Programming Assignment 1

Sankeerth.D
EE13B102
Electrical Engineering

September 21, 2016

Abstract

Linear regression, classification, kNN, logistic regression and Naive Bayes classifiers have been implimented and worked with in this assignment. The programs have been implimented in python, taking help from numpy and scipy libraries for array and gradient optimization respectively.

1 Problem 1: Synthetic Data Set Creation

For our case, a diagonal matrix whose elements are gaussian random variables is taken to be the covariance matrix. The covariance for the case presented is (shown to two decimal places):

Covariance = diag([-1.63 1.09 -0.31 -0.78 -0.33 0.30 -0.36 -1.44 0.25 0.33 -1.34 -0.48 1.74 1.10 0.07016 -1.09 0.66 1.10 -0.78 -1.28])

The mean is at (c, c, c.., c). For our case, we have chosen c = 0.35.

1.1 Program description:

Program imports numpy and scipy.stats package. stats package is used to create notmal random variables. The covariance is defined as a diagonal matrix, whose elements have been set as 20 samples of a standard gaussian random variable.

The mean of first set of samples is set at origin. The second class has its mean along (c, c,..., c) for some parameter c. The output is written in standard format, last element being +1 or -1 (depending on class) into files specified.

2 Problem 2: Linear Classification

The dataset DS1 is read off and linear regression is perfformed onto it. The coefficients are present in out_prob2.txt. The reported statistics are as follows:

Positive class precision and recall

S.no	Value
Accuracy	88.75%
Precision	0.883
Recall	0.893
F-measure	0.888

2.1 Program description:

We first read the data into lists and store input feature data into a numpy matrix X, and output y in a numpy array.

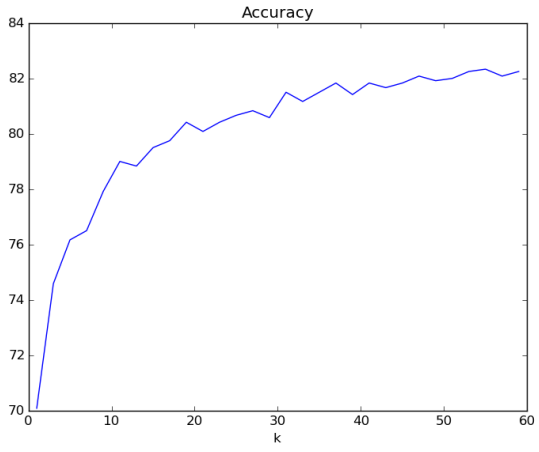
Two functions lsloss and lsgrad have been defined here. The imputs are X, y and beta. The computations are all vectorized for efficiency, and the output for gradient is a numpy array.

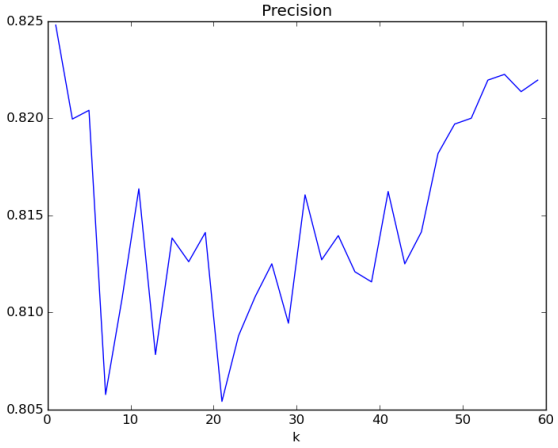
The standard optimization algorithm "Newton-CG" in scipy is used and the output vector, optimal beta, is written into output_prob2.txt file.

