BRSU

Neural Networks
Assignment 4

Bastian Lang

October 31, 2015

1 OUTLINE

1.1 STATISTICAL NATURE OF THE LEARNING PROCESS

- Bias/Variance Dilemma

1.2 STATISTICAL LEARNING THEORY

- Some Basic Definitions
  - Convergence in probability
  - Supremum and infimum
  - Empirical risct functional
  - Strict Consistency
- Principle of Empirical Risk Minimization
- VC Dimension
- Example 2.1
- Example 2.2
- Example 2.3
- Importance of the VC dimension and its Estimation
  - VC dimension of an arbitrary feedforward network using heaviside functions is O(W log W) where W is the total number of free parameters in the network.
  - The VC dimension of a MLP using sigmoid functions is $O(W^2)$, where W is the number of free parameters.
- Constructive Distribution-free Bounds on the Generalization Ability of Learning Machines
- Structural Risk Minimization

1.3 PROBABLY APPROXIMATELY CORRECT MODEL OF LEARNING

- Sample Complexity
- Computational Complexity

2 Consider the space of instances X corresponding to all points in the x, y plane. Give the VC dimension of the following hypothesis spaces:

2.1 Hr = the set of all rectangles in the x,y plane. i.e. Hr = $\{((a < x < b) \wedge (c < y < d)) | a, b, c, d \in IR\}$

4

2.2 Hc = the set of all circles in the x,y plane. Points inside the circle are classified as positive examples.

3

2.3 Ht = the set of all triangle in the x,y plane. Points inside the triangle are classified as positive examples.

7

3 Write a consistent learner for H_r from last Exercise (i.e. $H_r = \{((a < x < b) \wedge (c < y < d)) | a, b, c, d \in IR\}$). Generate a variety of target concept rectangles at random, corresponding to different rectangles in the plane. Generate random examples of each of these target concepts, based on a uniform distribution of instances within the rectangle from (0,0) to (100, 100). Plot the generalization error as a function of the number of training examples, m. On the same graph, plot the theoretical relationship between e and m, for d = .95. Does theory fit experiment?

I was not able to solve this exercise due to too many open questions. It took me about three hours to come to the understanding below:

Questions:

- Write a consistent learner...
  - How does a learner look like? Is it a function that outputs a hypothesis to a given input? Do we use any kind of ANN and use a learning technique?

- Generate a variety of target concept rectangles at random, corresponding to different rectangles in the plane.
  - How many? (I will choose some number, not that important)
- Generate random examples of each of these target concepts, based on a uniform distribution of instances within the rectangle from (0,0) to (100, 100).
  - By examples of a target concept you mean a number of points in the plain within this huge rectangle, classified according to the concept?
- Plot the generalization error as a function of the number of training examples, m.
  - Use a subset of all examples to train a model?
  - Use varying numbers of training examples for training until the learner is consistent?
- plot the theoretical relationship between e and m, for d = .95.
  - E is generalization error?
  - What is D? Confidence?
- Does theory fit experiment?
  - Which theory?!

I created a Python program that creates some (15) rectangles and for every rectangle 50 points. Those points get classified by the rectangles and both rectangles and points get plotted.

3.1 Python Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 31 13:33:38 2015

@author: bastian
"""

from random import *
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import pylab

class Rect:
    def __init__(self, x_low, y_low, x_high, y_high):
        self.x_low = x_low
        self.x_high = x_high
        self.y_low = y_low
```

```
        self.y_high = y_high

    def classify_points(self, points):
        result = []
        for point in points:
            if ((point[0] >= self.x_low) and (point[0] <= self.x_high) and (point[1] >=
                result.append(Point(point[0], point[1], 1))
            else:
                result.append(Point(point[0], point[1], 0))
        return result

class Point:
    def __init__(self, x, y, value):
        self.x = x
        self.y = y
        self.value = value

def create_random_point(lower_bound, upper_bound):
    return randint(lower_bound, upper_bound)


def create_random_rectangle():
    x = []
    y = []
    x.append(create_random_point(0, 100))
    x.append(create_random_point(0, 100))
    y.append(create_random_point(0, 100))
    y.append(create_random_point(0, 100))

    if x[0] < x[1]:
        x_low = x[0]
        x_high = x[1]
    else:
        x_low = x[1]
        x_high = x[0]
    if y[0] < y[1]:
        y_low = y[0]
        y_high = y[1]
    else:
        y_low = y[1]
        y_high = y[0]

    return Rect(x_low, y_low, x_high, y_high)
```

```
def create_uniformly_distributed_points(number_of_points):
    result = []
    for i in range(number_of_points):
        result.append((create_random_point(0,100), create_random_point(0,100)))
    return result

def create_rectangles(number_of_rectangles):
    result = []
    for i in range(number_of_rectangles):
        result.append(create_random_rectangle())
    return result

def add_rectangle_to_plot(axis, rectangle):
    axis.add_patch(
        patches.Rectangle(
            ((rectangle.x_high - rectangle.x_low) / 2, (rectangle.y_high - rectangle.y_l
            rectangle.x_high - rectangle.x_low,
            rectangle.y_high - rectangle.y_low,
            alpha = 0.1
            )
        )

def classify_random_points_with_random_rectangles(rectangles, number_of_points_per_recta
    classified_points = []
    for rectangle in rectangles:
        points = create_uniformly_distributed_points(number_of_points_per_rectangle)
        classified_points.extend(rectangle.classify_points(points))
    return classified_points


rectangles = create_rectangles(15)
classified_points = classify_random_points_with_random_rectangles(rectangles, 50)
fig1 = plt.figure()

ax1 = fig1.add_subplot(111, aspect='equal')

for rectangle in rectangles:
    add_rectangle_to_plot(ax1, rectangle)
#print classified_points
for point in classified_points:
    #print point
    if point.value == 1:
        ax1.plot(point.x, point.y, 'r+')
    else:
```

```
        ax1.plot(point.x, point.y, 'b.')

pylab.ylim([0,100])
pylab.xlim([0,100])

pylab.savefig('ex3.png')
```

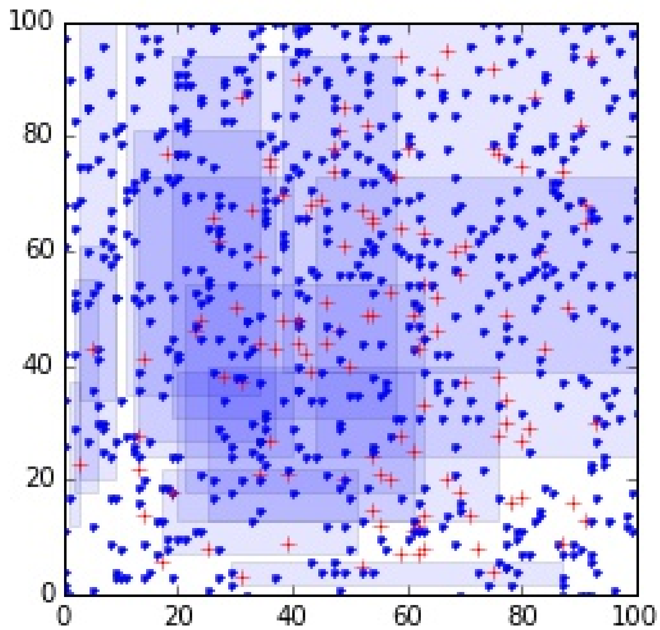

Figure 3.1: Output

4 Rephrase the Ex2.21 of chapter 2 from Haykin's book in such way that the next students will understand it better.