Homework 3
Regression, Gaussian processes, and Boosting

CMU 10-701: Machine Learning (Fall 2014)
https://piazza.com/cmu/fall2014/1070115781/home
OUT: Oct 31, 2014
DUE: Nov 12, 11:59 PM

START HERE: Instructions

- **Late days:** The homework is due Wesnesday Nov 12th, 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: Gaussian processes [Abu - 40 pts]

Background: The goal of this problem is to provide a better intuition and understanding about Gaussian processes and regression. First, we will lay out some notation. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a collection of n points where each $x_i \in \mathbb{R}$ (we assume the points are one-dimensional in this problem for simplicity, but it is straightforward to extend this to multiple dimensions). If $m : \mathbb{R} \mapsto \mathbb{R}$ is a function, then we denote $m(X)$ as the vector where the i^{th} element is given by $m(x_i)$. If $X' = \{x'_1, \ldots, x'_n\}$ is another collection of points in $\mathbb{R}$, and $k : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$ is a function that takes two arguments, then we denote $k(X, X')$ as the matrix where the element at position (i, j) is given by $k(x_i, x'_j)$. Intuitively, you can think of k as a kernel function.

The Gaussian process (GP) is a distribution over *functions*. It is fully characterized by two parameters: a *mean function* $m : \mathbb{R} \mapsto \mathbb{R}$, and a *covariance function* $k : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$. If a function f is distributed according to a GP, we write:

$$f \sim \mathcal{GP}(m, k).$$

The GP is defined such that for any finite collection of points $X = \{x_1, \ldots, x_n\}$, the function evaluated at those points is distributed according to a multivariate normal:

$$f(X) \sim \mathcal{N}(m(X), k(X, X)).$$

The following is a *non-exhaustive* list of example covariance functions:

- linear: $k(x, x') = x \cdot x'$
- polynomial: $k(x, x') = (x \cdot x')^d$
- squared exponential: $k(x, x') = \exp\left\{-\frac{1}{2\lambda^2}(x - x')^2\right\}$
- exponential: $k(x, x') = \exp\left\{-\frac{1}{\lambda}|x - x'|\right\}$
- periodic: $k(x, x') = \exp\left\{-\frac{2}{\lambda^2}\sin\left(\frac{1}{2}|x - x'|\right)^2\right\}$
- rational quadratic: $k(x, x') = (1 + (x - x')^2)^{-\alpha}$

Model: For this problem, we will work with the following regression model:

$$\begin{aligned} f &\sim \mathcal{GP}(\mathbf{0}, k), \\ y_i &\sim \mathcal{N}(f(x_i), \sigma^2) \text{ i.i.d. for } i = 1, \ldots, n. \end{aligned}$$

In words, the function f is drawn from a GP prior with a zero mean function and covariance function k. The output points y_i are set to $f(x_i)$ plus Gaussian noise with variance σ^2. Let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$. By the definition of the GP, we can re-write this as:

$$f(X) \sim \mathcal{N}(\mathbf{0}, k(X, X)), \tag{1}$$
$$Y \sim \mathcal{N}(f(X), \sigma^2 I). \tag{2}$$

a. [**9 pts**] Here, we will visualize the samples from a GP. Suppose we want to plot the function from $x = -10$ to 10. To plot any function in code, we can discretize the x-axis $\{-10, -9.98, -9.96, \ldots, 9.98, 10\}$ and compute the function at every point. Let this discretized sequence of points be X. We can compute $f(X)$ and Y by sampling from equations (1) and (2). You can use the function `mvnrnd` in Matlab or `multivariate_normal` in NumPy. However, you are free to use any programming language. **Add your code to the tar file for ALL parts to this question. Otherwise, you will not receive full credit.**

Select **three** covariance functions, such as from the list given above. In part **a.**, you will create a figure for each covariance function, for a total of three figures, using the following steps:

i. For each covariance function, generate **three** samples of $f(X)$ and plot them in a single figure, where $X = \{-10, -9.98, -9.96, \dots, 9.98, 10\}$. You should have a total of **nine** curves, three in each figure.

ii. In each figure, plot the mean function (zero in this case).

iii. The point-wise variance $\mathrm{Var}[f(x)]$ is the variance of $f(x)$ at a single point x. By the definition of the GP, the function evaluated at a single point $f(x)$ is a *univariate* normal, with mean $m(x)$ and covariance $k(x, x)$. For a sequence of points $X = \{x_1, \dots, x_n\}$, the point-wise variance is given by $k(x_1, x_1), \dots, k(x_n, x_n)$, which is identical to the diagonal of the matrix $k(X, X)$.

