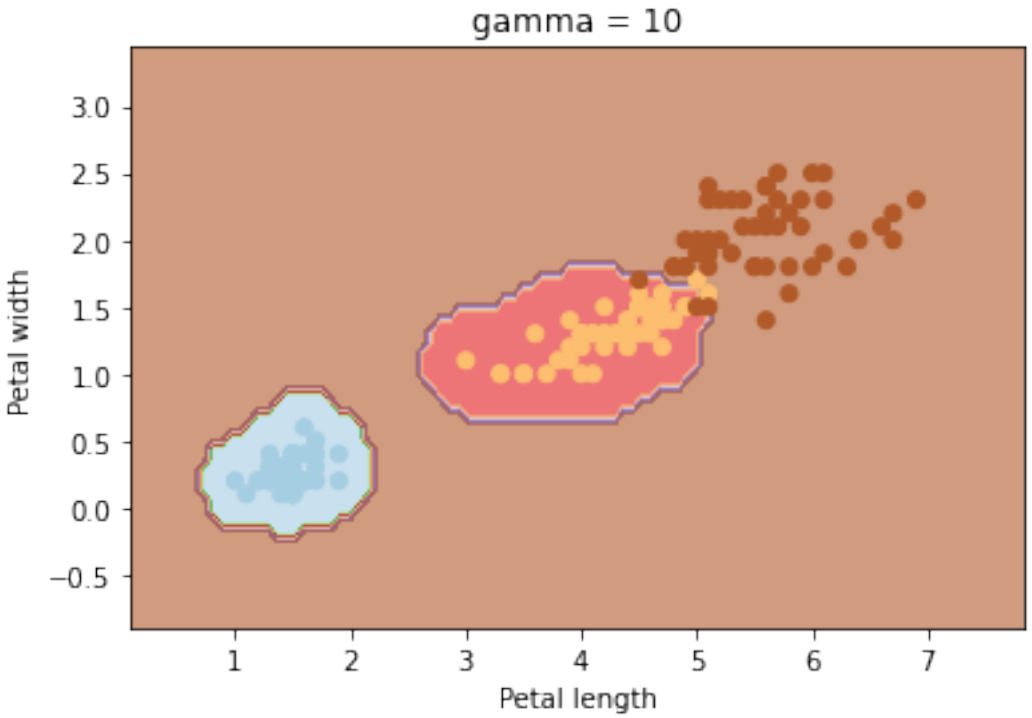
The gamma parameter is used for non linear hyperplanes. The higher the gamma value it tries to **exactly fit** the training data set.

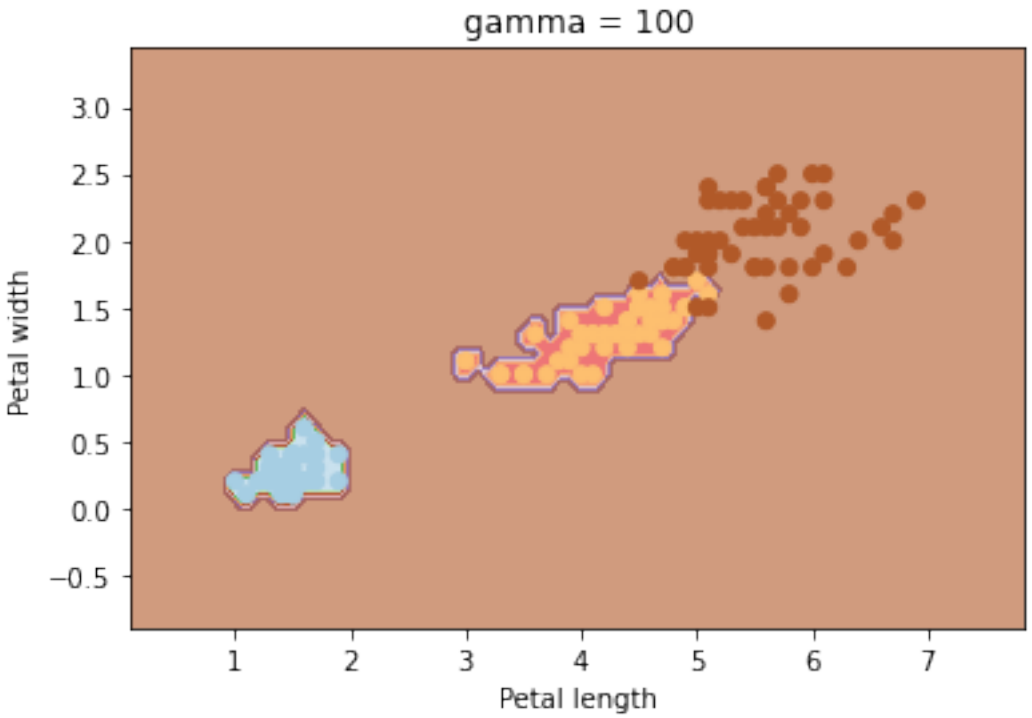
As we can see, increasing gamma leads to **overfitting** as the classifier tries to perfectly fit the training data.

