Homework 1
Naive Bayes and Logistic Regression

CMU 10-701: Machine Learning (Fall 2014)
https://piazza.com/cmu/fall2014/1070115781/home
OUT: Sept 12, 2014
DUE: Sept 25, 2014, 11:59 PM

START HERE: Instructions

- **Late days:** The homework is due Thursday September 25th, at 11:59PM. You have five late days to use throughout the semester, and may use at most three late days on any one assignment. Once the allowed late days are used up, each additional day (or part of a day) will subtract 1 from the normalized score for the assignment. View the full late days policy on Piazza.
- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get inspiration (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators and resources fully and completely (e.g., "Jane explained to me what is asked in Question 3.4" or "I found an explanation of conditional independence on page 17 of Mitchell's textbook"). Second, write up your solution independently: close the book and all of your notes, and send collaborators out of the room, so that the solution comes directly from you and you alone.
- **Programming:**
  - **Octave:** You must write your code in Octave. Octave is a free scientific programming language, with syntax identical to that of MATLAB. Installation instructions can be found on the Octave website. (You can develop your code in MATLAB if you prefer, but you *must* test it in Octave before submitting, or it may fail in the autograder.)
  - **Autograding:** All programming problems are autograded using the CMU Autolab system. The code which you write will be executed remotely against a suite of tests, and the results used to automatically assign you a grade. To make sure your code executes correctly on our servers, you should avoid using libraries which are not present in the *basic* Octave install.
- **Submitting your work:** All answers will be submitted electronically through the submission website: https://autolab.cs.cmu.edu/10701-f14.
  - Start by downloading the submission template. The template consists of directory with placeholders for your writeup ("problem1.pdf", "problem2.pdf"), and a single sub-directory for the programming parts of the assignment. *Do not modify the structure of these directories or rename these files.*
  - **IMPORTANT:** When you download the template, you should confirm that the autograder is functioning correctly by compressing and submitting the directory provided. This should result in a grade of zero for all programming questions, and an unassigned grade (-) for the written questions.
  - **Writeup:** Replace the placeholders with your actual writeup. Make sure to keep the expected file names ("problem1.pdf"), and to submit one PDF per problem. To make PDFs, we suggest pdflatex, but just about anything (including handwritten answers) can be converted to PDF using copier-scanners like the ones in the copier rooms of GHC.
  - **Code:** For each programming sub-question you will be given a single function signature. You will be asked to write a single Octave function which satisfies the signature. In the handout linked above, the "code" folder contains stubs for each of the functions you need to complete.
  - **Putting it all together:** Once you have provided your writeup and completed each of the function stubs, compress the top level directory *as a tar file* and submit to Autolab online (URL

above). You may submit your answers as many times as you like. You will receive instant feedback on your autograded problems, and your writeups will be graded by the instructors once the submission deadline has passed.

Problem 1: Naive Bayes vs. Logistic Regression [Nicole - 30pts]

Naive Bayes (NB) and Logistic Regression (LR) form what is called a *generative-discriminative pair* of classifiers. In this problem, we will explore their relationship in some detail.

Assume that we are performing a binary classification task with n samples. Each sample has p binary features $X_1, \ldots, X_p \in \{0, 1\}$, and a corresponding label $Y \in \{0, 1\}$.

1. For each of Naive Bayes and Logistic Regression briefly state:

   (a) The formula for the conditional likelihood, $P(Y = 1 \mid X, \ldots, X_p)$, assumed by each model. **[4 pts]**

   (b) The classification rule. **[2 pts]**

   (c) The parameters we have to estimate. **[2 pts]**

   (d) The method we use to do Maximum Likelihood Estimation (MLE) of the parameters. **[2 pts]**

2. Next, we show that Naive Bayes and Logistic Regression form a pair:

   (a) Briefly describe (in one sentence) what makes two classifiers a generative-discriminative pair. **[2 pt]**

   (b) Generative vs. discriminative algorithms learn parameters which have completely different intuitive interpretations. Figure 1 is a plot of a sample data set with two features, $X_1, X_2 \in \mathbb{R}$. Draw on the plot a visualization of what the parameters encode for Naive Bayes and Logistic Regression respectively. Please clearly indicate which is which. **[2 pts]**

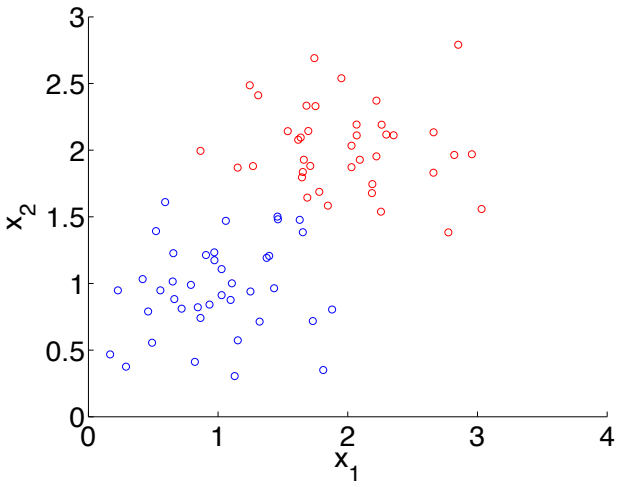

Figure 1: A sample data set. Clearly indicate with solid or dashed lines what the parameters learned in Naive Bayes and Logistic Regression respectively encode.