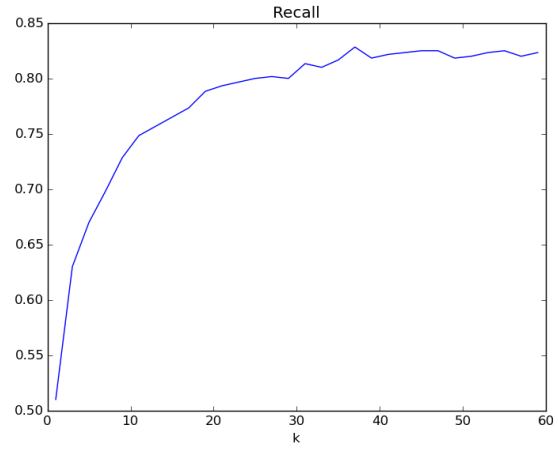
The performance_meas(beta, Xtest, ytest) function takes in the test data set computes the predicted y. From there, precision, reecall and f-measure are calculated and output given to file output_prob2.txt

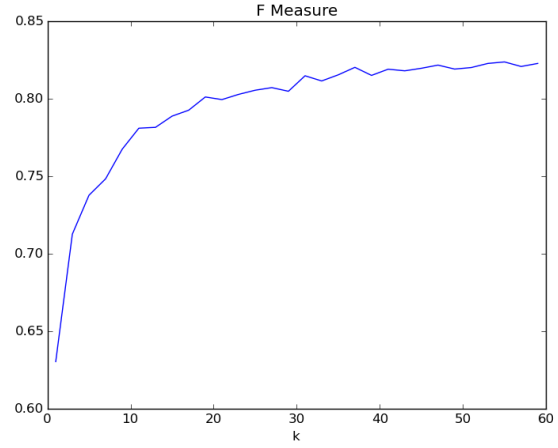
3 Problem 3: kNN classifier

For the dataset DS1, the kNN clasifier gives good results when k starts. Since our data has an unclear decision boundary, lower kyiedls a high variance decision boundary ang hence yields low accuracy. The file output_prob3.txt contains the statistics for the case of k given in the command line. The following graphs show the performance:




Precision
0.825
0.820
0.815
0.810
0.805
0
10
20
30
40
50
60
k


Recall
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0
10
20
30
40
50
60
k


F Measure
0.85
0.80
0.75
0.70
0.65
0.60
0
10
20
30
40
50
60
k

3.1 Program description:

There are two programs created, prob3fast.py and prob3slow.py.

prob3slow.py : Although there are efficient ways of implimenting kNN, it has been implimented here by iterating over all points and sorting. For a given point, the distance to the test-set points d and the class c are stored as a tuple (d, c). Then the tuples are sorted and the first k values are picked. The class values of these are summed up and stored in a variable score. if score > 0, the point belongs to class 1. Otherwise -1.

prob3fast.py : The class KNearestNeighbors from package sklearn.nearestneighbors is imported into the module. This has an efficient way of find the nearest neighbors. The class object is first trained, and then predictions are made. This program has been used to obtain the above plots.

3.2 Performance:

The running time of the prob3slow.py for given size of dataset is about 60 seconds. It's relatively slow because of the brute-force nature of the implimentation. The performance measureoutput is dumped in output_prob3.txt.

4 Problem 4: Linear Regression

The missing values in the dataset are replaced by the mean if the corresponding feature. This is not a good idea in general because if the features are correlated, the other features may drive off the actual value of the missing feature far off from it's mean value.

One possible remedy is to do a regression among other features taking the missing feature vector as output. The rest of the data is the training set (where all features are not missing).

4.1 Program description:

This program only does data processing over the features. The file is read line by line and the non missing features are summed up and stored in an array. The number of ?s are counted columnwise in this process. Using this data, mean in each column is calculated. While writing line by line, iterator checks for '?' in the input file and replaces these with the mean.

5 Problem 5: Linear, Ridge Regression

The data 80-20 division is made by randomly selecting the first 80% of the points. The remaining 20% are taken as a test set. Hence each time this test is performed, we get a different distribution of the data division. The datasets are written to CandC files.

As can be seen below, there is hude variation in test set error, indicating that the regressor is suffering from very high variance. The errors are ranging from 6 to 650. The high variance components need to be penalized, as done in prob6.

S.no	RSS Error
CandC1	850.66
CandC2	7.49
CandC3	515.96
CandC4	7.97
CandC5	6.72
Average RSS	277.76

5.1 Program description:

The linear regression part is implemented the same way as in problem 2. In addition to this, the 80-20 splits has been implemented in this itself. The data is firse shuffled up and the first 80% of data is set as training data, given to CandC-train file. This process is repeated 5 times and stored off as the required output files.