[12]:
```
gammas = [0.1, 1, 10, 100]

xlabel = 'Petal length'
ylabel = 'Petal width'

for gamma in gammas:
    svc = svm.SVC(kernel='rbf', gamma=gamma).fit(X, y)
    plotSVC('gamma = ' + str(gamma), xlabel, ylabel)
```




gamma = 1
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7


gamma = 10
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7

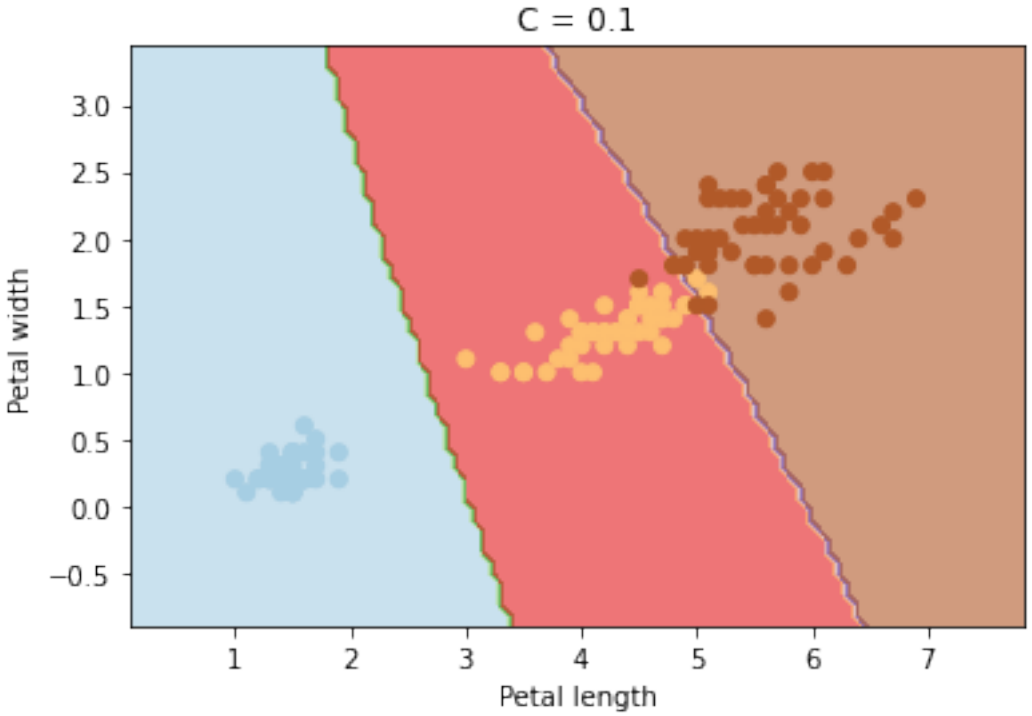