In each figure, draw a confidence band: Lightly-shade the regions around the mean function $\pm$**1 standard deviation**, computed from the point-wise variance of $f(X)$. Do not estimate the standard deviation empirically from your samples. Use the true point-wise standard deviation. See Matlab function `fill` and matplotlib's `fill_between`. Your submission for **a.** should have three figures (**and no more**), each with three functions, a mean function, and a confidence band.

b. [**4 pts**] Select **one** covariance function. Now, sample and plot Y for three different values of the Gaussian noise parameter σ^2. Describe the relationship between the noise parameter and the outputs Y.

c. [**9 pts**] In **a.** and **b.**, you visualized the prior. Now, we want to fit the model to observed data and visualize the posterior. To do so, we need to derive a useful result about the multivariate normal distribution.

Let x be a vector of length n that is distributed according to a multivariate normal with mean $\mu \in \mathbb{R}^n$ and covariance matrix $\Sigma \in \mathbb{R}^{n\times n}$. We can subdivide x into two subvectors, $x_1 \in \mathbb{R}^k$ and $x_2 \in \mathbb{R}^{n-k}$, and re-write $x \sim \mathcal{N}(\mu, \Sigma)$ in the following form:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right),$$

where μ_1 is the subvector of μ containing the first k elements, μ_2 is the subvector of μ containing the last $n-k$ elements, Σ_{11} is the top-left $k \times k$ submatrix of Σ, Σ_{12} is the top-right $k \times (n-k)$ submatrix of Σ, Σ_{21} is the bottom-left $(n-k) \times k$ submatrix of Σ, and Σ_{22} is the bottom-right $(n-k) \times (n-k)$ submatrix of Σ. Derive the conditional distribution $p(x_1|x_2)$.

Hint: Write out the joint probability $p(x_1|x_2) \propto p(x_1, x_2)$.

d. [**4 pts**] Suppose we observe the following five points: $(0.5, -1)$, $(1, 1)$, $(2, 3)$, $(2.5, 1.5)$, and $(3, 0)$. Thus, we let $X_* = \{0.5, 1, 2, 2.5, 3\}$ and $Y_* = \{-1, 1, 3, 1.5, 0\}$. These can be appended to X and Y in equations (1) and (2) to obtain:

$$\begin{bmatrix} f(X) \\ f(X_*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k(X,X) & k(X,X_*) \\ k(X_*,X) & k(X_*,X_*) \end{bmatrix}\right),$$
$$\begin{bmatrix} Y \\ Y_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} f(X) \\ f(X_*) \end{bmatrix}, \sigma^2 I\right).$$

Since $Y_* = f(X_*) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, we can write:

$$\begin{bmatrix} f(X) \\ Y_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k(X,X) & k(X,X_*) \\ k(X_*,X) & k(X_*,X_*) + \sigma^2 I \end{bmatrix}\right),$$

Use your result from **c.** to show that the distribution $p(f(X)|Y_*)$ is given by:

$$\begin{aligned} f(X)|Y_* \sim \mathcal{N}(&k(X,X_*)(k(X_*,X_*) + \sigma^2 I)^{-1} Y_*, \\ &k(X,X) - k(X,X_*)(k(X_*,X_*) + \sigma^2 I)^{-1} k(X_*,X)). \end{aligned} \tag{3}$$

e. **[9 pts]** Repeat the steps in **a.**, except your functions should be sampled from $p(f(X)|Y_*)$. Add the **training points** to your figures. Here, the point-wise variance is given by the diagonal of the covariance matrix in equation (3). Implementation tip: In almost all scenarios, you do not need to directly compute the inverse of a matrix. It is much more efficient and numerically stable to use Matlab's `mldivide`, `linsolve`, or SciPy's `linalg.solve`.

f. **[5 pts]** Select the **squared exponential** covariance function. Sample and plot $f(X)|Y_*$ for three different values of the width parameter λ^2, along with *both* the mean function and confidence band. You should have three figures for this subproblem. How does this parameter affect bias in this model? How does it affect variance?