The data is read from the file using the get_data(filename) function into lists Xlist and ylist, later converted into numpy arrays.

5.2 Performance:

The program takes around 5-6 seconds to fully make data sets, and calculate coefficients 5 times. This is due to many read and write operations to files taking place.

6 Problem 6: Ridge Regression

Example: To do rigde regression for lambda = 10.0, command line is given as:

$python prob6.py 10.0

On choosing lambda as 0.0001, the average RSS error is consistently around 8.0-8.5. This shows that the highly varying components have been penalized and we get a reasonably good fit and coefficients. For a range of values of lambda from 0.00001 to 10, we get a good average test set error. Also, the individual errors are in the same range. Hence for the given dataset, the best lambda to choose is 0.1.

Hence choose lambda=0.1 yields a reasonable set of coefficients with moderate bias and variance.

lambda	Average RSS
0.00000001	104.84
0.0000001	13.15
0.000001	8.17
0.00001	8.06
0.0001	8.055
0.001	8.039
0.01	7.97
0.1	7.71
1	7.72
10	8.32
100	10.91

6.1 Program description:

The program is exactly the same as problem 5. THE VALUE OF LAMBDA IS TAKEN FROM THE COMMAND LINE. The output coefficients and errors are given to output_prob6.txt. The performance is similar to that of prob5.

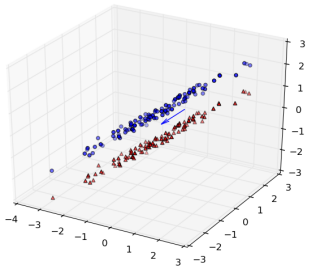
7 Problem 7: Principal Component Regression

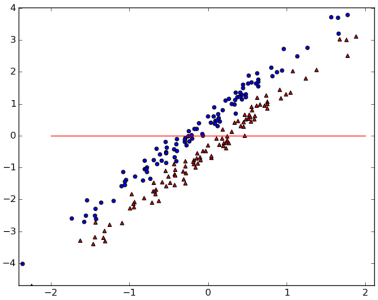
The principal directions are found out using max. eigenvalue. The beta corresponding to this is found and the values are given to output_prob7.txt file. The points are projected onto a plane parallel to beta, so that the decision boundary looks like a line in this reference frame. Hence one component is chosen to be beta itself the other perpendicular vector is found out by one gram shmit iteration over (1, 0, 0). Hence the basis for this plane are obtained as v1(=beta) and v2. The decision boundary is perpendicular to beta. So in this plane, if beta is y axis, the decision boundary os a horizontal line through the origin (standardized data). The transformation matrix to project points onto this plane is [v2 v1], with v1=beta being y axis. The resulting test set performance plots are as shown. The arrow indicated direction of principal component.

As we can see, PCA performs poorly on the data. This is attributed to the fact that PCA doesn't incorporate the role of y in deciding principal directions. Even of the two classes are far away, the direction chosen might only reduce the variance of fitted date, but need not perform well.

Positive class precision and recall:

S.no	Value
Accuracy	58.75%
Precision	0.58
Recall	0.58
F-measure	0.58





7.1 Program description:

To get the eigenvalues, numpy.linalg package is imported. The matplotlib and Axes3d libraries are used for plotting the data. Afterfinding the eigenvalues, the test set points are projected onto the plane and plotted.

8 Problem 8: Linear Differential Analysis

The sample mean of each class is found first. To obtain the LDA fit, first the within class variance and between class variance matrices have first been obtained. Then, the max. between class /within class variance is obtained as maximum eigenvalue of inv(Sw)*Sb matrix. Since we're using only one direction, this direction is chosen to estimate y.

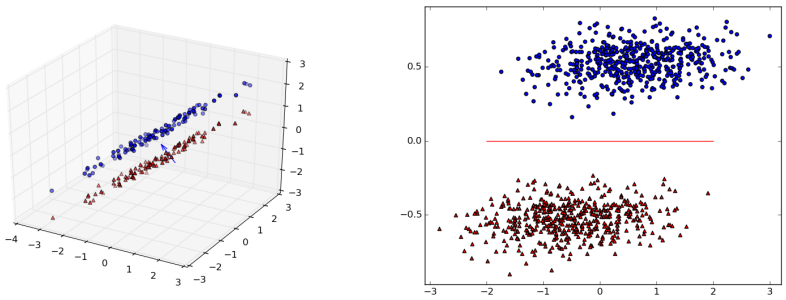
The following procedure is exactly similar to problem 7 from now on, difference being the principal vector in both the cases.