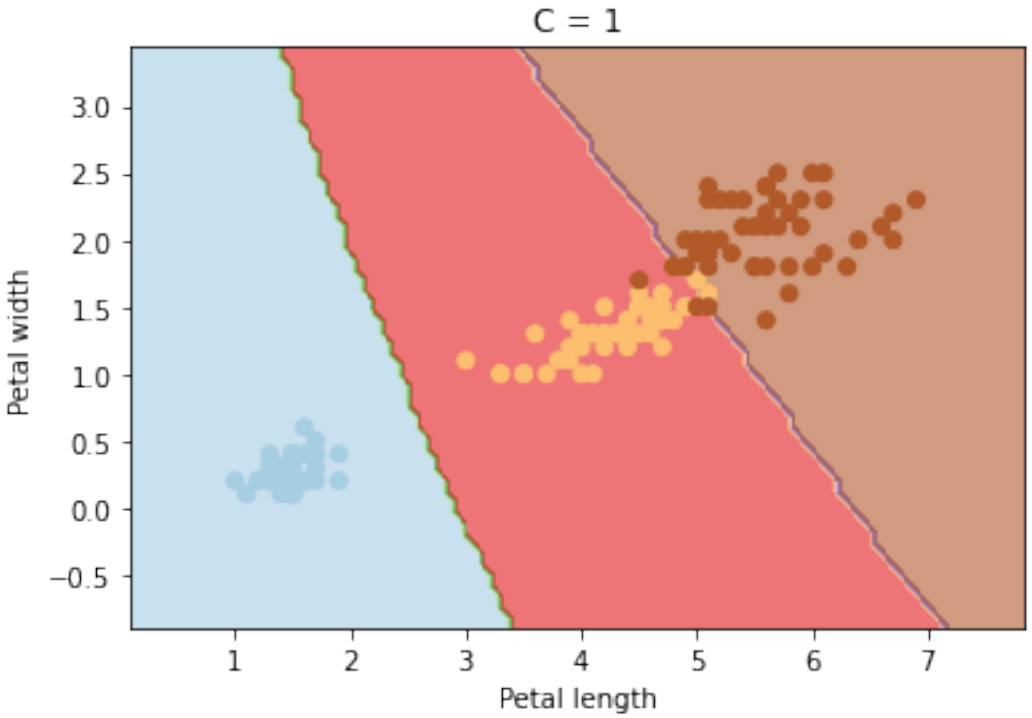
4 Vary C

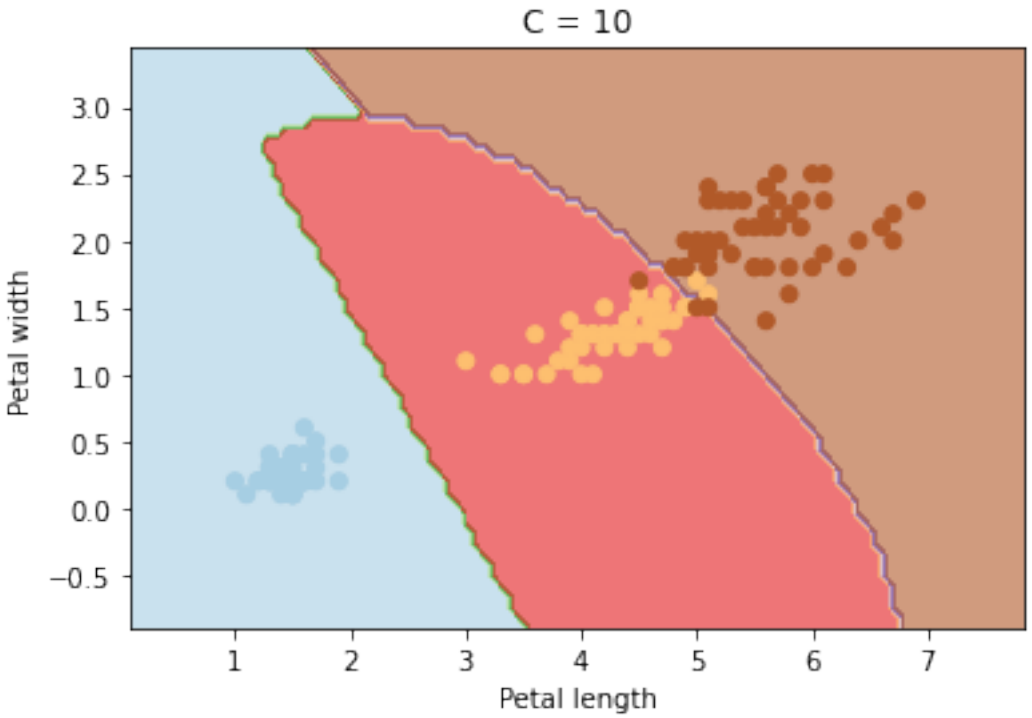
The C parameter is the **penalty** of the error term. It controls the trade off between smooth decision boundary and classifying the training points correctly.

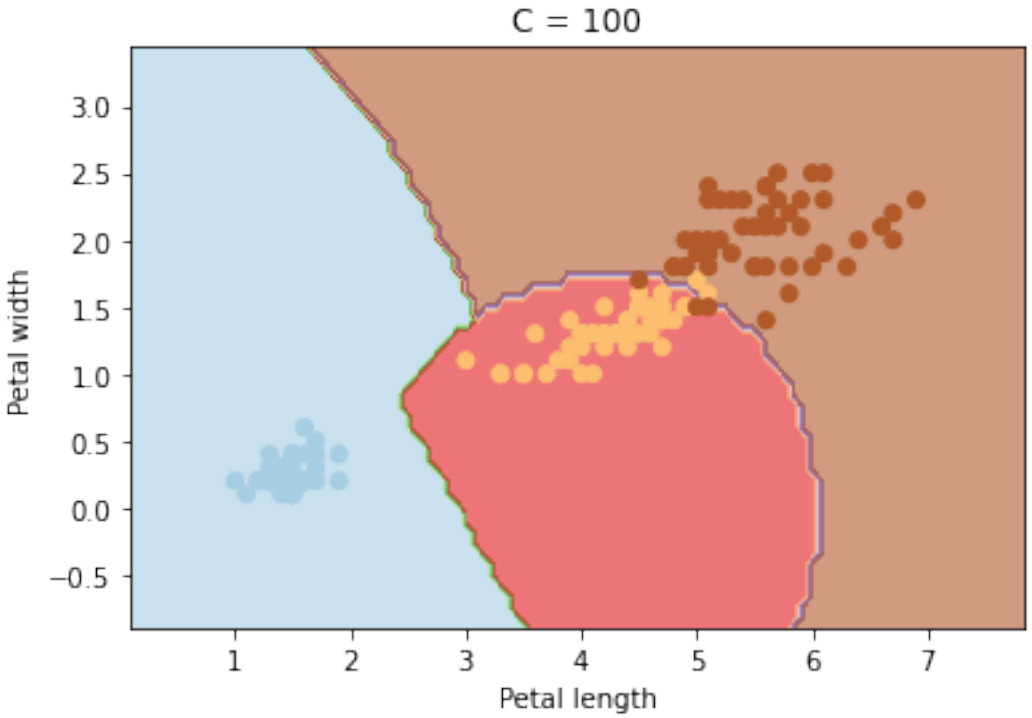
But be careful: to high C values may lead to **overfitting** the training data.