   (c) Prove that we can write the Naive Bayes class distribution $P(Y = 1 \mid X, \ldots, X_p)$ in a form that matches the Logistic class distribution. To start, it will help to write $P(X_j = 1 \mid Y = 1) = \theta_j$. Hint: this makes $P(X_j = x \mid Y = 1) = \theta_j^x (1 - \theta_j)^x$. **[4 pts]**

   (d) Show that Logistic Regression is the discriminative version of Naive Bayes. Hint: what happens when you optimize the conditional likelihood you derived in the previous part? **[2 pts]**

(e) The classifier we learn by optimizing the Naive Bayes conditional likelihood is nevertheless not the same as what we would have learned had we learned a Logistic classifier directly. What modeling assumption makes it somewhat less generic than Logistic Regression? **[2 pt]**

(f) Under what circumstances would Naive Bayes and Logistic Regression produce (asymptotically) the same classifier? **[2 pt]**

3. Let's compare the performance properties of these two algorithms. Let h_l be a Logistic Regression classifier, and h_b be a Naive Bayes classifier. Let h_l^* be the best possible Logistic classifier, and h_b^* the best possible Naive Bayes Classifier. These are the classifiers we would get if we had an infinite amount of data on which to train. The error rate of a classifier h is given by $\epsilon(h)$.

(a) Which classifier (h_l or h_b) will generally have a lower asymptotic error rate? Hint: think about what we showed in part 2 this question. **[2 pts]**

(b) Recall that we are performing classification in p dimensions (i.e. with p features) and have n samples. The error rate for h_l can be bounded as follows (note that we will be able to prove this later in the course; for now you can take it for granted):

$$\epsilon(h_l) \leq \epsilon(h_l^*) + O\left(\sqrt{\frac{p}{n}\log\frac{p}{n}}\right)$$

How many samples do we need (on what order should n be) to keep $\epsilon(h_l) = O(\epsilon(h_l^*))$? **[1 pt]**

(c) The parameters of h_b are guaranteed to be within $O\left(\sqrt{\frac{1}{n}\log p}\right)$ of those of h_b^*. How many samples do we need to feel confident that the performance of h_b is close to that of h_b^*? **[1 pt]**

(d) Given the answers to parts (b) and (c), can you think of a situation in which it would be better to use h_b than h_l? **[2 pts]**

Problem 2: Multi-class Logistic Regression [Anthony - 40 pts]

In class we learned about binary logistic regression. In this problem we consider the more general case where we have more than two classes. Let us denote the number of classes by C. Then, the conditional probability of the output class being c, given the input data point $\boldsymbol{x}$ is given by the following expression:

$$P(y = c \mid \boldsymbol{x}, \mathrm{W}) = \frac{e^{\boldsymbol{w}_c^\top \boldsymbol{x}}}{\sum_{c'=1}^{C} e^{\boldsymbol{w}_{c'}^\top \boldsymbol{x}}},$$

where $c \in \{1, \ldots, C\}$ and where W is a matrix whose rows are the weight vectors of each class, $\boldsymbol{w}_c$ for $c \in \{1, \ldots, C\}$. During the training phase of the algorithm we are given a set of n input-output pairs, $\{\langle \boldsymbol{x}_i, y_i \rangle\}_{i=1}^n$ and we want to *learn* the values of the weight vectors for each class that maximize the conditional likelihood of the output labels, $\{y_i\}_{i=1}^n$, given the input data $\{\boldsymbol{x}_i\}_{i=1}^n$ and those weights, W. That is, we want to solve the following optimization problem:

$$\mathrm{W}^* = \underset{\mathrm{W}}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i \mid \boldsymbol{x}_i, \mathrm{W}).$$

Note here that we use a product over all training data points and not a joint probability because we assume that those data points are *independent and identically distributed (i.i.d.)*.

1. When we actually implement this algorithm, instead of maximizing the conditional likelihood of the training data, we choose to maximize the log of that quantity, namely the conditional log-likelihood.

