CS 6316 – *Machine Learning*

Homework 1: Review of Basic Math/Probability – ***Solutions***

Due date: September 1, 2017

OBJECTIVE:

- Reviewing background on general math/probability and programming

QUESTIONS:

Problems are from the textbook ("***Predictive Learning***") Chapter 1 – a PDF version of the chapter 1 is available for your convenience on Collab Resources (if you don't have the textbook yet).

Note: Future chapters of this textbook cannot be made available in PDF format, so I encourage you to purchase the textbook.

		Points
1.	Problem 1.2 *	25
2.	Problem 1.3	10
3.	Problem 1.4	15
4.	Problem 1.7 **	30
5.	Problem 1.13	20

* An additional reference is numpy.polyfit if you prefer to program in Python [https://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html]

** For Problem 1.7, use the adjusted closing price of SP500 (symbol ^GSPC).

SOLUTIONS:

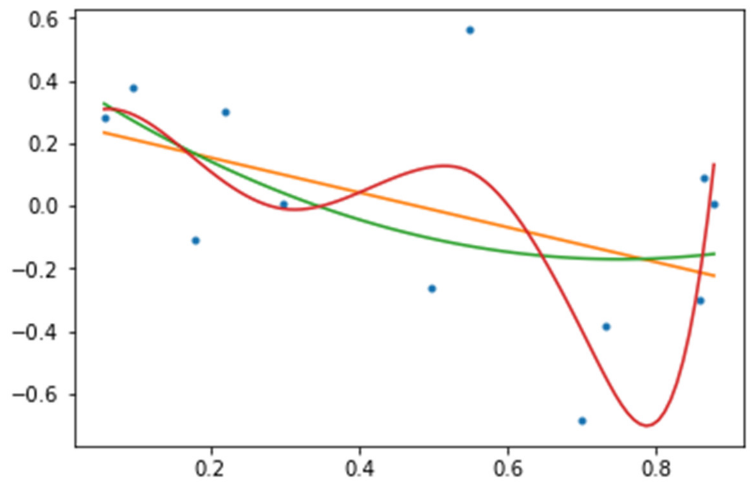
Problem 1.2 (25 pts)

We generate 12 data samples (x,y). The x-coordinate is uniformly distributed in [0, 1]. The y-coordinate has Gaussian distribution with zero mean and variance 0.5, N(0, 0.5).
The polynomial fitting models are shown below along with the data samples. The mean-squared error (MSE) for the linear, quadratic, and degree 6 models are 0.2564, 0.2325, and 0.1513, respectively.

A polynomial of degree 6 gives the best (smallest) fitting error. However, it would *not* give the best prediction accuracy for future data as the data samples are apparently not generated from a model of polynomial with degree 6.

Sample code:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import spline

x = np.random.uniform(0,1,12)
y = np.random.normal(0,0.5,12)

# fit with np.polyfit
a = np.polyfit(x, y, 1)
b = np.polyfit(x, y, 2)
c = np.polyfit(x, y, 6)

f1 = np.poly1d(a)
f2 = np.poly1d(b)
f3 = np.poly1d(c)

plt.plot(x, y, '.')
np.ndarray.sort(x)

# plot without smooth
# plt.plot(x, f1(x))
# plt.plot(x, f2(x))
# plt.plot(x, f3(x))

#  plot with smooth
x_smooth = np.linspace(x.min(), x.max(), 100)
f2_smooth = spline(x, f2(x), x_smooth)
f3_smooth = spline(x, f3(x), x_smooth)
plt.plot(x, f1(x))
plt.plot(x_smooth, f2_smooth)
plt.plot(x_smooth, f3_smooth)

plt.show()
```



Example output (see above) - *each run will give you a different output*

Problem 1.3 (10 pts)

(a) Note: assume there is unlimited supply of beer of each kind.
Each guest can be served 10 different types of beer. So for 2 guests, the number of different combinations is 10*10, for 3 guests 10*10*10 etc. Proceeding in the same manner, the number of different combinations for 10 guests is $\mathbf{10^{10}}$.
(b) Assume that guests show up at a party one at a time. Then the first guests can choose from 10 different bottles, the second quest can choose from 9 remaining bottles etc. So the total number of different combinations is 10*9*8*….*2*1 = **10!**

Problem 1.4 (15 pts)

The probability of receiving ***no*** response to a *single* spam message is 1 – 0.004 = 0.996. So the probability of receiving a favorable response (at least one sale) to 250 spam messages is $\mathbf{1-0.996^{250}}$

<u>Breaking this out further</u>:

1 - 0.004 = 0.996 = the chance of NOT getting a positive response to 1 spam message sent.
The Probability of NOT getting a positive response to 250 spam messages is:
0.996 * 0.996 * 0.996 … do this 250 times in total. This is equivalent to 0.996^{250} (multiplication rule of mutually exclusive events, repeated many times)

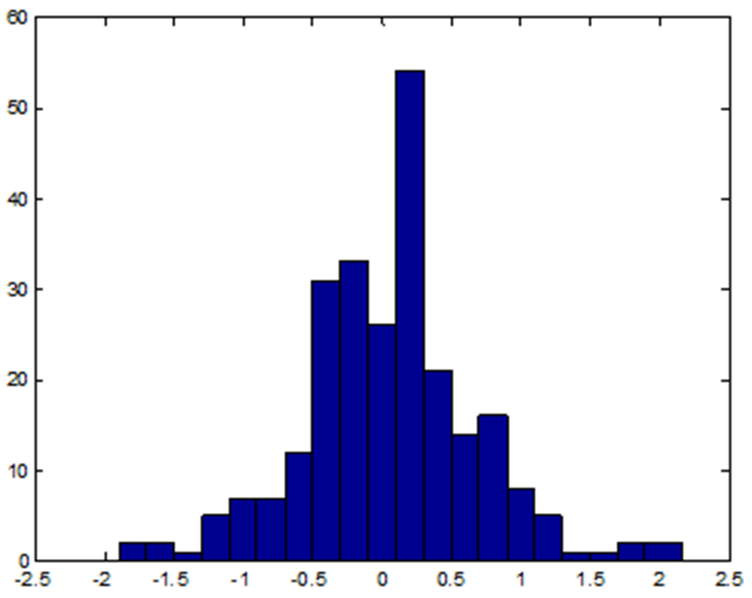
So now to get the chance of receiving a POSITIVE response to 250 spam messages, take 1 (total chance) and subtract the chance of not receiving a positive response (0.996^{250})

The result is: <u>$\mathbf{1-0.996^{250} = 0.63}$</u>

Problem 1.7 (30 pts)

Histograms for daily % change of SP500 (X) and its 5-day moving average (Y), for year 2006:

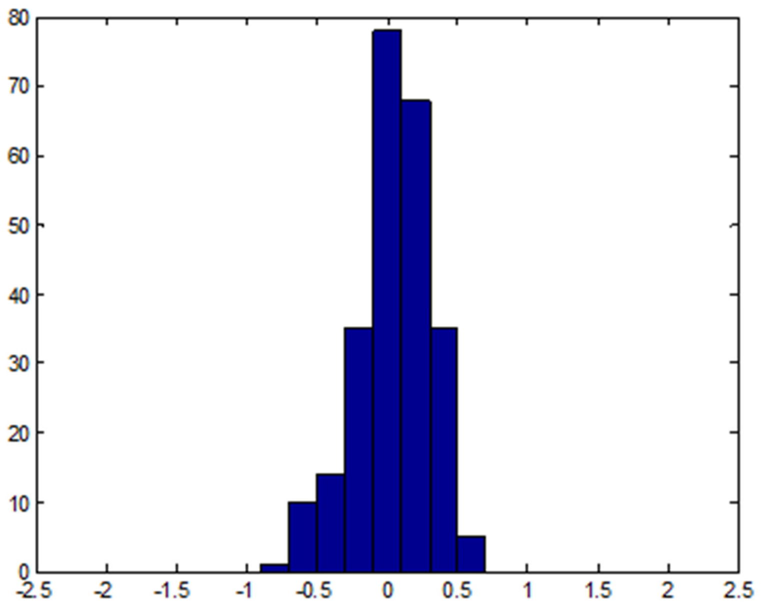
(a) Empirical distribution of observed X-values.



The estimated mean and standard deviation:

(Near the values of:) mean = 0.047 %;   standard deviation = 0.623 %

b)Empirical distribution of observed Y-values.



The estimated mean and standard deviation:

(Near the values of:) mean = 0.043 %;   standard deviation = 0.256 %

Example code:

```
import csv
import numpy as np
from math import *
from scipy.stats import norm
import matplotlib.pyplot as plt

def print_x_values():
        adjusted_closing_prices = []
        x_values = []       # all daily percentage price
change
        x_processed_values = []     #x values between -
2 to 2 percent
        num_of_acp = 0    # Total number of adjusted
closing prices.
        x_mean = 0
        x_stdv = 0
        bins = []

        with open('2006.csv') as csvfile:
                readCSV = csv.reader(csvfile,
delimiter=',')
                for row in readCSV:
                        try:
                                current =
float(row[5])

        adjusted_closing_prices.append(current)
                        except ValueError:
                                print("")
        num_of_acp = len(adjusted_closing_prices)

        for x in range(1, num_of_acp):
                temp_x =
100*((adjusted_closing_prices[x] -
adjusted_closing_prices[x-1])/adjusted_closing_prices[x-
1])
                x_values.append(temp_x)

        # Process x_values to keep only values between -
2% to 2%
        for x in x_values:
                if(-2<x<2):

        x_processed_values.append(x)
        x_mean = sum(x_processed_values) /
len(x_processed_values)
        differences = [x - x_mean for x in x_values]
        sq_differences = [d ** 2 for d in differences]
        ssd = sum(sq_differences)
        variance = ssd / len(x_processed_values)
        x_stdv = sqrt(variance)
        print("x-mean: " + str(x_mean))
        print("x-stdv: " + str(x_stdv))

        # Create bins
        b=float(-2.0)
        bins.append(b)
        while b < 1.8:
                b += float(0.2)
                bins.append(b)

        # Plot histogram
        plt.hist(x_processed_values,bins,normed=True,hi
sttype='bar',rwidth=0.8)

        plt.xlabel("Observed X Values")
        plt.ylabel("")

        plt.title("Histogram of x values ");

        # Fits a normal distribution to the data.
        x_mean, x_stdv = norm.fit(x_processed_values)

        # Plot the PDF.
        xmin, xmax = plt.xlim()
        x = np.linspace(xmin, xmax, 200)
        p = norm.pdf(x, x_mean, x_stdv)
        plt.plot(x, p, 'k', linewidth=3)
        plt.show()
        # print("x-mean: " + str(x_mean))
        # print("x-stdv: " + str(x_stdv))


def print_y_values():
        adjusted_closing_prices = []
        x_values = []
        MA = []
        x_processed_values = []
        num_of_acp = 0
        x_mean = 0
        x_stdv = 0
        bins = []

        with open('2006.csv') as csvfile:
                readCSV = csv.reader(csvfile,
delimiter=',')
                for row in readCSV:
                        try:
                                current =
float(row[5])

        adjusted_closing_prices.append(current)
                        except ValueError:
                                print("")
        num_of_acp = len(adjusted_closing_prices)

        for x in range(4, num_of_acp):
                temp_ma =
adjusted_closing_prices[x]+adjusted_closing_prices[x-
1]+adjusted_closing_prices[x-
```

```
        2]+adjusted_closing_prices[x-
3]+adjusted_closing_prices[x-4]
        MA.append(temp_ma/5)
    for y in range(1, len(MA)):
        temp_x = 100*((MA[y]-MA[y-
1])/MA[y-1])
        x_values.append(temp_x)

    # Process values to keep only values between -
2% to 2%
    for x in x_values:
        if(-2<x<2):

    x_processed_values.append(x)
    x_mean = sum(x_processed_values) /
len(x_processed_values)

    differences = [x - x_mean for x in x_values]
    sq_differences = [d ** 2 for d in differences]
    ssd = sum(sq_differences)
    variance = ssd / len(x_processed_values)
    x_stdv = sqrt(variance)

    # Create bins
    b=float(-2.0)
    bins.append(b)
    while b < 1.8:
        b += float(0.2)
        bins.append(b)

    # Plot histogram.
    plt.hist(x_processed_values,bins,histtype='bar',rw
idth=0.8)
    plt.hist(x_processed_values,bins,histtype='bar',rw
idth=0.8)
    plt.xlabel("Observed Y Values")
    plt.ylabel("")

    plt.title("Histogram of Y values ");
    plt.show()

    x_mean, x_stdv = norm.fit(x_processed_values)
    print("y-mean: " + str(x_mean))
    print("y-stdv: " + str(x_stdv))

if __name__ == '__main__':
    print_x_values()
    print_y_values()
```

Problem 1.13 (20 pts)

There are many correct answers to this question. Question will be graded based on clarity, sound arguments, and any examples you provide. As long as you justify your argument you should be fine.