<u>CMSC 678 – HOMEWORK 2</u>

Akshay Peshave (DA74652)

PROBEM 1 – Probability Decision Boundary

We'll calculate the expected loss for the possible class label predictions, which are two in this case, using the loss values provided in the loss matrix for each (ŷ,y) pair.

For ŷ=0:

$E[L(\hat{y}=0)] = Pr(y=0) . loss_{y=0} + Pr(y=1) . loss_{y=1}$

$= p_o . 0 + p_1 . 10$

$= 10.p_1$

For ŷ=1:

$E[L(\hat{y}=1)] = Pr(y=0) . loss_{y=0} + Pr(y=1) . loss_{y=1}$

$= p_0 . 5 + p_1 . 0$

$= 5.p_0$

The expected loss for a choice of ŷ depends on the probabilities p0 and p1. We would choose ŷ=0 as the class label given X if the expected loss in doing so is less than that in choosing ŷ=1, i.e.

$E[L(\hat{y}=0)] < E[L(\hat{y}=1)]$

$10.p_1 < 5.p_0$

$p_1 < p_0/2$

Substituting $p_0 = 1 - p_1$, we get $p_1<1/3$.

Therefore, **$\theta = 1/3$**.

Thus, if $p_1 \geq 1/3$ we can choose ŷ=0 to minimize expected loss else choose ŷ=1.

In general we can see that the decision of choosing ŷ is dependent on the probabilities and that the decision can be made based on a threshold θ, which is a function of the probabilities of the actual values of the class label in the training data set.

PROBLEM 2 – Dual Preceptron Algorithm

This algorithm compares a training instance with all other training instances to **quantify miss-classifications**. The comparison has following aspects:

- $x_l.x_i$ → this dot product provides us with the delta in magnitude of the two training instance vectors.
- $y_i y_l$ → this quantity provides the sign to entire product. i.e. if the two class labels are the same this product will be positive else negative.
- α_l → it signifies the number of training instances that render this instance misclassified.

The product of the first two components can be interpreted as follows:

- the feature vectors are similar (i.e. they point in the same general direction in the plane) but the class labels disagree, in which case the product will be negative.
- The features are similar and the class labels agree, in which case the product will be positive.
- The features are dissimilar and the class labels disagree, in which case the product will be positive.
- The features are dissimilar but the class labels agree, in which case the product will be negative.

Negative products indicate misclassified instances. **The α is a count of the evidence (number of instances) found suggesting miss-classification of current instance**. This adds to the value of the product, and **does so transitively**. α_l in the product ensures that the miss-classification evidence of the lth instance is also accounted in the ith instance's miss-classification calculation.

When we sum these products, we are essentially **aggregating the evidence found suggesting correct and incorrect classification of the current training instance**. If the incorrect classification evidence outweighs the evidence suggesting correct classification then the value of the summation will be negative and we can conclude that the ith training instance is miss-classified.

We loop over this process to account for all relative miss-classifications and should be able to constructively **adjust the weight vector once the α vector converges.**

References:
http://cvit.iiit.ac.in/thesis/ranjeethMS2007/thesis/node19.html

PROBLEM 3 – Double Counting the Evidence

$\hat{y}$ = argmax(y_k) [P(y_k) . $\prod_{i=1 \text{ to } 2}$ P(X_i | y_k)]
$\hat{y}$ = argmax(y_k) [P(y_k) . P(X_1 | y_k) . P(X_2 | y_k)]

To find: expected error when utilizing X_1

X_1	P (y=T | X_1)	P (y=F | X_1)
T	0.8x0.5 = 0.40	0.3x0.5 = 0.15
F	0.2x0.5 = 0.10	0.7x0.5 = 0.35

E[error|X_1] = P(y≠$\hat{y}$ | X_1=T) + P(y≠$\hat{y}$ | X_1=F) = 0.15 + 0.10 = **0.25**

