Assignment 3

ECEN 689: Machine Learning

Rangeesh Venkatesan

UIN: 527004482

Question 1

In each case here, we are required to compare least squares and logistic regression for learning a linear classifier. For the 2D data, the plot has been shown below the table.

Data Being Read

If the question asks us to solve for 10 samples, it has been assumed that 10 samples per class, hence a total of 20 samples has been taken per sub question

```
with open('Uniform_test.csv', newline='') as csvfile:
    data = list(csv.reader(csvfile))
at=(np.float_(data))
with open('Uniform_train.csv', newline='') as csvfile:
    data = list(csv.reader(csvfile))
a=(np.float_(data))

xt=at[:,0:2]
yt=at[:,2]
At=np.hstack((xt,np.ones((1000,1))))

N=10
c1=list(range(0,1000))
c2=list(range(1000,2000))
rds1=random.sample(c1,N)
rds2=random.sample(c2,N)

rs1=np.sort(rds1)
rs2=np.sort(rds2)

xnew=np.vstack((a[rs1,0:2],a[rs2,0:2]))
A=np.hstack((xnew,np.ones((2*N,1))))
y=np.concatenate((a[rs1,2],a[rs2,2]))
```

Linear Least Squares

```
W=np.matmul(np.linalg.inv(np.matmul(A.T,A)),np.matmul(A.T,y))
WX=np.matmul(At,W.T)
EB=np.matmul(At,W.T)
for i in range(0,1000):
        if WX[i]>0:
            WX[i]=1
        else:
            WX[i]=-1
1000-((WX-yt)==0).sum()
```

Logistic Regression

```
def logistic_loss(y,y_hat):
    return -np.mean(y*np.log(y_hat)+(1-y)*np.log(1-y_hat))
def sigmoid(Z):
    return 1/(1+np.e**(-Z))
m=len(y0)

for epoch in range(5000):
    Z=np.matmul(A,W)+b
    A1=sigmoid(Z)
    loss=logistic_loss(y0,A1)
    dz=A1-y0
    dw=1/m*np.matmul(A.T,dz)
    db=np.sum(dz)

    w=w-l*dw
    b=b-l*db

    if epoch%100==0:
        print(loss)
```

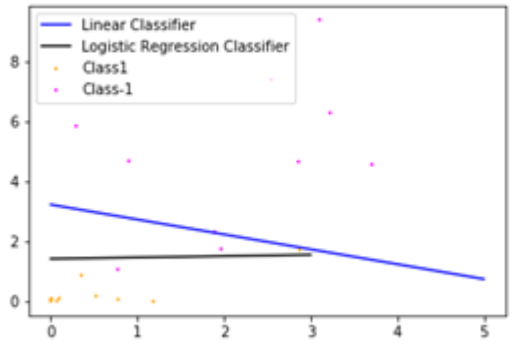
Accuracy

$$Accuracy\% = 100 * \frac{No.\ of\ Correct\ classifications}{No.\ of\ Test\ data}$$

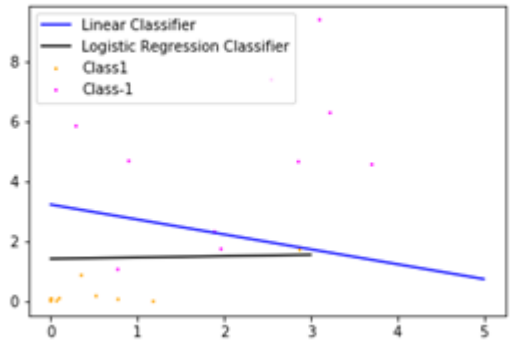
Gamma Functions - 2D Class conditional densities

SAMPLE SIZE	Linear Least Square accuracy	Logistic Regression accuracy
10	88.7	86.1
50	90.3	93.1
100	88.7	93.5
500	89.1	93.9