Problem 2: Regression [Zichao - 30pts]

2.1 Why Lasso Works

Lasso is a form of regularized linear regression, where the L1 norm of the parameter vector is penalized. It is used in an attempt to get a sparse parameter vector where features of little "importance" are assigned zero weight. But why does lasso encourage sparse parameters? For this question, you are going to examine this.

Let X denotes an $n \times d$ matrix where rows are training points, y denotes an $n \times 1$ vector of corresponding output values, β denotes a $d \times 1$ parameter vector and $\beta^\star$ denotes the optimal parameter vector. To make the analysis easier we will consider the special case where the training data is *whitened* (i.e., $X^\top X = I$). For lasso regression, the optimal parameter vector is given by

$$\beta^\star = \operatorname*{argmin}_{\beta} \frac{1}{2}||y - X\beta||^2 + \lambda\|\beta\|_1,$$

where $\lambda > 0$.

1. **[3 pts]** Show that whitening the training data nicely decouples the features, making $\beta_i^\star$ determined by the i^{th} feature and the output regardless of other features. To show this, write $J_\lambda(\beta)$ in the form

$$J_\lambda(\beta) = g(y) + \sum_{i=1}^{d} f(X_{\cdot i}, y, \beta_i, \lambda), \tag{4}$$

   where $X_{\cdot i}$ is the i^{th} column of X.

2. **[3 pts]** Assume that $\beta_i^* > 0$, what is the value of β_i^* in this case?

3. **[3 pts]** Assume that $\beta_i^* < 0$, what is the value of β_i^* in this case?

4. **[3 pts]** From 2 and 3, what is the condition for β_i^* to be 0? How can you interpret that condition?

5. **[3pt]**Now consider ridge regression where the regularization term is replaced by $\frac{1}{2}\lambda||\beta||_2^2$. What is the condition for $\beta_i^* = 0$? How does it differ from the condition you obtained in 4.

2.2 Bayesian regression and Gaussian process

In this part, we are going to examine the relationship between Bayesian regression and Gaussian process. Let the input for training point i be x_i which is a $d \times 1$ vector, we introduce a function $\phi(x)$ which maps a d-dimensional input vector x into a D dimensional features space. Let $\Phi(X)$, which is $D \times n$, be the aggregation of columns $\phi(x)$ for all training points. Now the regression model is

$$f(x) = \phi(x)^\top w, \tag{5}$$

where the wight vector w has length D. We want to use Bayesian approach to do regression. We assume that the observed values y differ from the function values $f(x)$ by additive noise

$$y = f(x) + \epsilon, \tag{6}$$

where the noise follows i.i.d. Gaussian distribution with zeros mean and variance σ_n^2, $\epsilon \sim N(0, \sigma_n^2)$. Let Y denote the vector of all observed values corresponding to the training points.

Further, we put a zeros mean Gaussian prior with covariance matrix Σ_p on the weights w, $w \sim N(0, \Sigma_p)$. For simplicity, we assume $\Sigma_p = \sigma_0^2 I$.

1. **[6 pts]** Given the training data X, Y, we want to derive the predictive distribution $p(f_\star|X_\star, X, y)$, where $X_\star$ is the predictive input and $f_\star = f(X_\star)$. Denote $\Phi = \Phi(X), \Phi_\star = \Phi(X_\star)$ in the derivation.
   (a) Inference in Bayesian linear model is based on posterior distribution over weights, we first want to compute the posterior distribution of weights using Bayes rule
   $$p(w|Y, X) = \frac{p(Y|X, w)p(w)}{p(Y|X)}, \tag{7}$$
   Derive the posterior distribution. (Hint: Since $p(Y|X, w)$ and $p(Y|X)$ are both Gaussian, $p(w|Y, x)$ will be Gaussian as well. You can assume this and only need to work out its mean and covariance. You can ignore the constant terms without w in it, such as $p(Y|X)$, to simplify the analysis.)
   (b) Now we can get the predictive distribution as
   $$p(f_\star|X_\star, X, Y) = \int p(f_\star|X_\star, w)p(w|X, Y)dw \tag{8}$$
   Again, you can assume $p(f_\star|X_\star, X, Y)$ is Gaussian. Derive its mean and covariance.