To find: expected error when utilizing X_2

X_2	P (y=T | X_2)	P (y=F | X_2)
T	0.5x0.5 = 0.25	0.5x0.1 = 0.05
F	0.5x0.5 = 0.25	0.5x0.9 = 0.45

E[error|X_2] = P(y≠$\hat{y}$ | X_2=T) + P(y≠$\hat{y}$ | X_2=F) = 0.25 + 0.05 = **0.30**

To find: expected error when utilizing X_1 and X_2

X_1	X_2	P (y=T | X_1 , X_2)	P (y=F | X_1 , X_2)
T	T	0.2	0.0150
T	F	0.2	0.1350
F	T	0.05	0.0350
F	F	0.05	0.3150

E[error|X_2] = 0.015 + 0.135 + 0.05 + 0.035 = **0.235**

To find: expected error when utilizing X_1, X_2 and X_3
We do not consider cases where $X_2 \neq X_3$ as the training set does not have such cases and the conditional probabilities will be calculated based in this assumption for X_3 and hence will be identical to those of X_2.

X_1	X_2	X_3	P (y=T | X_1 , X_2)	P (y=F | X_1 , X_2)
T	T	T	0.1000	0.0015
T	F	F	0.1000	0.1215
F	T	T	0.0250	0.0035
F	F	F	0.0250	0.2835

E[error|X_2] = 0.0015 + 0.1 + 0.0035 + 0.025 = **0.13**

Naive Bayes is based on an independent feature probabilistic model i.e. it assumes that the values of each feature is independent of the other. When we introduce X_3 as a replica of X_2, we are essentially introducing noise which **causes the Naive Bayes classifier to believe it is doing a better job of classifying accurately**. This is evident from the fact that the odds of predicting the class label as T when X_1 and X_2 are T went up from 10 to 66. Also the training set does not aid in modeling the case of $X_2 \neq X_3$.

In logistic regression we attempt to find a probability distribution which is a good fit over the training data and then try to predict the class of a given instance by fitting it within this probability distribution. Unlike Naive Bayes, here we do not try to fit the instance over the training set and predict the class label based on a prior and likelihood product.

When we attempt to learn the weight vector over the training set containing X_3 as the duplicate of X_2 the logistic regression process will assess that X_3 does not have substantial effect on the class label of the instances and will adjust X_3's component in the weight vector accordingly over successive iterations.

PROBLEM 4 – Logistic Regression Implementation

Data-set 1: Breast Cancer Wisconsin Diagnostic Data set
This data has two class labels: B=Benign=1, M=Malignant=0
The best accuracy achieved is 92.3976608187 with 2000 iterations.