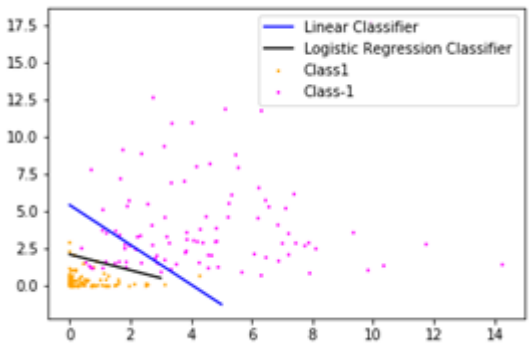
10 Samples



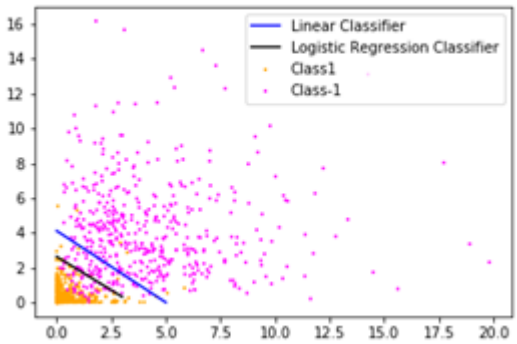
50 Samples



100 Samples



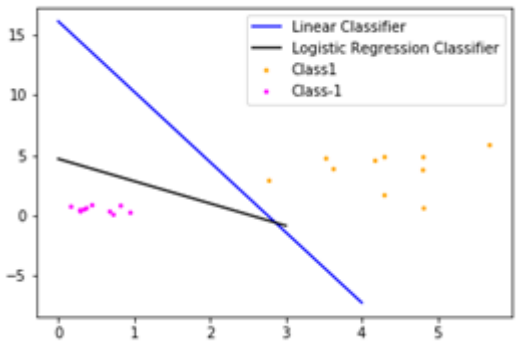
500 Samples



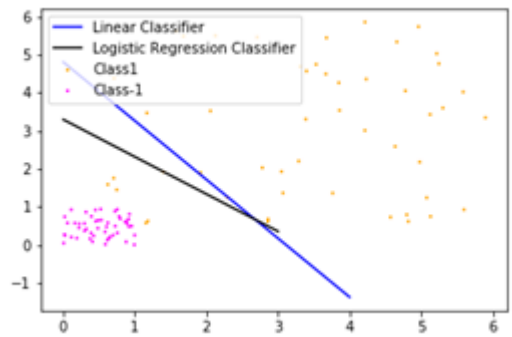
Uniform Densities: 2D Class conditional densities

SAMPLE SIZE	Linear Least Square accuracy	Logistic Regression accuracy
10	85.2	94.3
50	92.7	95.5
100	93.1	96.6
500	93.0	97.8

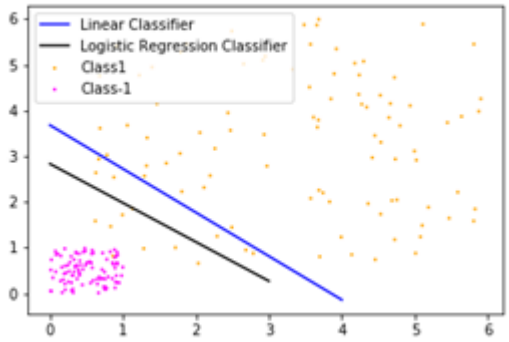
10 Samples



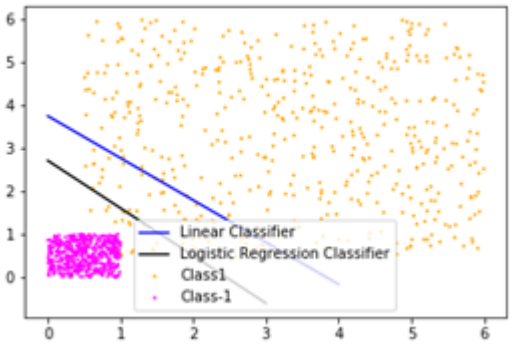
50 Samples



100 Samples



500 Samples



Normal Densities: 10D Class conditional densities

SAMPLE SIZE	Linear Least Square accuracy	Logistic Regression accuracy
10	83.8	79.4
50	94.8	87.0
100	93.7	86.3
500	94.0	87.6

Observations

- In general, the logistic regression based classifier has a higher accuracy than a linear classifier
- As the number of data samples increases, the performance of the both the classifier increases (albeit not linearly).
- On comparing the discriminability of the data points, we can see that the uniform densities data set has better discriminability over the gamma function data set.

Question 2

We need to compare two methods - three 2-class Classifiers(one vs rest Strategy) and 3-class linear classifier.