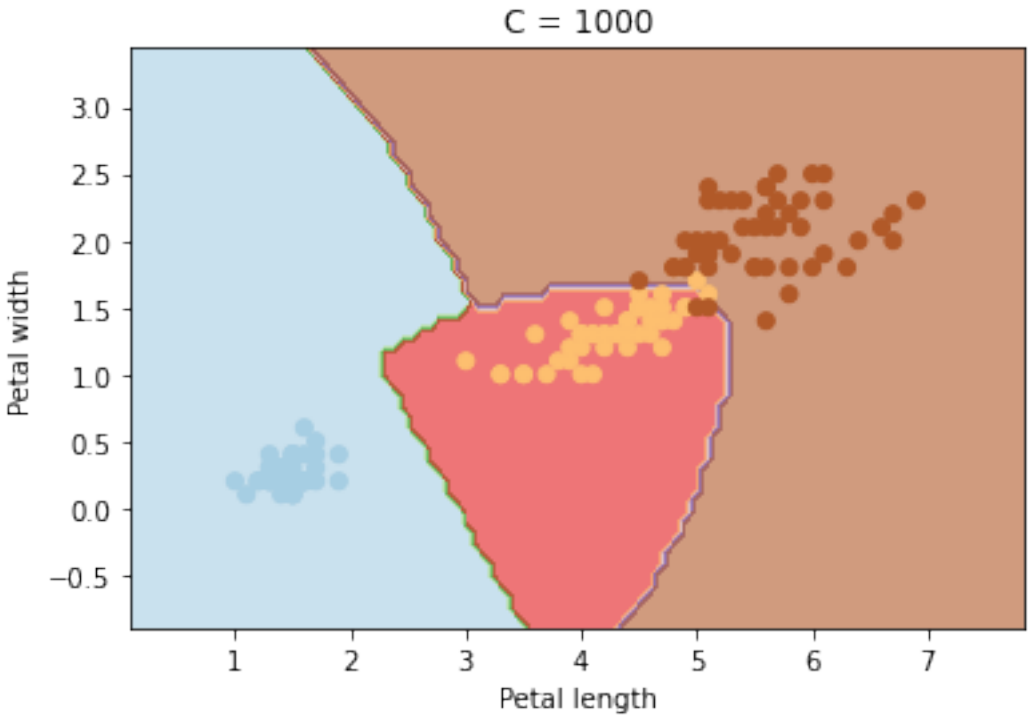
[13]:
```
cs = [0.1, 1, 10, 100, 1000]

xlabel = 'Petal length'
ylabel = 'Petal width'

for c in cs:
    svc = svm.SVC(kernel='rbf', C=c).fit(X, y)
    plotSVC('C = ' + str(c), xlabel, ylabel)
```


