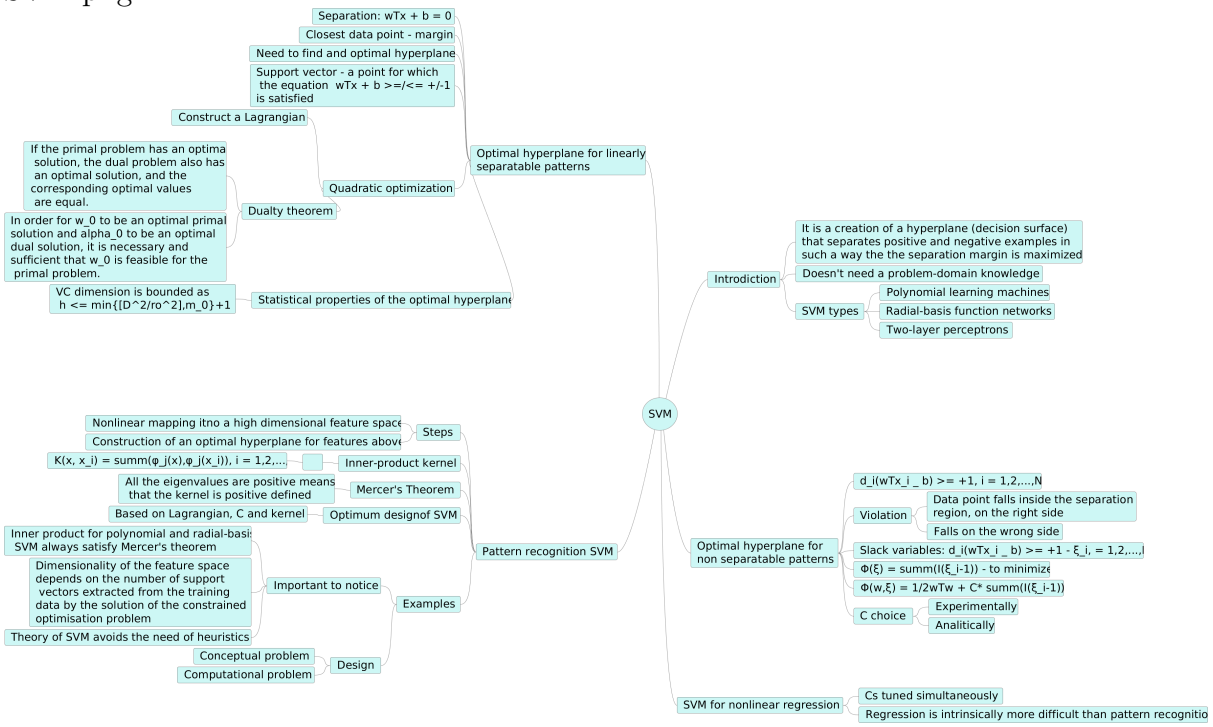
Neural Networks
- Homework 9 -

Petr Lukin, Evgeniya Ovchinnikova

Lecture date: 21 November 2016

1 Mind map

Figure 1: Mind map. Chapter 6 from Haykin's book. A zoomed version is attached as SVM.png



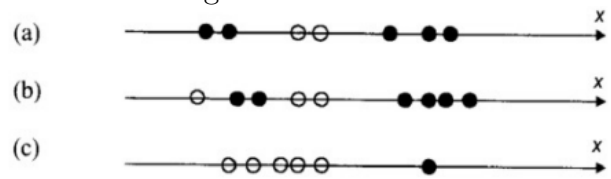
2 Exercises

2.1 Exercise 2

The graphs below represent three different one-dimensional classification (dichotomization) tasks (along a sketched x-axis, dash means "no data point"). What is the lowest-order polynomial decision function that can correctly classify the given data? Black dots d note class 1 with target function value y1 = +1 and white dots depict class 2 with targets y2 = -1. What are the decision boundaries?

If you wanted to classify the data sets (a), (b), (c) from Fig.2 using SVMs with Gaussian basis functions, how many hidden layer neurons would you need for each problem?

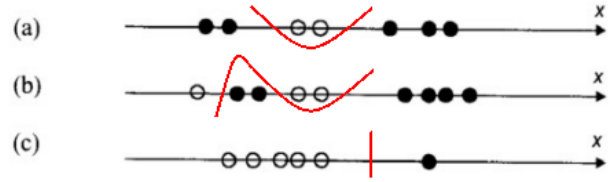
Figure 2: The dots



Solution:

From the positive and negative examples distribution we can assume that (a) requires a second order polynomial function, (b) - third degree and (c) - just a linear one (Fig. 3).

Figure 3: The dots with classification polynoms



To proof it we used the following code:

```
from itertools import product

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.svm import SVC
import sklearn.svm as svm

Xa = np.array([[1,0],[2,0],[5,0],[6,0],[9,0],[11,0],[12,0]])
ya = np.array([-1, -1, 1, 1, -1, -1,-1])
Xb = np.array
    ([[1,0],[3,0],[4,0],[6,0],[7,0],[10,0],[11,0],[12,0],[14,0]])
yb = np.array([1, -1, -1, 1, 1, -1, -1, -1, -1])
Xc = np.array([[1,0],[2,0],[3,0],[4,0],[5,0],[9,0]])
yc = np.array([1,1,1,1,1,-1])

def classify_and_plot(X,Y, degr):
    clf = svm.SVC(kernel='poly', coef0 = 1, probability=True,
        degree = degr, gamma=2)
    clf.fit(X, Y)
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    h = 0.1
    x, y = np.meshgrid(np.arange(x_min, x_max, h), np.arange(
        y_min, y_max, h))
    #print np.c_[xx.ravel(), yy.ravel()]
```

```
    Z = clf.predict(np.c_[x.ravel(), y.ravel()])
    Z = Z.reshape(x.shape)
    plt.pcolormesh(x, y, Z, cmap=plt.cm.Paired)
    plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.Paired)
    plt.title('SVM classification')
    plt.axis('tight')
    plt.grid()
    plt.show()
```