Data Being Read

Training Data - Out of the 150 Data points given (50 for each class), 40 from each class was taken as training data and the rest 10 from each class was taken as test data.

```
A401_d=np.concatenate((x1[0:40,:],x2[0:40,:]),axis=0)
A40=np.concatenate((A401_d,x3[0:40,:]),axis=0)

yt1=[]
for i in range(0,120):
    if i<40:
        yt1.append(1)
    else:
        yt1.append(0)
yt2=[]
for i in range(0,120):
    if i>=40 and i<80:
        yt2.append(1)
    else:
        yt2.append(0)
yt3=[]
for i in range(0,120):
    if i>=80:
        yt3.append(1)
    else:
        yt3.append(0)
```

One vs Rest

$$W^* = (A^T A)^{-1} A^T Y$$

```
wt1=np.matmul(np.linalg.inv(np.matmul(A40.T,A40)+l*np.eye(5)),np.matmul(A40.T,yt1))
```

```
wt1
```

```
array([ 0.03687217,  0.25469282, -0.17527012, -0.13660971,  0.16477177])
```

```
wt2
```

```
array([ 0.06987437, -0.49959785,  0.06715944, -0.2358833 ,  1.47651436])
```

```
wt3
```

```
array([-0.10674654,  0.24490503,  0.10811068,  0.372493  , -0.64128613])
```

The weights for each of the 2-class classifier is given above. The last term is the bias term.

3-class Linear Classifier

Using a 'one-hot' representation, we have y as a 3D vector. If the data point falls in a particular class, it's corresponding element in the one-hot representation is taken to be 1 and others to be 0.

```
Yhottr = np.zeros([120,3])
for j in range(0,120):
    if j<=40:
        Yhottr[j,0]=1;
    elif j>40 and j<=80:
        Yhottr[j,1]=1;
    else :
        Yhottr[j,2]=1;
whot=np.matmul(np.linalg.inv(np.matmul(A40.T,A40)+0*np.eye(5)),np.matmul(A40.T,Yhottr))
hotclass = np.argmax((np.matmul(A10t,whot)),axis = 1)+1
```

```
array([[ 0.07627817, -0.01404609, -0.06223207],
       [ 0.24412797, -0.43995868,  0.19583071],
       [-0.18588862,  0.08785451,  0.09803411],
       [-0.14711771, -0.1780974 ,  0.32521511],
       [ 0.0261516 ,  1.64094161, -0.6670932 ]])
```

Observations

One vs rest

- Some of the cases were ambiguous (there wasn't a definite answer on what class the data point falls in). At least 4 such data points weren't classified due to this ambiguity.
- Error: 26.67% - for the Ambiguous cases
- Error: 26.67% - for the Misclassification cases alone

3-class linear classifier

- Since all the three elements of y are compared and there is definitely an output from the classifier, there is no ambiguity involved.
- Error: 6.67% - for the misclassified cases

Question 3

We are required to compare the performance of the linear least squares and logistic regression for the 24D German credit data.

Data

70% of the given data (randomly sampled) is taken as training data

30% of the data (remaining) is taken as test data

```
a=np.loadtxt(open("german.data-numeric", "rb"))

for i in range(0,1000):
    if a[i,24]==2:
        a[i,24]=-1


c1=list(range(0,1000))
rds=random.sample(c1,700)
rds_sort=np.sort(rds)
x=a[rds_sort,0:24]
y=a[rds_sort,24]
np.ones(700).shape
xnew=np.hstack((x,np.ones((700,1))))
```

Linear Least Squares

```
A = xnew
w_star= np.matmul(np.linalg.inv(np.matmul(A.T,A)),np.matmul(A.T,y))
```

```
w_star
```

```
array([ 1.79618955e-01, -1.42903235e-02,  1.39529540e-01, -6.28861581e-04,
        2.88069614e-02,  4.52969183e-02,  8.87101952e-02, -1.55399800e-02,
       -2.55188472e-02,  1.32017706e-03,  5.16084864e-02, -1.66732280e-01,
        2.49112886e-02,  1.66339818e-01,  3.54357813e-01, -2.40651361e-01,
        2.36427698e-01, -2.54015441e-01, -2.70776302e-01, -5.66088422e-02,
        8.20843961e-02,  3.97480448e-01,  6.68692130e-02,  3.40594650e-02,
       -8.40216257e-01])
```

The weights W for the classifier is found to be as shown above. The last term is the bias term.