For η=0.01

```
Iterations: 10
Accuracy: 22.8070175439
Weight    Vector    =    [58.67823500000009,    101.84520000000008,    358.4406499999999,
386.60950000000366,    0.6215429499999996,    0.1816134000000008,    -0.3248597645000002,
-0.18273625499999982,    1.1880574999999982,    0.4663726500000003,    0.2031284999999987,
8.655561500000001,    0.9096909999999965,    -144.53365999999994,    0.0548792600000005,
0.09681757000000026,   0.1165962290000001,   0.044138660000000024,   0.15904040000000014,
0.025735869499999946,    52.60527500000008,    126.80545000000006,    318.31370000000004,
-1031.180999999999,    0.7910952500000004,    0.1126671999999997,    -0.47928002499999933,
-0.11400144499999978, 1.676542, 0.507017500000001]

Iterations: 30
Accuracy: 71.9298245614
Weight    Vector    =    [188.22525500000037,    319.49440000000027,    1155.7320499999996,
1892.4295000000116,    1.9337308499999994,    0.6397180000000025,    -0.8638592105000008,
-0.48438735499999924,    3.694798499999996,    1.4405768500000007,    1.1175474999999957,
26.625836499999995,       6.173004999999987,       -366.00424,       0.16937161000000017,
0.30681818000000094,   0.36960374100000026,   0.1410622100000002,   0.48931080000000043,
0.0792263934999998,     173.8137350000002,     399.4270500000001,     1060.9222000000004,
-1841.7689999999966,    2.473458750000001,    0.5785718000000001,    -1.1409154849999974,
-0.21613592499999917, 5.2496649999999985, 1.581845600000003]

Iterations: 50
Accuracy: 22.8070175439
Weight    Vector    =    [264.5898550000005,    455.6802000000006,    1614.0729499999998,
1739.2315000000167,    2.7980648499999994,    0.7728603000000036,    -1.5374488605000016,
-0.8508915549999991,     5.349972499999995,     2.099450950000001,     0.994856499999994,
38.00000849999999, 3.8325029999999805, -635.79153, 0.2431004400000003, 0.4055126700000014,
0.4827274020000004,    0.18699948000000013,    0.7012735900000011,    0.1124015744999997,
242.0522450000003,     570.2621500000001,     1462.021800000001,     -3972.530999999994,
3.582645750000002,    0.4551537000000002,    -2.234793974999996,    -0.5316344349999986,
7.588291999999997, 2.286677500000004]

Iterations: 100
Accuracy: 88.8888888889
Weight    Vector    =    [468.38564500000064,    788.9497000000011,    2861.5208500000003,
3932.722500000027, 4.83508905, 1.3536663000000064, -2.645779296500001, -1.423610404999999,
9.2483494999999,      3.6100350500000014,      2.3482524999999903,      63.08109649999998,
9.260738999999958,    -1011.4671999999998,    0.40238883000000036,    0.6424833400000023,
0.7567690670000006,    0.30362556000000007,    1.166978590000002,    0.1841596664999995,
446.01215500000086,    1001.1137500000003,    2703.4419000000007,    -3807.8219999999874,
6.264436450000002,    1.1538434000000017,    -3.4002066349999946,    -0.7057576549999975,
13.317146999999999, 3.9737474000000073]

Iterations: 1000
Accuracy: 87.7192982456
Weight    Vector    =    [1270.9825656999665,    1593.2533121205001,    7443.0852170346225,
3503.7502813120673,    11.821340278287309,    -4.787004149028575,    -19.645797845576336,
-9.129795108222778,     23.053975824661634,     9.309790341864955,     1.6308399350339036,
106.16373368120021,     -64.06953281932776,     -3527.9680637082756,     0.6408073897071205,
-1.1481351560762159,    -1.8322089298378441,    -0.26035109741924806,    2.1218048155610867,
0.22619786878400155,     1333.3097472553598,     2010.6642902019003,      7515.774928241337,
-3803.6494245846516,     14.848821709647419,     -21.446787032062176,     -40.07367662330075,
-10.662563223888045, 31.179763330795257, 8.780057248021876]

Iterations: 2000
Accuracy: 92.3976608187
Weight    Vector    =    [1349.3548311964469,    859.4531076127349,    7537.390852620249,
1903.5011271060262,     9.921563884014711,     -16.78256104459133,     -35.814455963520494,
-15.478556925167052,     20.720148238185274,     8.670594043150892,     -1.3981759837346228,
63.50788156115068,     -143.0551917710631,     -4254.678265914609,     0.21985479666960275,
-4.485747219655174,    -6.136298092505728,    -1.3148270035615277,    1.1228427434932489,
0.01947820000575663,     1440.1001979081395,     901.7620482670926,     7382.605976225036,
-3283.296878393316,     10.539381893975646,     -65.12902470653047,     -90.71184816038713,
-24.135106375388276, 19.643651052049144, 4.98464413571763]

Iterations: 5000
Accuracy: 88.8888888889
Weight    Vector    =    [1817.136183299503,    -731.2761698455288,    9233.908038463784,
2268.7864623260853,     6.061229116824742,     -52.726081582317306,     -86.03337476003732,
-35.763253443186926,     16.543649465488848,     8.283342776634402,     4.050858516996821,
4.16956883084926,     -289.03264639143174,     -5287.3810886777965,     -0.7425461413502026,
-13.567827728959408,     -17.94175978399588,     -4.13439360773538,     -1.2719810106504514,
-0.43621371880007437,    1929.5161078232932,    -1780.5140794115785,    7932.706749417155,
-3751.725813712104,     -0.852287867861544,     -198.5327366380969,     -247.60963619946222,
-67.55810346305505, -12.883818012350165, -5.098647780982971]

Iterations: 10000
Accuracy: 89.4736842105
Weight    Vector    =    [2480.754817738638,    -2676.640241946668,    11409.182974631442,
2164.372101417562,    -1.0881493422828918,    -111.85972726607885,    -165.94713151420797,
-68.44188523967284,     7.321167029194031,     7.0436845510360415,     30.488148355408654,
-18.242175936196777,    -416.88535842320215,    -5239.877087112677,    -1.9479332190608947,
-28.32195336061592,     -36.27212411288633,     -8.430329539063381,     -5.508385374330081,
-1.1621694347187284,     2640.4430654572857,     -5331.160100089489,     8204.79532905184,
-3799.726228162722,     -20.387391476212787,     -420.6623359618927,     -505.0022292771449,
-139.86605442608496, -71.7403968739181, -22.78920595679801]
```