(a) Provide two (2) potential reasons why we might want to do that. **[5 pts]**

(b) Explain why the solution we obtain by maximizing the log of a function is the same as the solution we obtain by maximizing the function itself. **[5 pts]**

2. In order to solve the optimization problem of the training phase we need to use some numerical solver. Most solvers require that we provide a function that computes the objective function value given some weights (i.e. the quantity within the "arg max" operator) and the gradient of that objective function (i.e. its first derivatives) and they take care of solving the problem for us. Some solvers usually also require the Hessian of the objective function (i.e. its second derivatives). So, in order to implement the algorithm we need to derive those functions.

   (a) Derive the conditional log-likelihood function for the multi-class logistic regression model. You may call that function $l(\mathrm{W})$. **[10 pts]**

   (b) Derive the gradient of that function with respect to the weight vector of class c. That is, derive the value of $\nabla_{\boldsymbol{w}_c} l(\mathrm{W})$. You may call the gradient $\boldsymbol{g}_c(\mathrm{W})$. **[10 pts]**
   Note: The gradient of a function $f(\boldsymbol{x})$ with respect to vector $\boldsymbol{x}$ is itself a vector, whose i^{th} entry is defined as $\frac{\partial f(\boldsymbol{x})}{\partial x_i}$, where x_i is the i^{th} element of vector $\boldsymbol{x}$.

   (c) Derive the Hessian sub-matrix of that function with respect to the weight vector of class c and the weight vector of class c'. You may call this sub-matrix $\mathrm{H}_{c,c'}(\mathrm{W})$. **[10 pts]**
   Note: The Hessian of a function $f(\boldsymbol{x})$ with respect to vector $\boldsymbol{x}$ is a matrix, whose $\{i,j\}^{\text{th}}$ entry is defined as $\frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j}$. In this case, we are asking for a sub-matrix of the Hessian of the conditional log-likelihood function, taken with respect to weight vectors of only two classes. The $\{i,j\}^{\text{th}}$ entry of the sub-matrix you need to derive is defined as $\frac{\partial^2 l(\mathrm{W})}{\partial w_{ci} \partial w_{c'j}}$.

Problem 3: Implementing Naive Bayes and Logistic Regression [Adona - 50 pts]

In this question you will code two of the classification algorithms covered in class: *Naive Bayes* and *Logistic Regression*. Remember to follow the detailed coding and submission instructions at the top of this file!

- **Data:** All questions will use the following datastructures:
  - *XTrain* $\in \mathbb{R}^{n \times f}$ is a matrix of training data, where each row is a training point, and each column is a feature.
  - *XTest* $\in \mathbb{R}^{m \times f}$ is a matrix of test data, where each row is a test point, and each column is a feature.
  - *yTrain* $\in \{1, ..., c\}^{n \times 1}$ is a vector of training labels
  - *yTest* $\in \{1, ..., c\}^{m \times 1}$ is a (hidden) vector of test labels.
- **SUBMISSION CHECKLIST**
  - Submission executes on our machines in less than 20 minutes.
  - Submission is smaller than 1MB.
  - Submission is a `.tar` file.
  - Submission returns matrices of the *exact* dimension specified.

Logspace Arithmetic [7 pts]

When working with very small and very large numbers (such as probabilities), it is useful to work in *logspace* to avoid numerical precision issues. In logspace, we keep track of the logs of numbers, instead of the numbers themselves. (We generally use natural logs for this). For example, if $p(x)$ and $p(y)$ are probability values, instead of storing $p(x)$ and $p(y)$ and computing $p(x) * p(y)$, we work in log space by storing $\log p(x), \log p(y), \log[p(x) * p(y)]$, where $\log[p(x) * p(y)]$ is computed as $\log p(x) + \log p(y)$.

The challenge is to add and multiply these numbers *while remaining in logspace*, without exponentiating. Note that if we exponentiate our numbers at any point in the calculation it completely defeats the purpose of working in log space. Hint: Alex Smola has an excellent post on his blog about this topic.

1. **Logspace Multiplication [2 pts]**
   Complete the function logProd(x) which takes as input a vector of numbers in logspace (i.e., $x_i = \log p_i$), and returns the product of these numbers in logspace – i.e., logProd(x) $= \log \prod_i p_i$.

2. **Logspace Addition [5 pts]**
   Complete the function logSum(x) which takes as input a vector of numbers in logspace (i.e., $x_i = \log p_i$), and returns the sum of these numbers in logspace – i.e., logSum(x) $= \log \sum_i p_i$.

Naive Bayes [12 pts]

The dataset for this question can be downloaded at https://autolab.cs.cmu.edu/10701-f14/attachments/view/230. This is a real dataset called *Iris* from the UCI machine learning repository.

