Homework 2

A and B due March 3

C due at 11:59pm March 6

A. Mathematics:

Consider a classification problem where we wish to determine if a human subject is likely to get a concussion in the next year. We use four features - x_1 (Age), x_2 (concussHistory), x_3 (FavoriteSport), x_4 (LastEducation), x_5 (Gender) . Each feature takes on one of a discrete number of values, shown below:

Age	Child	Teen	Adult	
concussHistory	Never	Recent	DecadesAgo	
FavoriteSport	Boxing	Golf	Rugby	Baseball
LastEducation	Masters	HighSchool	College	
Gender	Female	Male		

We wish to classify each user as either y^i=LikelyConcuss or y^i=NotLikelyConcuss.

1. How can the features above be transformed to use a logistic classifier? Explain in 1-4 sentences.

2. How many parameters are required to learn a separating hyper-plane (**w** and any other necessary elements) for logistic classification with the features converted in question 1? (Work from your answer to question 1. If you could not figure out question 1, assume we have a new space of 6 continuous numeric features x_1, x_2, …, x_5 – this may or may not be a valid result from question 3.)

3. Which of the following vectors has (approximately) unit magnitude? (At most 1.1, at least 0.9.)

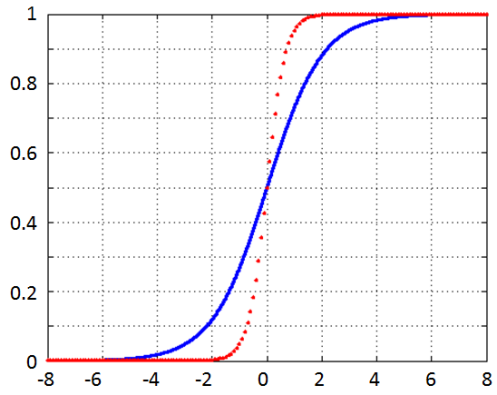
(a) $\begin{bmatrix} 0.3 \\ -0.6 \\ 0.7 \\ -0.6 \end{bmatrix}$ (b) $\begin{bmatrix} 0 \\ 0.6 \\ -0.4 \\ 0.7 \end{bmatrix}$ (c) $\begin{bmatrix} 1 \\ -1 \\ 0 \\ -1 \end{bmatrix}$ (d) $\begin{bmatrix} 2 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$

4. What is the projection (dot product) of the vectors below onto the unit magnitude vector from question 3? (If you believe none of the vectors in question 3 have unit magnitude, just use vector (d) from question 3.)

(a) $\begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ (b) $\begin{bmatrix} -2 \\ 1 \\ 0 \\ 2 \end{bmatrix}$ (c) $\begin{bmatrix} 0.5 \\ 1.5 \\ -1 \\ 1.5 \end{bmatrix}$

5. Let us assume we have a classifier with $\boldsymbol{w} = \begin{bmatrix} 1 \\ -4 \\ 6 \end{bmatrix}$ and $b = 2$. Provide **two** distinct inputs $\mathbf{x}^i$ that would be on the boundary between class 0 and 1, specifically find two inputs $\mathbf{x}^i$ where $g(\mathbf{x}^i;\mathbf{w})=0.5$.

6. In class, we define the logistic/sigmoid function as $g(h) = \frac{1}{1+e^{-h}}$, producing the curve in solid blue shown to the right. Note while $g(h)\approx 0$ when h is a very low negative number ($h<-4$) and $g(h)\approx 1$ when h is a very high positive number ($h>4$), when h is close to zero, $-2<h<2$, g(h) transitions roughly linearly between 0.1 and 0.9 . Consider in contrast $g^{alt}(h)$ in the dashed **red** curve, which stays close to 0 for input h = - 2 and stays close to 1 for input h=2.



(a) <u>How can we adjust the definition of g(h) to create the dashed **red** curve $g^{alt}(h)$?</u> (You do not need to give the exact equation for $g^{alt}(h)$, but explain what alterations if any are made to the numerator, denominator, exponent, etc.)

(b) We now wish to have a modified sigmoid g^{alt2} where $g^{alt2}(h) < .05$ only when h<-10 and $g^{alt2}(h) > .95$ only when h>10. Provide an exact algebraic formula for $g^{alt2}(h)$ to fit this specification.