C = 0.1
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7


C = 1
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7


C = 10
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7


C = 100
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7

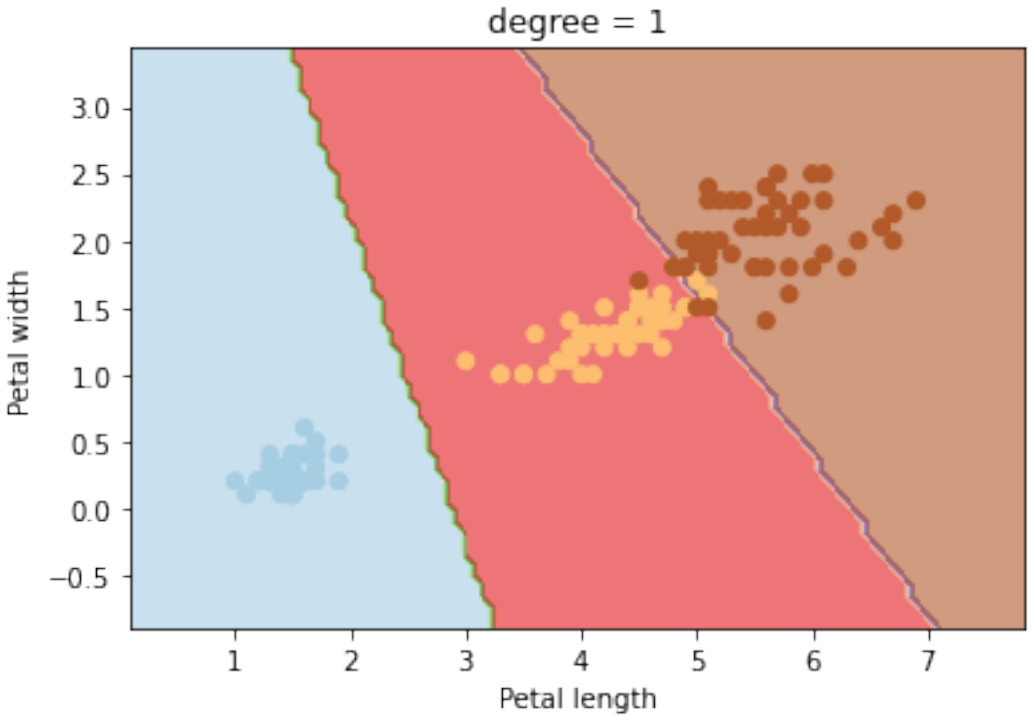


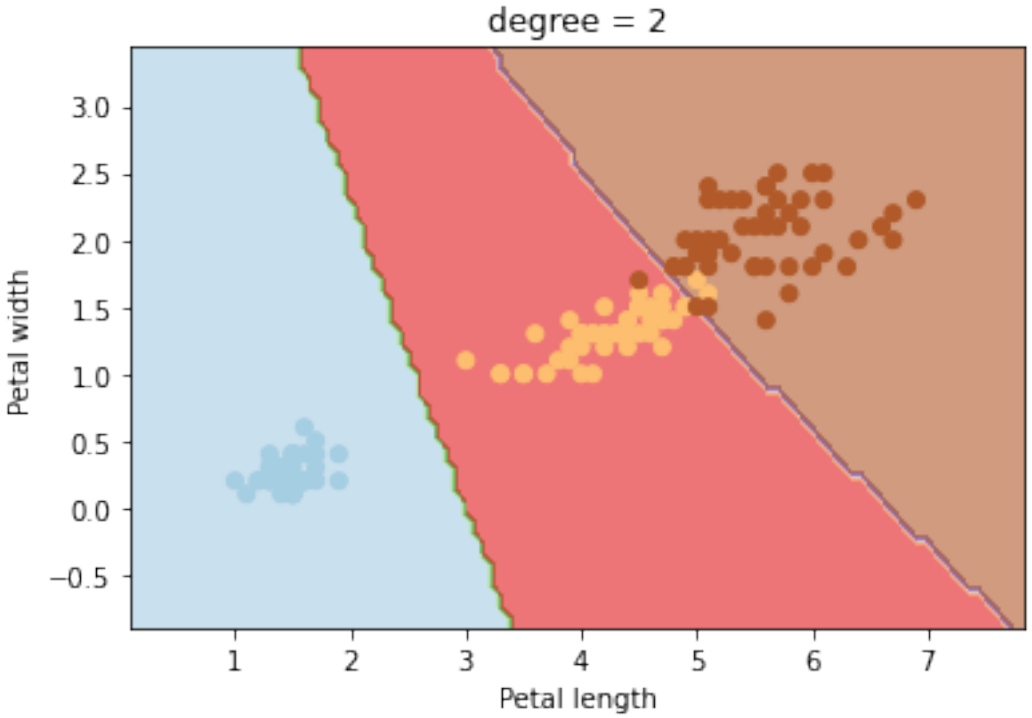
5 degree

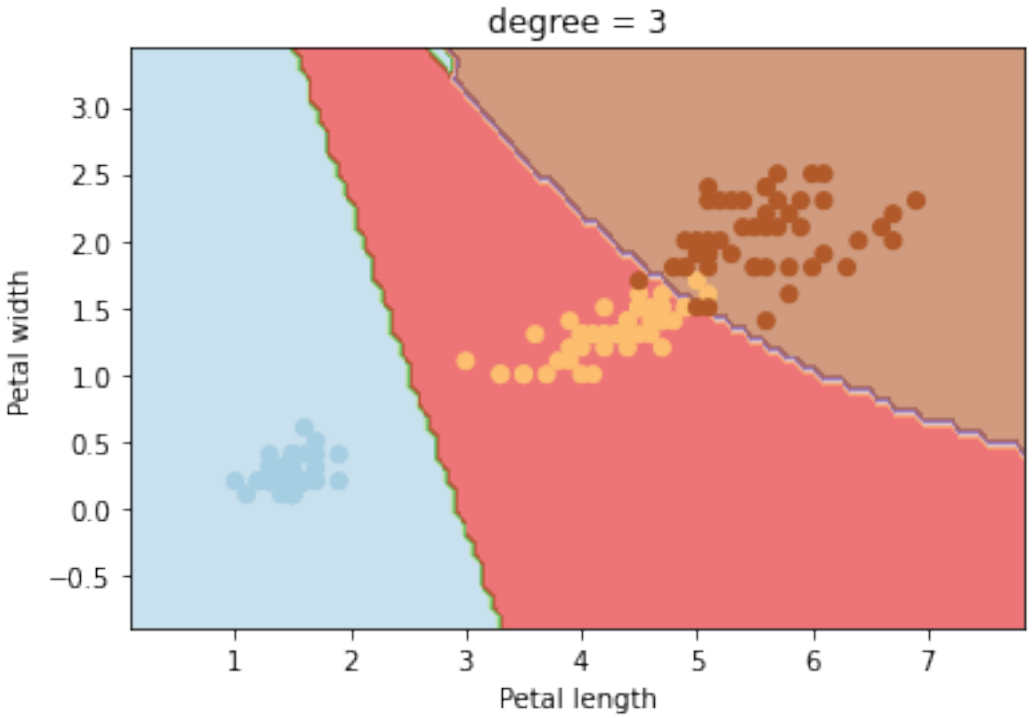
The `degree` parameter is used when the `kernel` is set to `poly`. It's basically the **degree of the polynomial** used to find the hyperplane to split the data.

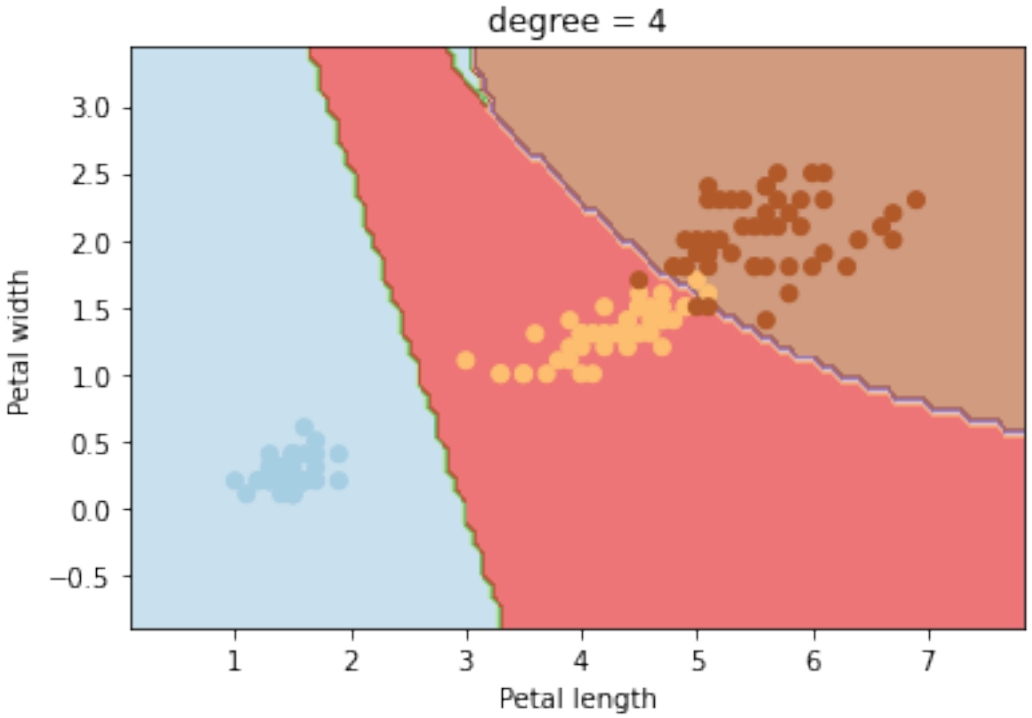
Using `degree = 1` is the same as using a `linear` kernel. Also, increasing this parameters leads to **higher training times**.