Logistic Regression

```
parameters
```

```
array([[ 0.50580472, -0.0381684 ,  0.38890611, -0.00274013,  0.10310249,
         0.11974022,  0.21349138, -0.05960118, -0.10906901,  0.00392915,
         0.13773397, -0.46030471,  0.04250914,  0.39421072,  0.66926972,
        -0.61303955,  0.70294453, -0.64785843, -0.55427491, -0.2708434 ,
         0.11568048,  0.68530403,  0.00722078, -0.01752441, -0.80140666]])
```

The Weights are shown above. The last term is the bias term.

Accuracy

$$Accuracy\% = 100 * \frac{No.\ of\ Correct\ classifications}{No.\ of\ Test\ data}$$

Accuracy (Linear Least Squares) = 75.3%

Accuracy(Logistic Regression) = 79%

Mean Square Error(Linear Least Squares) = 0.5897

Observations

- It is observed that the accuracies of both the classifiers are nearly the same, with the Linear least squares classifier having a better accuracy value.
- Due to the less complexity involved in using Linear Least Square classifier, we can use the Linear Least Square Classifier since they have comparable accuracies and this is computationally less expensive than the Logistic Regression Classifier.

Question 4

We are required to fit different polynomial functions and find the optimum degree and the coefficients for the polynomial function. The data given has a zero-mean Gaussian Noise added to it.

Data

70% of the given data (randomly sampled) is taken as training data

30% of the data (remaining) is taken as test data

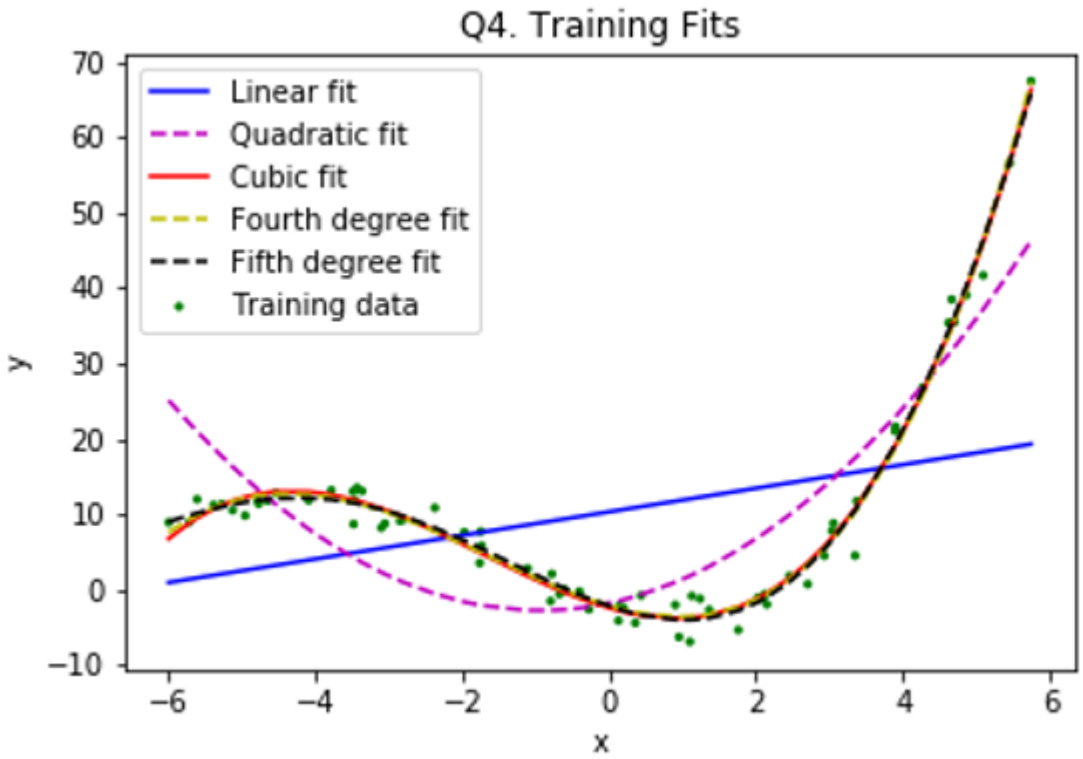
```
a=np.loadtxt(open("1D_regression_data.csv", "rb"))
c1=list(range(0,100))
rds=random.sample(c1,70)
rds_sort=np.sort(rds)
x=np.transpose(a[rds_sort,0])
y=np.transpose(a[rds_sort,1])

#Cube
A3=np.array([np.power(x,3),np.power(x,2),x,np.ones(70)]).T
W_star3=np.matmul(np.linalg.inv(np.matmul(A3.T,A3)+l*np.eye(4)),np.matmul(A3.T,y))
cost3=np.dot((y-np.matmul(A3,W_star3.T)),(y-np.matmul(A3,W_star3.T)))
```