In class, we considered the logistic function output as a pseudo-probability and used it to define a likelihood for data given the parameters in **w** (for the following three questions we will assume b is included in **w**).

$$L(y|\boldsymbol{x};\boldsymbol{w}) = \prod_i \left(1 - g(\boldsymbol{x}^i;\boldsymbol{w})\right)^{(1-y^i)} g(\boldsymbol{x}^i;\boldsymbol{w})^{y^i}$$

Note for binary classification, y=0 or y=1. The goal is to find **w** to maximize L for all data ($\mathbf{x}^i$,y^i). Let us say we skip the logistic function and learn the two-way classifier **w** to maximize a sum:

S(y|**x**; **w**) = $\sum_i \left(\frac{d}{\boldsymbol{w}^T\boldsymbol{x}^i}\right)$

7. Replace *d* with a valid expression such that S(y|**x**; **w**) will be largest when **w** is correctly selected to correctly classify data from class 0 (y^i=0) as $\mathbf{w}^T\mathbf{x}$<0 and from class 1 (y^i=1) as $\mathbf{w}^T\mathbf{x}$>0. *d* should be an algebraic expression involving some variables **w**, $\mathbf{x}^i$, and/or y^i. For example: *d=|**x**^{i}|* *or d=10+y^i* .

8. What is the derivative of S(y|**x**; **w**) with respect to a single element of **w**, w_j? (Consider the definition of $\mathbf{w}^T\mathbf{x}$.)

9. Recall L2 regularization affects the gradient ascent learning rule by adding $+\frac{w_j}{\lambda}$.

Let us now create a **new** regularization with the prior probability $(w_j) = e^{-\frac{|w_j-\mu_j|^3}{\alpha}}$.

What is the resulting change to the gradient ascent learning rule?

B. SVMs

1.

(a) Let us use SVM to define a linear classifier with the following support vectors and α's:

$$\boldsymbol{x}^1 = \begin{bmatrix}-2\\3\\0\\0\end{bmatrix}, \boldsymbol{x}^2 = \begin{bmatrix}-2\\0\\0\\3\end{bmatrix}, \boldsymbol{x}^3 = \begin{bmatrix}2\\-4\\2\\0\end{bmatrix}, \boldsymbol{x}^4 = \begin{bmatrix}1\\-4\\0\\5\end{bmatrix}$$

$$y^1 = +1, \quad y^2 = +1, \quad y^3 = -1, \quad y^4 = -1$$

$$\alpha^1 = 1.2, \quad \alpha^2 = 0.9, \quad \alpha^3 = 1.2, \quad \alpha^4 = 0.9$$

What is the resulting **w**?

(b) Consider the same support vectors as before but a different set of α's:
$\alpha^1 = 1.5,\ \ \alpha^2 = 1,\ \ \alpha^3 = 2,\ \alpha^4 = ???$

Presuming $\mathbf{x}^1$, $\mathbf{x}^2$, $\mathbf{x}^3$, and $\mathbf{x}^4$ are the only support vectors available, what value must α^4 have?

What is the resulting **w**?

2. The respective update rules for logistic classification and SVM learning are

$$w \leftarrow w + \varepsilon \boldsymbol{x}^i \left(y^i - g\right)$$

and

$$w \leftarrow w - 2w - \sum_i y^i \lambda_i \boldsymbol{x}^i$$

(a) Explain in 1-2 sentences how logistic classification learning minimizes the impact of correctly-labeled data points far from the hyperplane.

(b) Explain in 1-2 sentences how SVM learning minimizes the impact of correctly-labeled data points far from the hyperplane.

(c) Which of the two methods (or both) allows all data points to be used in the final hyper-plane definition?

We did not yet define "Kernel functions", but I recommend students try this question anyway … just treat it as a norm function that takes in two vectors. It will count for a small amount of credit.

3. (6 points) Consider the following kernel function:

$$K(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a}^{\mathrm{T}}\, \mathbf{b}}{|\boldsymbol{a}||\boldsymbol{b}|}$$

And further consider the following support vectors:

$$\boldsymbol{x^4} = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -2 \end{bmatrix} \qquad \boldsymbol{x^5} = \begin{bmatrix} 0 \\ 0.5 \\ 0 \\ 1 \end{bmatrix} \qquad \boldsymbol{x^6} = \begin{bmatrix} -0.2 \\ 1 \\ 1 \\ -0.5 \end{bmatrix}$$

What is the output of the kernel function for each pair of inputs below?