2. **[3 pts]** Now we want to examine the regression from Gaussian process perspective. Let the kernel function be $k(x, x') = \sigma_0^2 \phi(x)^\top \phi(x')$. We assume $f(x) \sim GP(0, k(x, x'))$ and the output y still follows (6). Derive the predictive distribution $p(f_\star|X_\star, X, Y)$ given the training data X, Y and predictive input $X_\star$. Actually, Problem 1.d already gives you the results, you can only plug in the kernel function and get the predictive distribution.

3. **[3 pts]** Show 1 and 2 are equivalent, you only need to show that mean of the Gaussian distribution are equivalent, the equivalence of the variance is not required.

4. **[3 pts]** When $D > n$, which form would you use in prediction? How about $D < n$?

Problem 3: Adaboost with Decision Stumps [Adona - 40pts]

In this question you will implement the *Adaboost algorithm with Decision Stumps.* Remember to follow the detailed coding and submission instructions at the top of this file!

- **Data:** All questions will use the following datastructures:
  - $X \in \mathbb{R}^{n\times 2}$ is a matrix of 2D data, where each row is a data point, and each column is a feature.
  - $Y \in \{-1, +1\}^{n\times 1}$ is a vector of training labels
  - $D \in [0, 1]^{n\times 1}$ is a vector of data point weights
- **SUBMISSION CHECKLIST**
  - Submission executes on our machines in less than **10 minutes**.
  - Submission is smaller than 30MB.
  - Submission is a `.tar` file.
  - Submission returns matrices of the *exact* dimension specified.

Background: Boosting is a strategy for constructing a powerful classifier as a carefully weighted sum of simple, *weak* classifiers, each of which is good at classifying *different parts of the input space.* The weak classifiers are trained to be good at different parts of the input space by being presented with different *re-weighted* versions of the same dataset. Ideally, a boosted classifier combines the low-variance of its weak classifiers with the low bias resulting from taking arbitrary linear combinations of such weak classifiers, resulting in a high-accuracy classifier which is robust to overfitting. It's a really ingenious theory, let's see how it works in practice!

Formally, we construct an (Ada)boosted classifier H over T training stages. At each stage $t \in 1..T$, we train a weak classifier h_t on the weighted dataset (X, Y) with weights D_t, (X, Y, D_t). Based on h_t's *weighted error rate*, ϵ_t, we calculate h_t's associated multiplicative factor $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$. α_t intuitively captures our confidence in h_t's judgement, and is thus big when ϵ_t is close to zero or one, and small when ϵ_t is close to .5 (random guess). Also based on the confidence α_t and the predictions of the weak classifier h_t we update our data point weights to emphasize points which we mis-classified, and de-emphasize points which we got correctly: $D_{t+1}(i) \propto D_t(i)\exp(-\alpha_t Y(i) h_t(X(i,:)))$. Finally, we put it all together by defining the boosted classifier $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$.

As you probably have noticed, most of the calculations above are on the weighted dataset (X, Y, D_t). Let's look closer at the intuition behind a weighted dataset, to understand how to perform calculations with it. Weighting a dataset (X, Y) with weights D corresponds to implicitly building a new dataset (X', Y') which we obtain by re-drawing the elements of (X, Y) with probability D. There are infinitely many ways to re-draw (X', Y'), but on average any particular instantiation of (X', Y') has D_i versions of (x_i, y_i). An equivalent way of looking at re-weighting is that we are changing the ground truth probability distribution P, from which (X, Y) are drawn, to a new distribution P', from which (X', Y') are drawn. Throughout the problem, whenever we use the weighted dataset (X, Y, D), we are implicitly using a dataset (X', Y') drawn from the underlying distribution P'. The main way we use (X, Y, D) is to compute its empirical probability distribution, in other words to compute $\hat{P}'$, the empirical estimate of P'. What is $\hat{P}'(x)$ equal to? $\hat{P}'(x) = \sum_{j=1}^{n'} \delta(x'_j = x) = \sum_{i=1}^{n} D_i \delta(x_i = x) =$ the weighted empirical counts! We will use $\hat{P}'$, the weighted empirical counts, everywhere below: to compute the weighted information gains in 1, to compute the weighted error of the best stump in 2, etc.