First, we will try to classify all the cases with degree = 1:

```
classify_and_plot(Xa, ya, 1)
classify_and_plot(Xb, yb, 1)
classify_and_plot(Xc, yc, 1)
```

The results are shown in Fig. 4. We can see that the classification for (c) case is correct, so a linear function is fine for this case. Classifications for (a) and (b) are wrong.

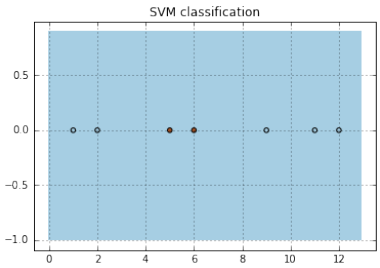


Figure 4: Classification for (a) case with degree = 1

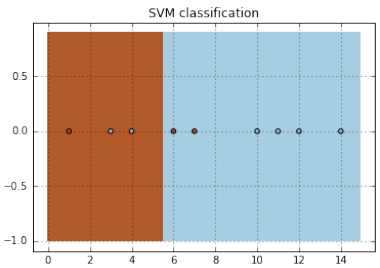


Figure 5: lassification for (b) case with degree = 1

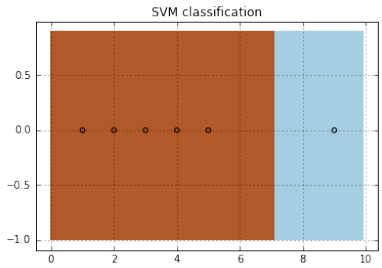


Figure 6: Alassification for (c) case with degree = 1

So, we will test (a) and (b) with degree = 2:

```
classify_and_plot(Xa, ya, 2)
classify_and_plot(Xb, yb, 2)
```

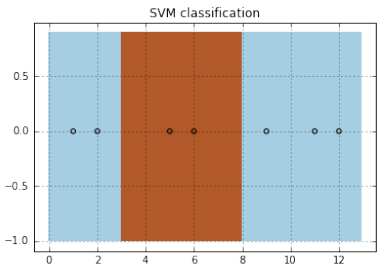


Figure 7: Classification for (a) case with degree = 2

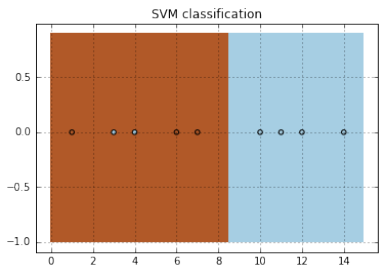
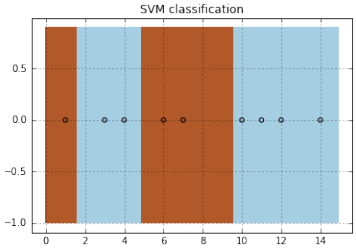


Figure 8: lassification for (b) case with degree = 2

In Fig. 7 we see that, as expected, classification for (a) is good, but classification for (b) still failing. So, we need to try degree = 3:

```
classify_and_plot(Xb, yb, 3)
```

Figure 9: Classification for (b) case with degree = 3



If we wanted to classify the data sets (a), (b), (c) using SVMs with Gaussian basis functions, we to use another - 'rbf' - kernel function:

```
clf = svm.SVC(kernel='rbf', coef0 = 1, probability=True, degree
    = degr, gamma=2)
```

We will obtain the following results:

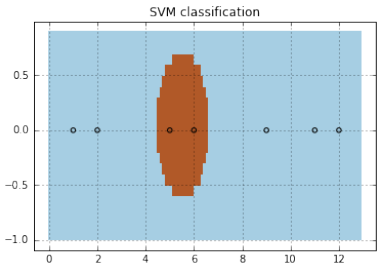


Figure 10: Classification for (a) case with Gaussian basis function

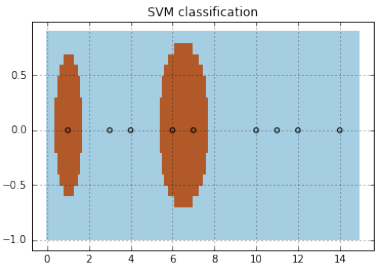


Figure 11: lassification for (b) case with Gaussian basis function

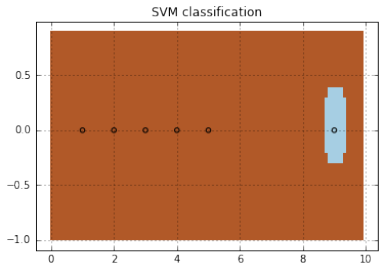


Figure 12: Alassification for (c) case with Gaussian basis function

To determine the number of neurons we need a number of features in a feature space that depends on the number of support vectors extracted from the training data. From Fig. 10 we can see that in (a) case we have 4 support vectors, in (b) - 6 and in (c) - 2. So, we should have 4, 5 and 2 neurons.