Data set 2: Contraceptive Method Choice Data set
This data set has three class labels: No Choice, Short Term, Long Term.
I aggregated these to two class labels: 0=No Choice, 1=Short Term/Long Term
This data-set was less accurate and gave erratic results for η=1 v/s η=0.01. I couldn't figure out why though.
The best accuracy achieved is 63.5746606335 with 2000 iterations.

For η=1

```
Iterations: 10
Accuracy: 59.72850
Weight Vector = [454.5, 1760.5, 1192.0, 2124.5, -31.0, 167.5, 44.0, 1236.0, -229.0]

Iterations: 30
Accuracy: 59.72850
Weight  Vector  =  [10079.162318183382,  5727.142391233934,  4187.523188311911,  6758.5,
104.88079707797789, 658.3807970779778, 623.6423912339337, 4294.761594155956, -641.0]

Iterations: 50
Accuracy: 40.27149
Weight    Vector    =    [-12379.839235159192,    6177.142235899686,    3287.523032977663,
7422.49992233286,      -686.1192029220218,      320.38079707797806,      -998.3576475996285,
3837.761438821699, -1083.0]

Iterations: 100
Accuracy: 40.27149
Weight    Vector    =    [-11781.9941479935,    12821.540638168195,    7306.920095824372,
15181.783243810554,     -1160.0100291025815,     773.4899708974184,     -1324.2062304211706,
8328.927242236501, -2093.8284078584247]
```

For η=0.01

```
Iterations: 10
Accuracy: 59.72850
Weight  Vector  =  [4.545000000000044,  17.605,  11.92,  21.245,  -0.3099999999999987,
1.6749999999999972, 0.4399999999999977, 12.359999999999994, -2.2899999999999996]

Iterations: 30
Accuracy: 59.72850
Weight    Vector    =    [105.60549076226073,    57.52655014520058,    42.253719300534556,
67.9980502143525,      1.1653845130021026,      6.677596608520678,      6.522006676063051,
43.281467222021675, -6.368059524055605]

Iterations: 50
Accuracy: 59.72850
Weight     Vector     =     [133.66269664324165,     93.41201833755554,     67.165625301745,
111.06075506101469,      0.9735088365624112,      10.344862194245934,      8.418174285413915,
69.1838490121625, -10.827200786222988]

Iterations: 100
Accuracy: 59.72850
Weight     Vector     =     [64.71172923871433,     152.43298403951053,     99.2622363130166,
181.68353616926345,      -5.249774449737348,      13.71991514588021,      0.8096409906578561,
107.34789797704084, -20.94497886392786]

Iterations: 1000
Accuracy: 63.1221719457
```

```
Weight Vector = [-108.724795549194, 743.2361877619659, 292.616977047889,
719.3892790923135, -189.68508634129293, -29.77751054571924, -175.74297428845705,
408.80218552243133, -175.75619206740726]
```

```
Iterations: 2000
Accuracy: 63.5746606335
Weight Vector = [-108.90952163786204, 909.2557378916676, 192.50558122318094,
739.6110616676001, -365.5123117885233, -84.30513688684177, -225.15161491751155,
405.33839217034887, -299.9543161981474]
```