K($\mathbf{x}^4$,$\mathbf{x}^6$) K($\mathbf{x}^6$,$\mathbf{x}^5$)

C. Programming

In this question you will implement algorithms for learning parameters and classifying with Logistic Classification.

To submit Part C, leave your code in the Python file hw2.py inside your private/CIS5800 directory.

Accessing our data
The file hw2data.mat is available on our website (and on erdos using `cp ~dleeds/MLpublic/hw2data.mat .`) Load this file into Python to get access to the fullData numpy array. For this array, each row is one example student. Columns 0 through 9 represent **ten** features, corresponding to diverse information about the student – her/his GPA, SAT scores, age, level of financial aid, etc. The last column (column fullData[:,10]) represents the student major y^i – 0 (History) or 1 (Chemistry).

1. Write a function called **sigmoidLikelihood** that takes in the *m* features for *n* students as a np-array of dimension (n,m) , the corresponding class labels for *n* students as a np-array of dimension (n), and the separating hyper-plane represented as **w** by a np-array of dimension (m+1). The b scalar offset is included as the final element of **w**. The function returns the sigmoid-based pseudo-likelihood of each student being in the corresponding class, as a np-array of dimension (n).

Use the following syntax:
```
LVector = sigmoidLikelihood(X, y, w)
```

The pseudo-likelihood is computed using the sigmoid function (which you must implement for this homework).
2. The function sigmoidLikelihood returns an np-array of pseudo-likelihoods for *n* data points. If we wish to calculate the full pseudo-likelihood across all input, we will multiply the function outputs: $\mathcal{L} = \prod_i P(\boldsymbol{x}^i, y^i; \boldsymbol{w})$ or `np.prod(LVector)` . However, if too many small probabilities are multiplied together, the total pseudo-likelihood will be approximated to exactly zero, losing potentially valuable information. This estimation to zero can be avoided by taking the pseudo **log-likelihood** `np.sum(np.log(LVector))` .

(a) If LVector contains all values of 0.05 ([0.05, 0.05, 0.05, …, 0.05]), how many data points (elements in LVector) are needed for np.prod to estimate the pseudo-likelihood as perfectly 0?

(b) What is the pseudo-log-likelihood equivalent given the number of data points from part (a)?

Include your answers as a comment in hw2.py.

3. Write a function called **learnLogistic** that takes in the initial hyper-plane vector $\mathbf{w^0}$ as a np-array of dimension (m+1) with b as the final element, training data **x** with n data points and m features as a np-array of dimension (n,m), the correct labels for each point y as a np-array of dimension (n), and the number of learning loops *K*. The function outputs the new weights **w** as a np-array of dimension (m+1) and the log-likelihood of the input data after each loop as a np-array of dimension (K).

Use the following syntax:

```
w, LHistory = learnLogistic(w0, X, y, K)
```

Note: For each "loop," learnLogistic should loop through each data point in the training set and use gradient ascent to update **w** for each data point.

```
# this is pseudo-code!
for dataPt i :
  for feature j :
   update[j] += stepSize * updateRule(x[i,j],y[i])
for feature j :
  w[j] += update[j]
```

Assume stepSize=0.01.

4. In class, we discussed that learnLogistic can run faster if you use fewer loops. For example, instead of looping on feature j, numpy allows you to write:

```
# this is pseudo-code!
update += stepSize * updateRule(x[i,:],y[i])
```

(a) Write `learnLogisticFast` to accept the same inputs and produce the same outputs as `learnLogistic` above, but using vector math to remove at least one loop (you may be able to remove more than one, but only have to remove one).

(b) Compare the speeds of `learnLogistic` and `learnLogisticFast`, by running them on the data in hw2data.mat with at least 10 loops. **Provide your answer in a comment.**
Specifically, use time.time() to determine the time it takes each function to run:

```
# this part is not pseudo-code
# comment this out/delete it before submitting hw2.py
# JUST SUBMIT THE TIME IT TAKES TO RUN learnLogistic AND
# learnLogisticFast
import time
timeStart=time.time()
w, LHistory = learnLogistic(w0,X,y,K)
timeEnd=time.time()
```

```
print(timeEnd-timeStart)
```

5. Write a function **logisticClassify** that takes in the *m* feature values for the *n* data points as a np-array with dimensions (n,m) and the weight vector ***w*** as a np-array with dimension (m+1) (with b as the final element of w). The function returns the 0/1 label for each data point as a np-array of dimension (n).

Use the following syntax:

```
classLabels=logisticClassify(x,w)
```