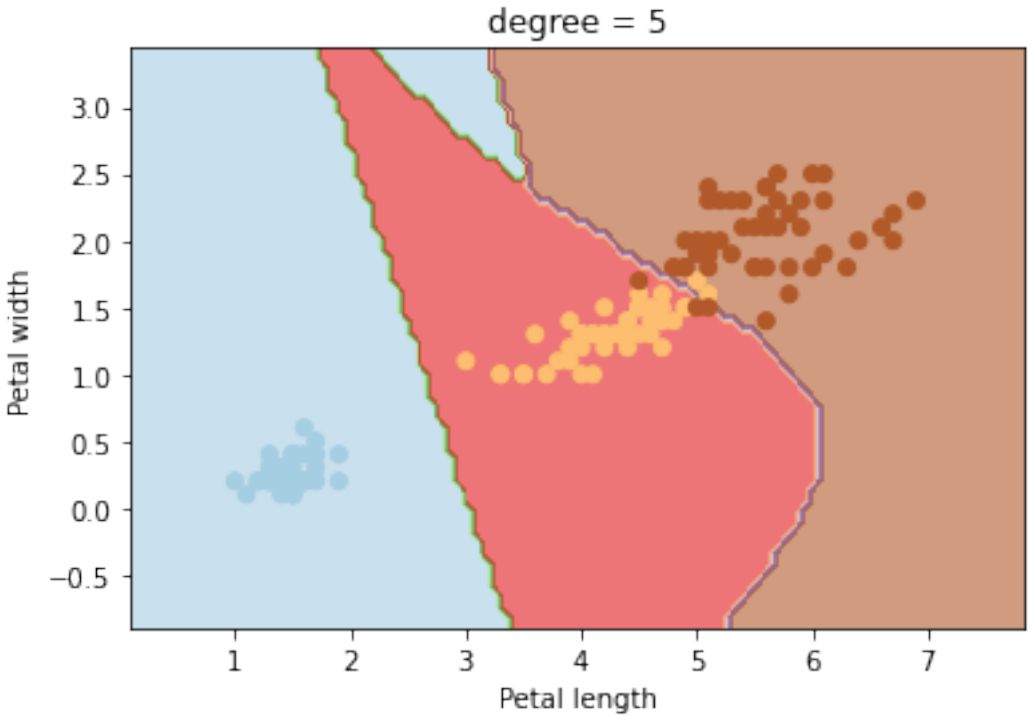
[14]:
```
degrees = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

xlabel = 'Petal length'
ylabel = 'Petal width'

for degree in degrees:
    svc = svm.SVC(kernel='poly', degree=degree).fit(X, y)
    plotSVC('degree = ' + str(degree), xlabel, ylabel)
```