The statistics of the data (for both positive and negative class precision,recall) are as follows:

S.no	Value
Accuracy	100%
Precision	1.0
Recall	1.0
F-measure	1.0

We get a perfect separating plane between the datasets. Hence obviously this is a better approach for the problem.

On plotting the data and it's projection, we obtain the following plot:



8.1 Program description:

The inverse and eigenvalues are obtained using the numpy.linalg package. The rest of the program is similar to prob7.

8.2 Notes:

The LDA performs better than PCR in this case because we have only found the max. variance component in prob7, but this direction need not yield a good direction to classify points (but only reduce variance), unlike in prob 8, where the learnt direction is constructed in such a way as to get a direction with max. separation between classes, i.e. by maximizing (between class variance)/(within class variance).

9 Problem 9: Logistic Regression

The command line scripts have been included in the python file prob9_boyd. Note that the binary files for Boyd's code need to be present in the same folder to be able to run this file. Hence the values, coefficients and test-set predictions are found out using boyd's code. The python script contains the sequence of commands that need to be executed to run Boyd's code.

To run a regularized logistic regression with lambda = 0.01, run:

python prob9_boyd 0.01

The data is taken into python file to calculate the statistics of the program. The stats are as follows:

when lambda = 0.0:

Positive class precision and recall are noted here:

S.no	Value
Accuracy	82.5%
Precision	0.76
Recall	0.95
F-measure	0.84

Negative class precision and recall:

S.no	Value
Accuracy	82.5%
Precision	0.76
Recall	0.96
F-measure	0.84

On varying lambda, over a range,

lambda	Accuracy	Precision	Recall	F-measure
0.0001	85%	0.79	0.95	0.86
0.001	90%	0.86	0.95	0.90
0.01	97.5%	0.95	1.0	0.975
0.1	87.5%	0.86	0.9	0.88

Goig to lambda > 1 yields a zero in one of the denominators in one of the measures. Also, the accuracy drops to near 50%. Hence the best value of lambda is clearly 0.01.

9.1 Program description:

The binaries from the file implemented by Boyd's program are downloaded and kept in file in the current folder. The binaries are run using the python script prob9_boyd.py. The program data_processing.py is used to create usable output files for the binaries. Running the method gives the performance on the test set as 7/40 with lambda=0.0. When lambda = 1.0, the error increases to 50%. On varying values, it's found that at lambda = 0.01, we get test error of 1/40. Hence best value of lambda to choose is 0.05

10 Problem 10: Naive Bayes method

There would be three programs, namely the data_processing_prob10.py, prob10a.py and prob10b.py. prob10a.py deals with the multinomial with/without dirichlet prior case and prob10b.py has the bernoulli with/without beta prior case. data_processing is used to get the inputs of the files. To run the file, put the Q10 directory in the same directory as the programs prob10a.py and prob10b.py.

10.1 Directions:

Using the programs:

If it is required that part4 and part5 are the test sets, and rest all are training sets, type in the command

python prob10a.py 3

or
python prob10b.py 3
In general for part i and part i+1 as test sets, type in
python prob10a.py i-1
or
python prob10b.py i-1
Default value of i = 1
The parameters of the priors are set inside the files, in the first few lines.

10.2 Multinomial with/without Dirichlet prior:

The case for multinomial with dirichlet prior is the same as that with a constant prior when the alpha = 1.0. Hence parts 1 and 3 of the problem is incorporated into a single file. The parameters alpha of dirichlet distribution are all taken to be equal (to alpha). The MLE estimates, on working out, the independant probability terms in the multinomial turns out to be

A*(n+alpha)

A:normallizing factor

n: number of words w in the concatenated text.

The program is implemented with refernece to Manning, Raghavan's book

If there are a total of V words in the vocabulary, there are 2V such samples are found out (V in each class).