In this question you will implement the Gaussian Naive Bayes Classification algorithm. As a reminder, in the Naive Bayes algorithm we calculate $p(c|f) \propto p(f|c)p(c) = p(c) \prod_i p(f_i|c)$. In Gaussian Naive Bayes we learn a one-dimensional Gaussian for each feature in each class, i.e. $p(f_i|c) = N(f_i; \mu_{i,c}, \sigma^2_{i,c})$, where $\mu_{i,c}$ is the mean of feature f_i for those instances in class c, and $\sigma^2_{i,c}$ is the variance of feature f_i for instances in class c. You can (and should) test your implementation locally using the *XTrainIris* and *yTrainIris* data provided.

1. **Prior [2 pts]**
   Complete the function [p] = NBprior(yTrain). p is a $c \times 1$ vector where p_i is the prior probability of class i.

2. **Parameter Learning [5 pts]**
   Complete the function [M,V] = NBparameters(XTrain, yTrain). M is an $c \times f$ matrix where $M_{i,j}$ is the conditional mean of feature j given class i. V is an $c \times f$ matrix where $V_{i,j}$ is the conditional variance of feature j given class i.

3. **Naive Bayes Classifier [5 pts]**
   Complete the function [t] = NBclassify(XTrain, XTest, yTrain). t is a $m \times 1$ vector of predicted class values, where t_i is the predicted class for the ith row of *XTest*.

Binary Logistic Regression [17 pts]

In the following two problems, we will implement Binary and Multi-class Logistic Regression. Although Multi-class LR is a direct generalization of the binary case, the two algorithms are often formalized slightly differently (e.g. the two binary case only has one vector $w \in f \times 1$ while the equivalent "multi-class" binary implementation has a matrix $Win2 \times f$). The two formulations are equivalent, but each is more natural to manipulate in its specific context. Binary LR is slightly easier to derive and implement, so we will start here. As you progress to Multi-class LR, try to notice the parallels - this will minimize work and maximize intuition :).

NOTE: If you're feeling adventurous, you can instead start by directly implementing multi-class LR. However, you will then have to write a wrapper function for each binary method converting between the two representations (e.g. converting W to w). This should be less work if you're experience with ML, but is not recommended if this is your first time implementing these algorithms.

The dataset for this question can be downloaded at https://autolab.cs.cmu.edu/10701-f14/attachments/view/230. This is another real dataset called the *Wisconsin Breast Cancer Database*, also from UCI.

You will train the Logistic Regression Classifier using numerical optimization of the Maximum Likelihood Estimator (MLE). Since we haven't covered numerical optimization techniques in class yet, we have provided you with helper code, implementing a very rudimentary version of gradient descent. Feel free to check it out!

1. **Sigmoid Probability [2 pts]**
   Complete the function [p] = LRprob(x, w). x is a single training example $(1 \times f)$, w is a $f \times 1$ weights vector, and $p = p(y|x)$ is a 2×1 vector of class probabilities.

2. **Conditional Log Likelihood [5pts]**
   Complete the function [logl] = LRlogLikelihood(X, y, w). *logl* is the scalar conditional log likelihood of the data, $(y|X)$, under the model with parameters w.

3. **Gradient of the Conditional Log Likelihood [5pts]**
   Complete the function [grad] = LRgradient(X, y, w). *grad* is a $f \times 1$ vector representing the gradient of the conditional log likelihood with respect to w.

4. **Logistic Regression Classifier [5pts]**
   Complete the function [cls] = LRclassify(XTrain, yTrain, XTest). *cls* is a $m \times 1$ vector representing the predicted test class labels.

Multiclass Logistic Regression [14 pts]

To test the MultiClass LR Classifier we will again use the *Iris* dataset we used for Naive Bayes above. As always, you can and should test your implementation locally using $XTrainIris$ and $YTrainIris$ data provided.

1. **Probability [2 pts]**
   Complete the function [p] = mLRprob(x, W). x is a single training example $(1 \times f)$, W is a $c \times f$ weights *matrix* where each row c is the weights vector w_c, and $p = p(y|x)$ is a $c \times 1$ vector of class probabilities.

2. **Conditional Log Likelihood [4pts]**
   Complete the function [logl] = mLRlogLikelihood(X, y, W). As before, *logl* is the scalar conditional log likelihood of the data, $(y|X)$, under the model with parameters W.

3. **Gradient of the Conditional Log Likelihood [4pts]**
   Complete the function [grad] = mLRgradient(X, y, W). *grad* is a $c \times f$ *matrix* representing the gradient of the conditional log likelihood with respect to W.

4. **Logistic Regression Classifier [4pts]**
   Complete the function [cls] = mLRclassify(XTrain, yTrain, XTest). As before, *cls* is a $m \times 1$ vector representing the predicted test class labels.