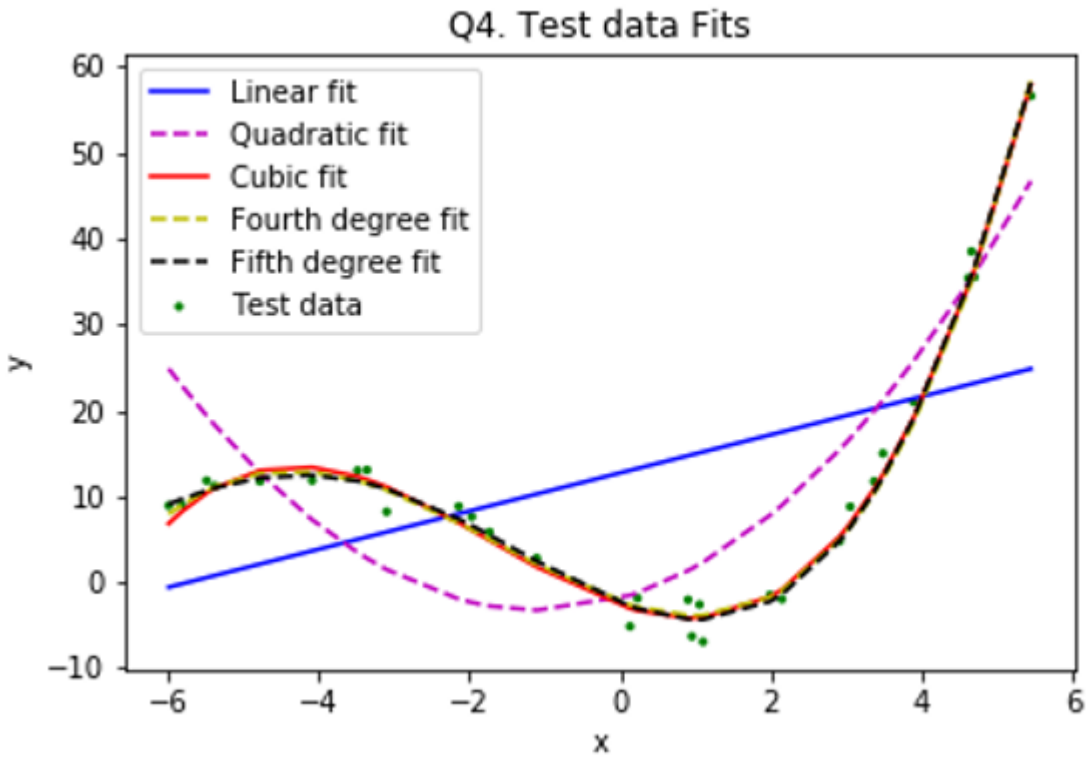
Results

Polynomial Function	Weights
Linear	[1.56419862, 10.317306]
Square	[1.09497894, 2.07813272, -1.78695273]
Cubic	[0.23672435, 1.21168547, -2.77992274, -2.45175198]
Fourth degree	[0.00333844, 0.23912179, 1.11496001, -2.81813922, -2.13797765]
Fifth Degree	[-1.66855e-03, 2.14807e-03, 3.02830e-01, 1.14140, -3.29023, -2.18031]

Variance of the given data: 3.69315 - ML Estimate of the variance is given by residual avg. squared error

Plots





Observations

- From the question given, we can identify that the value of the weights for the different degree polynomials behaves in a manner as expected. The cubic function seems to be the most optimum to describe the data.

- 4th and 5th degree polynomial seems to fit even better, but on observing the coefficient values, we find that they are near zero for the 4th and 5th powers
- Test errors were as follows: [11.25390314, 3.01963996, 0.13732424, 0.12744998, 0.1539235] - The regularisation constant was taken as lambda=0.1. Although the error for 4th degree polynomial is lower, the coefficient values suggest the cubic polynomial is a better fit
- The value for the variances are shown below: They tend to decrease as the degree increases

  [221.96922393, 63.36898945, 3.54877549, 3.40796124, 3.4781953]