LOGISTIC REGRESSION IMPLEMENTATION

```
'''
Created on Mar 5, 2013

@author: akshaypeshave
'''

from string import *
import decimal
import math

class logisticRegression_2Class:
    def __init__(self, trainingDataFile, testDataFile, numberOfFeatures):
        self.number_of_features=numberOfFeatures
        self.trainingDataFile = trainingDataFile
        self.testDataFile = testDataFile

        #initialize weight vector
        self.weightVector = []

        for index in range(0, self.number_of_features):
            self.weightVector.append(0.0)


    def VectorDotProduct(self, w, x):
        dotProduct = 0.0
        for component in range(0,len(w)):
            dotProduct += float(w[component]) * float(x[component])

        return dotProduct

    def VectorSum(self, vector1, vector2):
        vectorSum = []

        for component in range(0,len(vector1)):
            vectorSum.append(float(vector1[component]) + (0.01*
float(vector2[component])))

        return vectorSum

    def LearnWeightVector(self):
        for i in range(0,10000):
            #initialize gradient vector
            gradientVector=[]
            for index in range(0, self.number_of_features):
                gradientVector.append(0.0)

            #iterate over all samples from training set
            trainingDataSet=open(self.trainingDataFile)
            trainingInstance = split(str(trainingDataSet.readline()).rstrip(),',')

            while trainingInstance[0] != '' :
                w_dot_instance=self.VectorDotProduct(self.weightVector,trainingInstance)

                class1_instance_probability= 1/(1 + decimal.Decimal(-
w_dot_instance).exp())
                class1_instance_probability=float(class1_instance_probability)
                instance_error = float(trainingInstance[self.number_of_features]) -
class1_instance_probability

                for vectorComponent in range(0,self.number_of_features):
                    gradientVector[vectorComponent] += (instance_error *
float(trainingInstance[vectorComponent]))

                trainingInstance = split(str(trainingDataSet.readline()).rstrip(),',')

            trainingDataSet.close()
            self.weightVector=self.VectorSum(self.weightVector, gradientVector)

        print 'Weight Vector = ' + str(self.weightVector)

    def TestWeightVector(self):
        testDataSet=open(self.testDataFile)

        testInstance=split(str(testDataSet.readline()).rstrip(),',')

        totalTestInstances=0
        totalIncorrectPredictions=0

        while testInstance[0] != '' :
            totalTestInstances+=1
            w_dot_instance = self.VectorDotProduct(self.weightVector, testInstance)

            if w_dot_instance > 0 and testInstance[self.number_of_features]=='0':
                totalIncorrectPredictions+=1
            elif w_dot_instance < 0 and testInstance[self.number_of_features]=='1':
                totalIncorrectPredictions+=1

            testInstance=split(str(testDataSet.readline()).rstrip(),',')

        print 'Test Instances \t: '+str(totalTestInstances)
        print 'Correct Predictions\t: '+str(totalTestInstances-totalIncorrectPredictions)
        print 'Accuracy \t\t: ' + str((float(totalTestInstances)-
totalIncorrectPredictions)*100/totalTestInstances)


classifier=logisticRegression_2Class('training', 'test', 30)
#classifier=logisticRegression_2Class('cmc_training.data', 'cmc_test.data', 9)
classifier.LearnWeightVector()
classifier.TestWeightVector()
```