ERHVERVSAKADEMI AARHUS

Machine Learning, March 10th, 2023.

Sila, March 2023.

SVM exercises.

In groups of 2-3.

Exercise 1.

What is the intuition behind a SVM?

Well, run the file SVM_1.py in order to (perhaps) get an intuition about what is actually going on, and how this might work behind the scenes. In the example we see a dataset where linear classification is impossible. Using a function to “lift” some of data to higher dimensions might change this.

Exercise 2.

Given a dataset, sometimes linear classification is impossible. In those cases it might be possible to use a Support Vector Machine instead. But notice that an SVM can work with linear separation as well.

Again we will look at the Iris data, which we’ll later use to classify. This time we will look at the first two features:

- sepal length
- sepal width

We have now tried to separate with a linear line a couple of times in the previous exercises. But what if the dataset cannot be separated by a linear line?

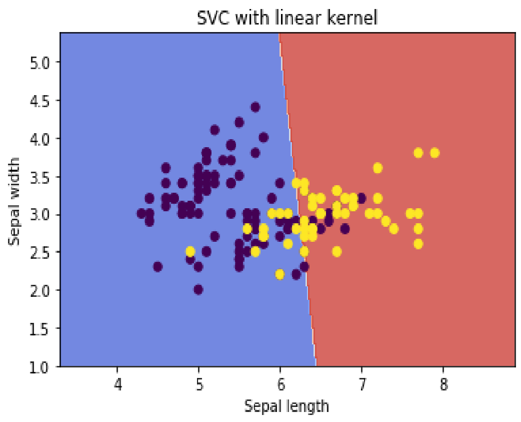
Well, then we can use a kernel, a function that we give to a machine learning algorithm (a kernel is not limited to an svm). There is some trickery involved here, but we can try out the mechanisms without going into all of the details of why it works, and see if this can help us make good contour maps.

First the code (See SVM_2.py, Canvas)

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn import svm, datasets
```

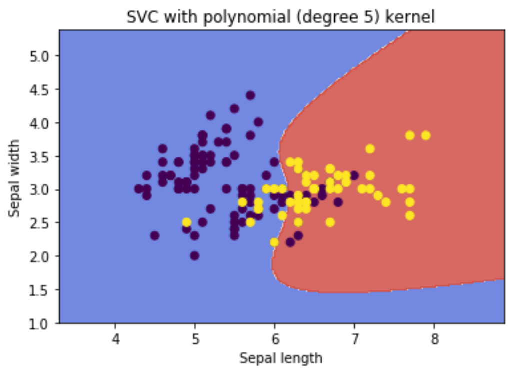
```
# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features. We could
                      # avoid this ugly slicing by using a two-dim dataset
yvalues = iris.target

y = []
for i in yvalues:
  if i == 2:
     y = np.append(y,[1])
  else:
     y = np.append(y,[0])

h = .02  # step size in the mesh

# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0  # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=C).fit(X, y)
#rbf_svc = svm.SVC(kernel='rbf', gamma=0.7, C=C).fit(X, y)
#poly_svc = svm.SVC(kernel='poly', degree=5, C=C).fit(X, y)
#lin_svc = svm.LinearSVC(C=C).fit(X, y)

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

plt.subplot(1, 1, 1)
plt.subplots_adjust(wspace=0.4, hspace=0.4)

Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title("SVC with linear kernel")
plt.show()
```

First, lets try with a linear kernel:

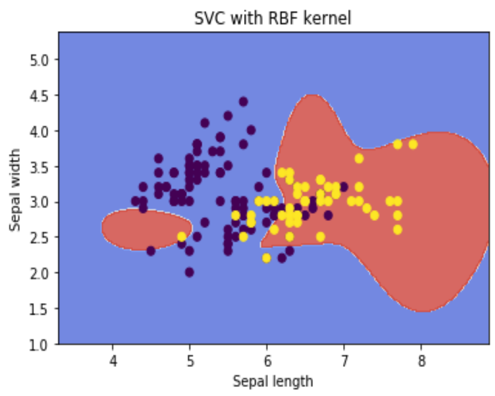


Maybe we can improve this by using a SVC with a polynomial (degree 3) kernel? Or a 5 degree kernel?

Try out various versions of this, (at least 1,3,5 and 7) just as you should try out various values for C (at least a small value, 1, and a big value for each degree of the polynomial). What seems to work the best?



We might also want to try out a RBF kernel.

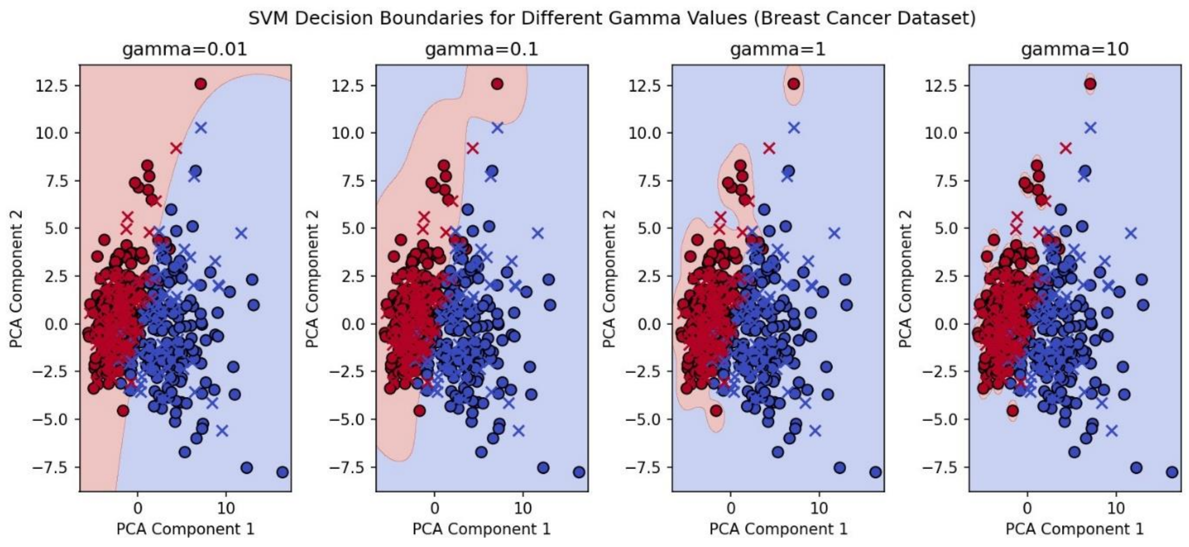


Here you should try out different gamma values. Higher gamma values means a more exact fit to the training set, with a risk of overfitting. You should try out values between 0.1 to say 100.

Exercise 3.

In this example we use the breast cancer dataset (See SVM_gamma.py, Canvas), and make plots for different values of gamma (using a support vector machine with a rbf kernel).

Notice the effects on the test set (here marked with x). Try with different values of C and gamma.


SVM Decision Boundaries for Different Gamma Values (Breast Cancer Dataset)
gamma=0.01
gamma=0.1
gamma=1
gamma=10
PCA Component 1
PCA Component 2