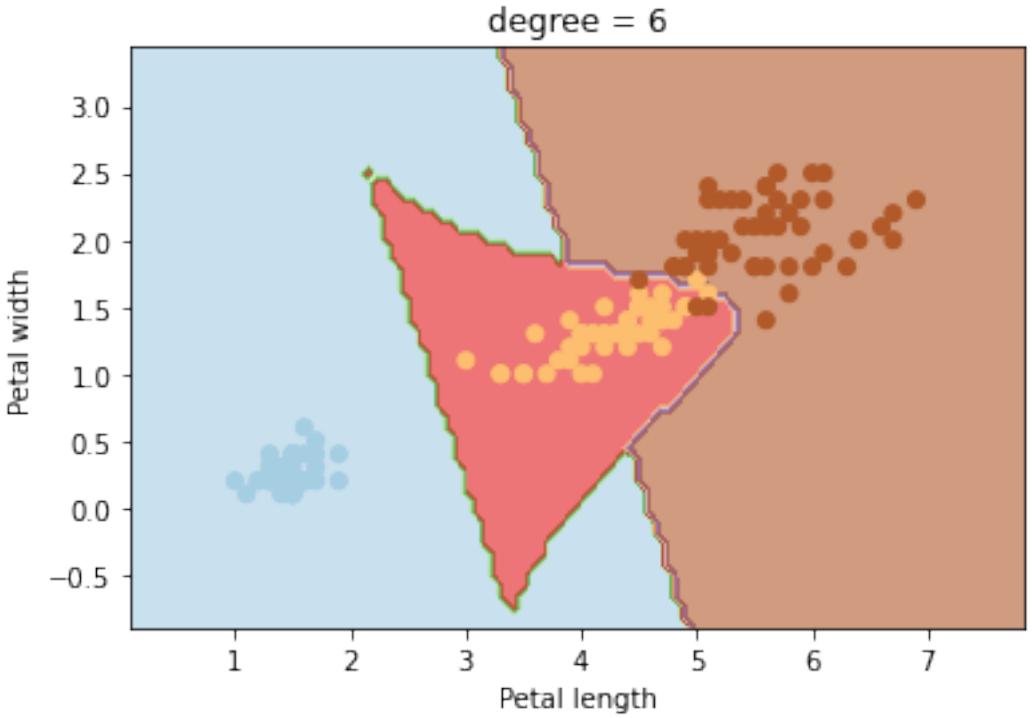

degree = 1
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7

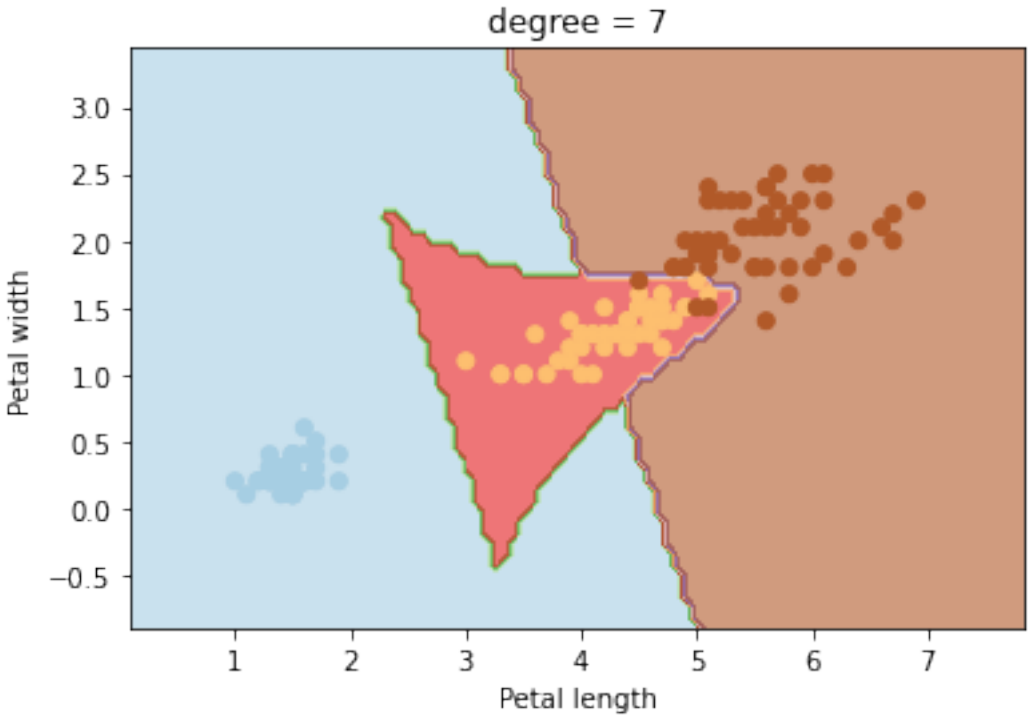

degree = 2
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7

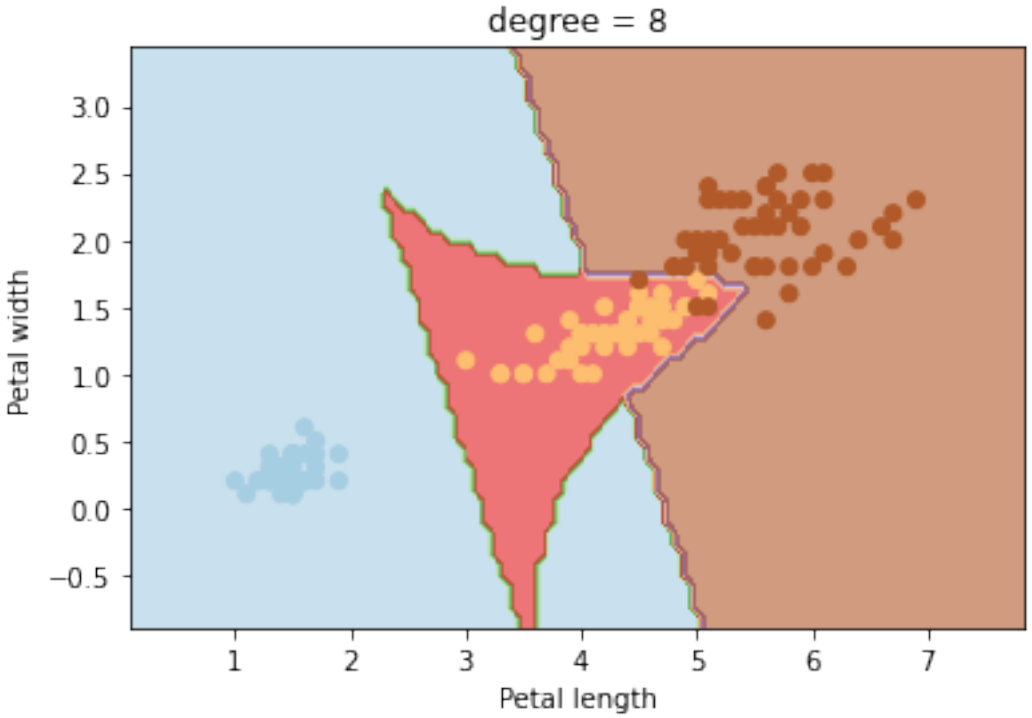

degree = 3
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7

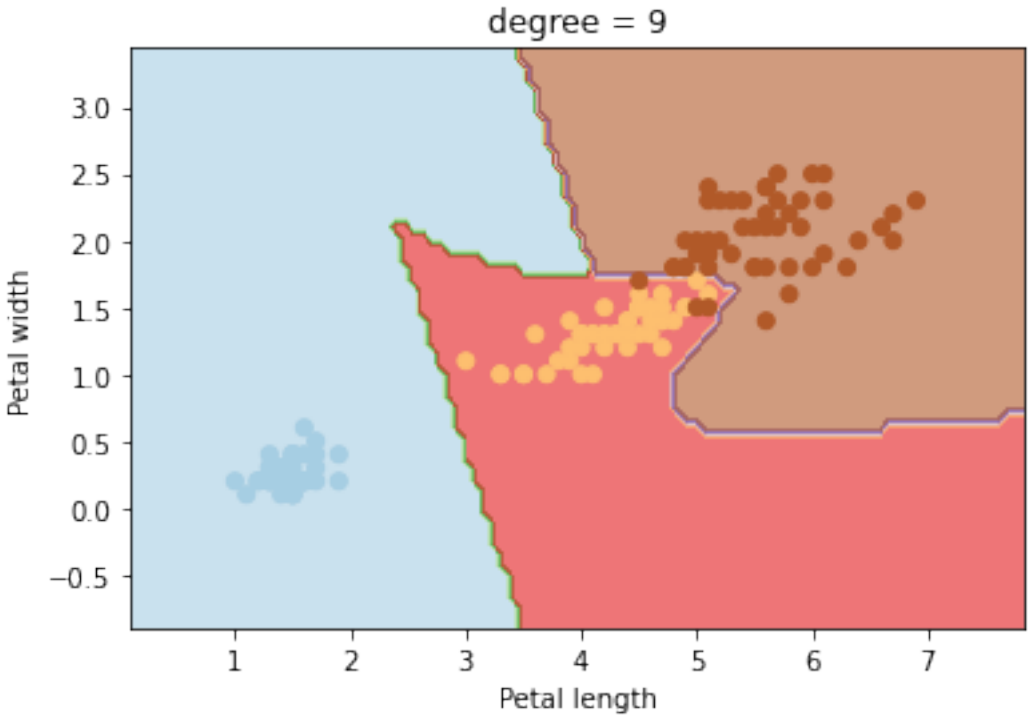

degree = 4
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7

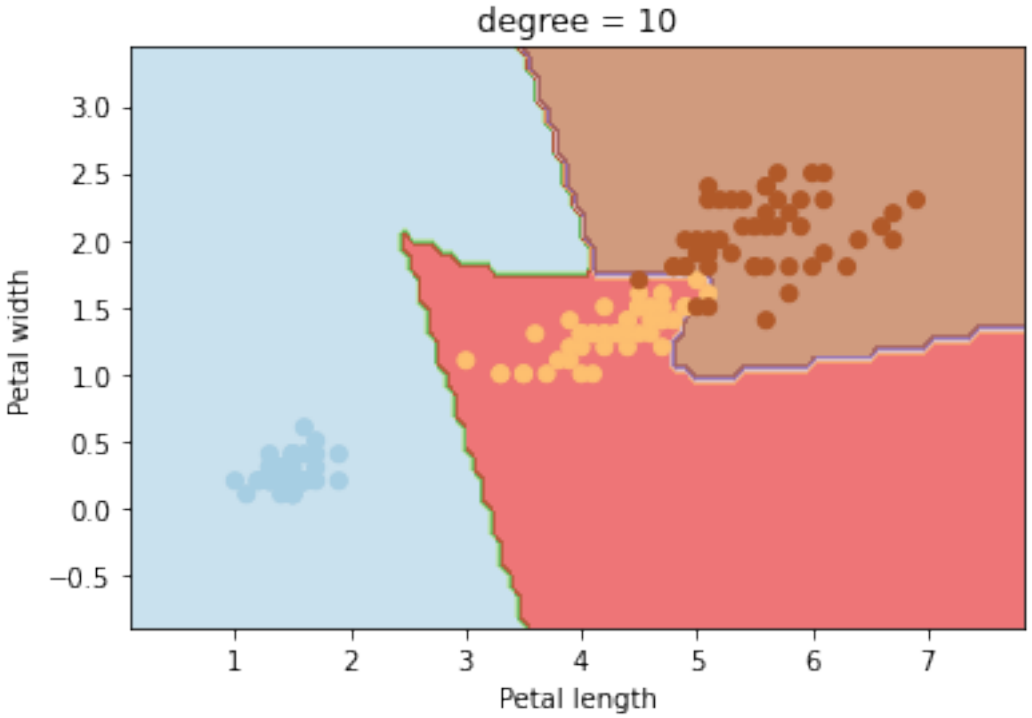

degree = 5
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7


degree = 6
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7


degree = 7
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7


degree = 8
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7


degree = 9
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7


degree = 10
Petal width
Petal length
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
1
2
3
4
5
6
7

[ ]: