CSCE 421 HW 3

Jose Garza (225008812)

October 2019

Question 1

Question 1.a Plot a Histogram of the Outcome Variable

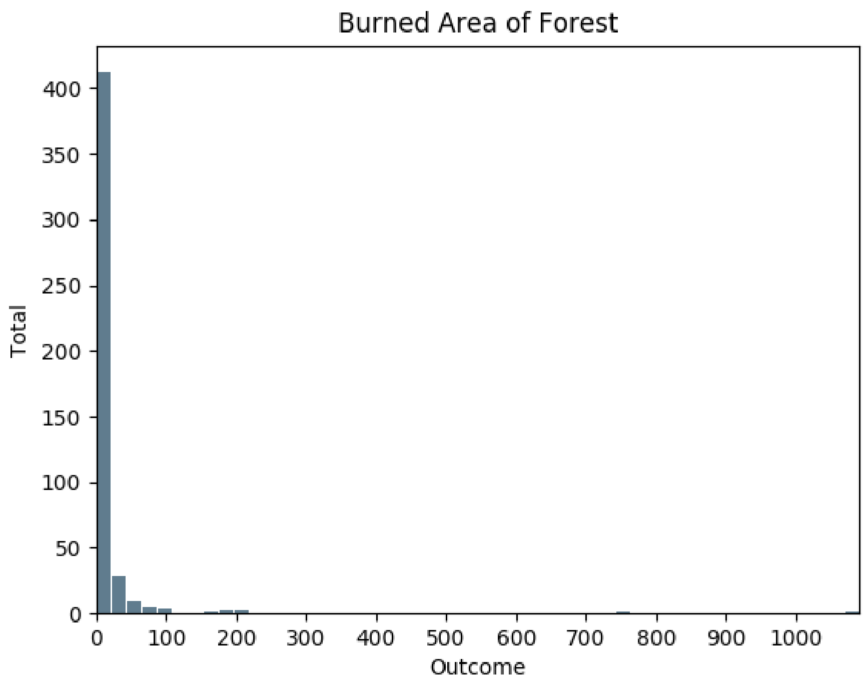


Figure 1: Burned Area of Forest

We can observe that the burned area of the forest seems to be relatively low, mainly under an area of 25. This will most likely give us a good indication of when a forest is not burnt, but could potentially lack in successfully predicting a forest that is.

Question 1.b Classification

On Code

```
Average Accuracys over the testing sets:
Fold 1 : 0.3695652173913043
Fold 2 : 0.45652173913043476
Fold 3 : 0.41304347826086957
Fold 4 : 0.5434782608695652
Fold 5 : 0.5434782608695652
Fold 6 : 0.5652173913043478
Fold 7 : 0.43478260869565216
Fold 8 : 0.5
Fold 9 : 0.6304347826086957
Fold 10 : 0.5

Final Average: 0.49782608695652186
```

Question 2 Maximum Likelihood Estimation:

Question 2.i Plot the number of tickets sold over these two years

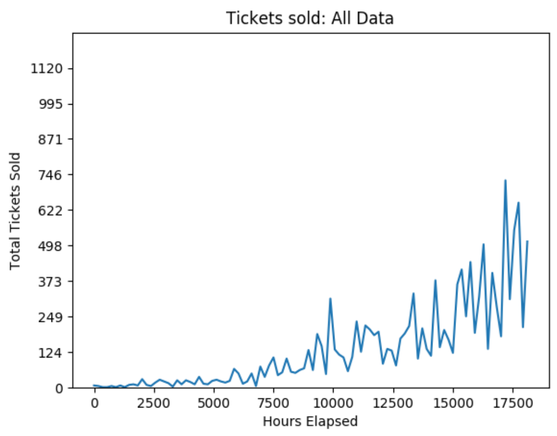


Figure 2: Ticket Data for 2 years

We can observe that the trend line of the tickets sold over the past two years is positive, where more tickets are sold each month compared to its past. There are spikes which might indicate popular seasons, however, the common trend is positive.

Question 2.ii Statistical Model of First Month of Data:

ii.a Plot the number of tickets sold over this first month

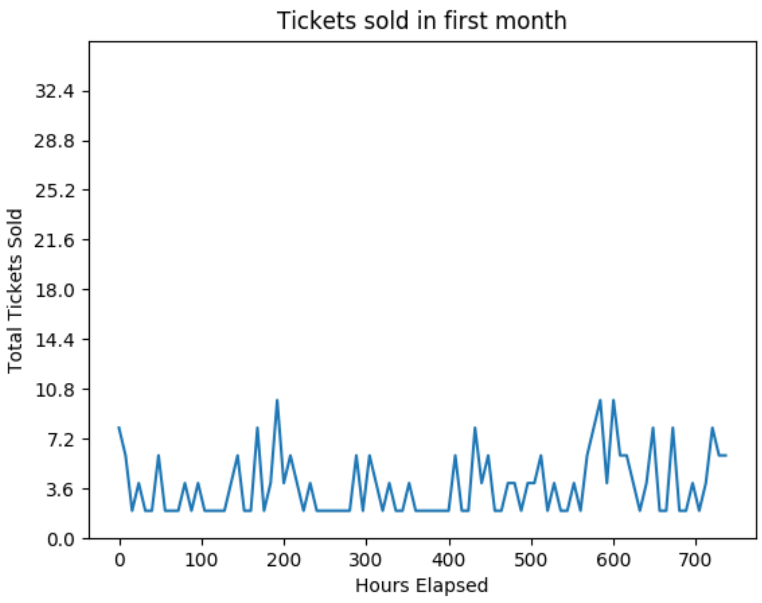
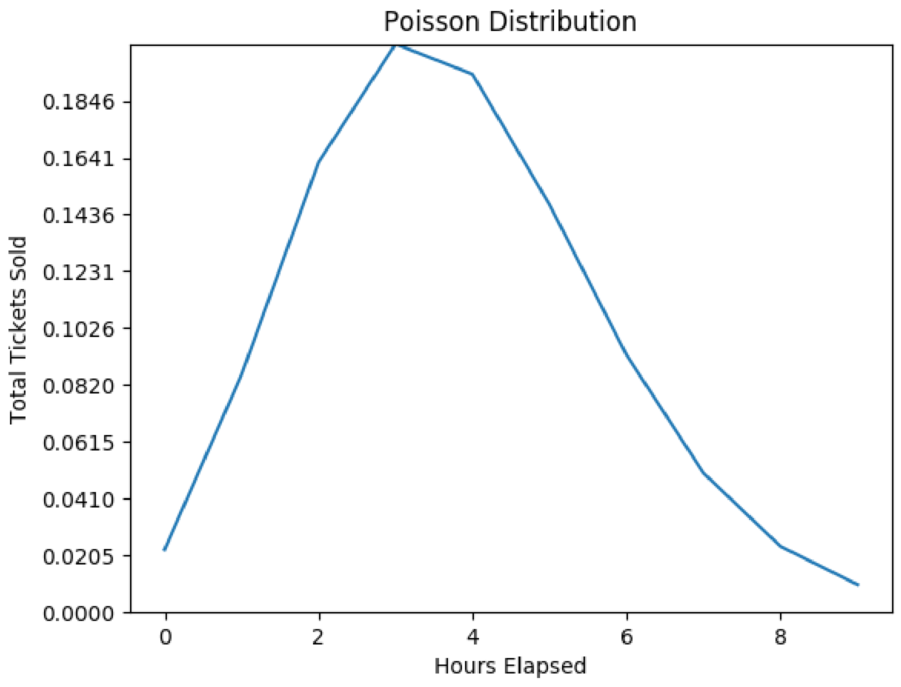


Figure 3: Ticket Data for the First Month

As mentioned before, we can see a more clear indication of popular seasons. Although the graph doesn't directly depicts when or why the tickets are popular in certain time intervals, it shows consistent patterns of when the total tickets sold is higher.

ii.b Poisson Distribution



$$f(\lambda) = \frac{e^{-\lambda}\lambda^{x}}{x\,!}.$$

$$l(\lambda) = \prod_{n=1}^{N} f(x_n) = \prod_{n=1}^{N} f(\log{[\frac{e^{-\lambda}\lambda^{x_n}}{x_n\,!}]})$$

ii.c Negative log-likelihood L(λ)

$$-\log l(\lambda) = \prod_{n=1}^{N} f(x_n)$$

$$L(\lambda) = \log(\lambda) = \log \prod_{n=1}^{N} f(x_n) = \sum_{n=1}^{N} \log f(x_n) =$$

$$= \sum_{n=1}^{N} \log{[\frac{e^{-\lambda}\lambda^{x_n}}{x_n\,!}]} = \sum_{n=1}^{N} [\log e^{-\lambda} + \log \lambda^{x_n} - \log(x_n\,!)]$$

$$= \sum_{n=1}^{N} \log\left[-\lambda\,(1) + x_n \log\lambda - \log(x_n!)\right]$$

ii.d Maximum likelihood estimation of (λ) Expression

$$0 = -\lambda N + [\log(\lambda)] \sum_{n=1}^{N} (x_n) - \log \sum_{n=1}^{N} x_n!$$

$$\frac{\partial L(\lambda)}{\partial \lambda} = -N + \frac{1}{\lambda} \sum (x_n) - \cancel{\log \sum (x_n)!}$$

$$\frac{-N\lambda + \sum x_n}{\lambda} = 0$$

$$-N\lambda + \sum x_n = 0$$

$$\hat{\lambda} = \frac{\sum (x_n)}{N}$$

ii.e Maximum likelihood estimation of (λ) For The First Month

$\sum_{n=1}^{N} (x_n) = 2818$

$N = 277$

$\lambda = 2818/277$

$\lambda = 3.7876344086021505$

Code 1

```
import matplotlib.pyplot as plt
import numpy as np
import random
import math
from sympy import *

from sklearn.linear_model import LogisticRegression


def ArrangeData(data):
    dataOut = []
    dataOutcomes = []
    for i in range(len(data[0])):
        row = []
        for j in range(len(data) - 1):
            row.append(float(data[j][i]))
        dataOut.append(row)
        dataOutcomes.append(data[(len(data) - 1)][i])
    return dataOut, dataOutcomes


def getW(data, y):
    xT = np.transpose(data)
    xTx = np.dot(xT, data)
    inverse = np.linalg.inv(xTx)
    w = np.dot(inverse, np.dot(xT, y))
    return w


def plotFeatures(data, feature, y_int, weight, f_name):
    outcome = len(data[0]) - 1
    plt.figure()
    plt.title('feature: ' + str(f_name))
    for i in range(len(data)):
        plt.plot(float(data[i][feature]), float(data[i][outcome]),
             marker='o', markersize=3, color="red")

    axes = plt.gca()
    x_vals = np.array(axes.get_xlim())
    y_vals = y_int + weight * x_vals
    plt.plot(x_vals, y_vals, '--')


def plotGraph(items, title='title'):
    step = math.ceil(len(items) / 100)
    x = range(0, len(items), step)
    plt.figure()

    plt.title(title)
    items = list(map(float, items))
    ymax = max(items)
    plt.xlabel('Hours Elapsed')
    plt.ylabel('Total Tickets Sold')
    plt.yticks(np.arange(0, ymax, step=ymax / 10))
    plt.ylim([0, ymax]) # places plot from 0 -> max

    new_items = [items[i] for i in range(0, len(items), step)]

    plt.plot(x, new_items)
```

```
def graphHist(data):
    x_max = max(data)
    plt.figure()
    plt.title('Burned Area of Forest')
    plt.xlabel('Outcome')
    plt.ylabel('Total')
    plt.xticks(np.arange(0, 1200, step=100))
    plt.xlim([0, x_max]) # places plot from 0 -> max
    plt.hist(data, bins=50, rwidth=0.9, color='#607c8e')


def linearRegression(data, feature):
    xArray = []
    y = []
    outcome = len(data[0]) - 1
    for row in range(len(data)):
        xArray.append([1, float(data[row][feature])])
        y.append([float(data[row][outcome])])

    w = getW(xArray, y)

    y_int = float(w[0])
    slope = float(w[1])
    for i in range(1, len(w)):
        plotFeatures(data, feature, y_int, slope, feature)


def GetSegmentsAccuracies(data, segment, class_y):
    accuracies = []
    for i in range(len(segment)):
        x_train, x_test, y_train, y_test = SplitDataRandomly(data,
             class_y, segment[i])
        lr = LogisticRegression(solver='lbfgs', max_iter=10000)
        lr.fit(x_train, y_train)
        predictions = lr.predict_proba(x_test)
        acc = CalcAcc(class_y, predictions, segment[i])
        accuracies.append(acc)

    return accuracies


def CreateSegments(data, total):
    totalSamples = len(data) - 1
    segments = random.sample(range(0, totalSamples), totalSamples)
    segment = []
    interval = math.floor(len(segments) / total)
    for i in range(total):
        row = []
        for j in range(interval):
            row.append(segments[(i * interval) + j])
        row.sort()
        segment.append(row)

    return segment


def SplitDataRandomly(data, class_y, segment):
    x_train = []
    x_test = []
    y_train = []
    y_test = []

    index = 0

    for i in range(len(data)):
        row = []
        for j in range(len(data[i])):
            row.append(float(data[i][j]))
        if segment[index] == i:
            x_test.append(row)
            y_test.append(class_y[i])
            if len(segment) - 1 > index:
                index += 1
        else:
            x_train.append(row)
            y_train.append(class_y[i])

    return x_train, x_test, y_train, y_test


def CalcAcc(actual, predicted, segment):
    correct1 = 0
    correct2 = 0

    total = len(actual)

    for i in range(len(predicted)):
        index = segment[i]
        if actual[index] == 0 and predicted[i][0] > 0.5:
            correct1 += 1
        if actual[index] == 1 and predicted[i][0] < 0.5:
            correct2 += 1

    return (correct1 + correct2) / len(predicted)


def main():
    data_1_csv = 'hw3_question1.csv'
    data_1 = np.loadtxt(data_1_csv, delimiter=",", unpack=True,
        skiprows=1)
    graphHist(data_1[12])
    data, outcome = ArrangeData(data_1)
    # grouped_hourly[day][hour][time/count]

    # Linear Regressions
    # tot_features = 1
    # ran = random.sample(range(0, len(data[0]) - 1), tot_features)
    # for x in range(tot_features):
    #     f1 = ran[x]
    #     linearRegression(data, f1)

    totalSegments = 10

    outcomes = [math.ceil(i / (i + 1)) for i in outcome]
    segment = CreateSegments(data, totalSegments)
    accuracies = GetSegmentsAccuracies(data, segment, outcomes)

    for i in range(len(accuracies)):
        print(accuracies[i])

    plt.show()


if __name__ == "__main__":
    main()
```

Code 2

```
import matplotlib.pyplot as plt
import numpy as np
from numpy import genfromtxt
import random
import graphs as graph
import math
from sympy import *
from sklearn.linear_model import LogisticRegression


def SplitDataRandomly(data, class_y, segment):
    x_train = []
    x_test = []
    y_train = []
    y_test = []
    index = 0
    for i in range(len(data)):
        row = []
        for j in range(len(data[i])):
            row.append(float(data[i][j]))
        if segment[index] == i:
            x_test.append(row)
            y_test.append(class_y[i])
            if len(segment) - 1 > index:
                index += 1
        else:
            x_train.append(row)
            y_train.append(class_y[i])

    return x_train, x_test, y_train, y_test


def PoissonDistribution(month):
    sze = len(month)
    euler = math.e
    mean = sum(month) / sze
    product = 1
    print(mean)
    distribution = []
    totalSum = 0
    for i in range(10):
        xi = i
        num1 = euler ** -mean
        num2 = mean ** xi
        num = num1 * num2
        den = math.factorial(xi)
        p = num / den
        distribution.append(p)
        totalSum += p
    graph.plotGraph(distribution, 'Poisson Distribution')
    return totalSum/sze


def main():
    data_csv = 'hw3_question2.csv'
    data = genfromtxt(data_csv, delimiter=",", dtype='unicode')

    grouped_hourly = [data[i:i + 24] for i in range(1, len(data), 24)]
    # grouped_hourly[day][hour][time/count]
    ticketSold = []
    date = []

    for i in range(len(grouped_hourly)):
        for j in range(len(grouped_hourly[0])):
            date.append(grouped_hourly[i][j][0])
            ticketSold.append(int(grouped_hourly[i][j][1]))

    graph.plotGraph(ticketSold, 'Tickets sold: All Data')

    daysInMonth = 31
    hoursInMonth = daysInMonth * 24
    month = [ticketSold[i] for i in range(0, hoursInMonth)]
    graph.plotGraph(month, 'Tickets sold in first month')

    PoissonDistribution(month)

    plt.show()


if __name__ == "__main__":
    main()
```

Code Graphs

```
import matplotlib.pyplot as plt
import math
import numpy as np


def plotFeatures(data, feature, y_int, weight, f_name):
    outcome = len(data[0]) - 1
    plt.figure()
    plt.title('feature: ' + str(f_name))
    for i in range(len(data)):
        plt.plot(float(data[i][feature]), float(data[i][outcome]),
            marker='o', markersize=3, color="red")

    axes = plt.gca()
    x_vals = np.array(axes.get_xlim())
    y_vals = y_int + weight * x_vals
    plt.plot(x_vals, y_vals, '--')


def plotGraph(items, title='title'):
    step = math.ceil(len(items) / 100)
    x = range(0, len(items), step)
    plt.figure()

    plt.title(title)
    items = list(map(float, items))
    ymax = max(items)
    plt.xlabel('Hours Elapsed')
    plt.ylabel('Total Tickets Sold')
    plt.yticks(np.arange(0, ymax, step=ymax / 10))
    plt.ylim([0, ymax]) # places plot from 0 -> max

    new_items = [items[i] for i in range(0, len(items), step)]

    plt.plot(x, new_items)
```