The output is written into the file out_prob10a.txt

For the case of uniform prior:

	Test 1, 2	Test 3, 4	Test 5, 6	Test 7, 8	Test 9, 10
Accuracy	98.17%	96.36%	96.80%	96.82%	98.19%
Precision	1.0	0.9915	0.967	0.992	1.0
Recall	0.97	0.94	0.976	0.952	0.967
F-measure	0.98	0.967	0.972	0.97	0.984

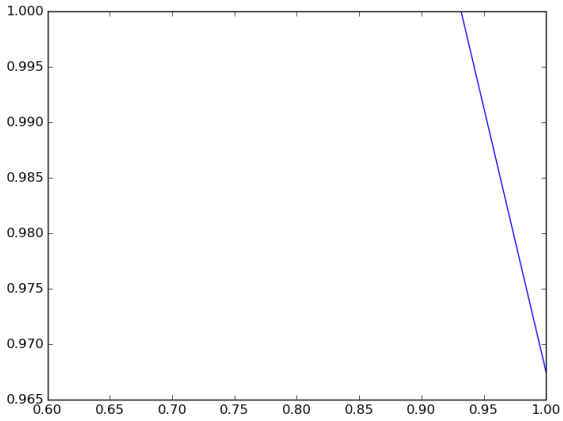
On varying alpha, the accuracy tends to reduce. if alpha = 1000, accuracy drops to 67%.

The precision and recall have been calculated for different alphas given by:

alphavec = [1.0, 500, 1000, 2000, 3000, 4000, 5000, 10000]

The following PR curve has been obtained.

NOTE: The PR curve is plotted by sweeping accross dirichlet prior coefficients instead as it as the precisions and recalls are almost 1 in test cases, not demonstrating the PR characteristic.



The x axis is the precision, y axis the recall. Notice that as P increses, R decreases.

10.3 Bernoulli with/without Beta prior:

NOTE: The running time of this program is considerably longer than that of the previous case. It takes around 20 seconds to classify and get statistics of one file.

The case for a constant beta prior can be acheived by setting constants a and b of the beta distribution to 1. On evaluating the parameters by MAP, we'll get the parameters to be equal to

(nw + a)/(N+a+b)

nw is the number of documents containing the word w. a and b are beta function parameters. for the purpose if this exercise, we'll choose a = b.

On running the experiment with a = b = 1, we get almost identical results as that of the previous case (upto a higher decimal).

	Test 1, 2	Test 3, 4	Test 5, 6	Test 7, 8	Test 9, 10
Accuracy	98.17%	96.36%	96.80%	96.82%	98.19%
Precision	1.0	0.9915	0.967	0.992	1.0
Recall	0.97	0.94	0.976	0.952	0.967
F-measure	0.98	0.967	0.972	0.97	0.984

The accuracy decreases gradually to average of 92% when a = b=2, and decreases further on increaseing a and b.

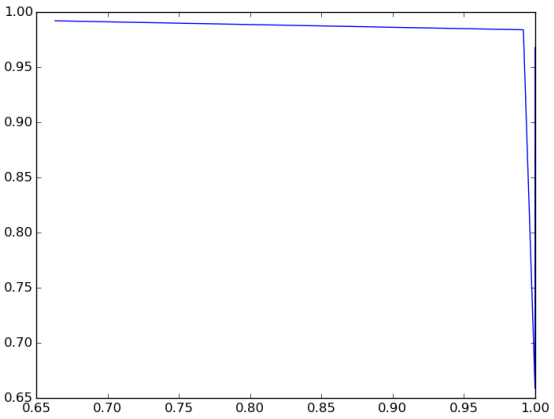
Setting a =b=0sharply decreases the accuracy to 71% The precision goes to 0.69, recall to 0.99 and f_measure to 0.79.

Changing the value of parameter a (=b), the P-R samples are taken (since P and R are nearly one in the above case.) The plot when varying parameter across the values:

a = b = [0.0, 0.0001, 0.50, 10.0, 25, 50.0] fro test p1 and p2

is shown below:

NOTE: The PR curve is plotted by sweeping accross beta prior coefficients instead as it as the precisions and recalls are almost 1 in test cases, not demonstrating the PR characteristic.



The x axis is the precision, and y axis is the recall. Notice that as P increases, R decreases. The above plot is by changing the values of beta.