Finally, the last decision to take is which weak classifiers to use. The weak classifiers could be any type of classifier: Naive Bayes, logistic regression, SVM, decision trees, etc. In this problem, we will use *Decision Stumps.*

Decision Stumps [22]

Decisions stumps are perhaps the most commonly used *weak classifier* in boosting methods. A decision stump is a model consisting of a one-level decision tree. We can describe a decision stump via three values (f, x, o): the feature $f \in \{1, .., k\}$ along which we are splitting; the value x which we use as our splitting threshold ($\leq x$ or $> x$); and the orientation, $o \in \{-1, +1\}$, where the classifier assigns the label o to elements $\leq x$, and label $-o$ to elements $> x$. Given a dataset with k features (in this problem we will use $k = 2$), we can generate up to $(n-1) \times k$ different decision stumps by iterating through the features: for each feature, we sort the datapoints X along that feature, then build up to $(n-1)$ decision stumps by splitting through the midpoint between each two consecutive *distinct* points. NOTE: As not all points may be distinct, the total number of stumps may in practice be smaller than $(n-1) \times k$.

At any iteration t of the Adaboost algorithm, we are given a weighted dataset (X, Y, D_t) and are asked to identify the *"best"* decision stump h_t. In this problem, we measure "best" via the notion of information gain: the information gain of splitting along feature i on threshold x is given by the decrease in entropy $\mathcal{H}$ between the Bernoulli distributions of Y and of $Y \mid X_i, x$: $I(Y, X_i, x) = \mathcal{H}(Y) - \mathcal{H}(Y \mid X_i, x)$. Notice that again, we are using the weighted dataset (X, Y, D), or, implicitly, the modified dataset (X', Y'). For this reason, all entropy calculations above are over the modified distribution P', and thus use the *weighted empirical counts* defined above. Also, if one of the empirical counts ends up being 0, we define $0\log(0) = 0$. To put it all together: at each iteration t, we identify all possible decision stumps, compute for each decision stump its information gain, and choose the decision stump with the highest information gain as our h_t.

NOTE: We will not provide you with a development dataset (X, Y). Instead, we expect you to generate your development datasets following standard ML development procedure: First, start with small ($n = 5$), hand-defined datasets for which you can manually calculate what the desired outputs should be. An example of such a dataset is provided in the course slides on Adaboost. Once you are confident that your algorithm works on your small dataset, think about what a "standard" dataset you might want to classify with Adaboost looks like. What does the decision boundary look like? How might you generate a 2-class dataset with such a decision boundary? Figure out a method, generate a random dataset (recommended: $n = 1000$), and check that your algorithm still does what you expect it to do. The simplest way to check is visually: what graphs can you plot to get a sense for what your algorithm is doing?

1. **Calculating Information Gains[15 pts]**
   Complete the function [ns, fs, xs, gains] = getWeightedInfoGainForStumps(X, Y, D). fs, xs and $gains$ are each $ns \times 1$ vectors, where ns is the number of distinct decision stumps on X. For decision stump $i \in \{1, .., ns\}$, $fs[i] \in \{1, 2\}$ is the feature along which the stump splits, $xs[i] \in \mathbb{R}$ is the splitting threshold, and $gains[i] \in \mathbb{R}_+$ is the *weighted information gain* obtained by using the decision stump.

2. **Choosing the best stump [5 pts]**
   Complete the function [s, eps] = chooseBestStump(X, Y, D, fs, xs, gains). s is a "struct" data structure encapsulating all relevant information for the best (highest info gain) decision stump: $s.f$, the feature along which the stump splits; $s.x$ the splitting threshold; and $s.o$ the stump orientation, the label the stump classifier assigns to elements $\leq s.x$. $eps \in [0, 1]$ is the *weighted error* of the resulting stump classifier s. The orientation $s.o \in \{-1, +1\}$ is chosen such that the classifier has minimal weighted error ϵ (with ties $\epsilon = 0.5$ assigned $s.o = -1$).

3. **Predicting using a stump [2 pts]**
   Complete the function [Ypred] = predictWithStump(X, s). $Ypred$ is a $n \times 1$ vector of predicted labels for data X using decision stump s.

Adaboost Training [18]

Now that we have a method for choosing the best decision stump, we can proceed with training the Adaboost classifier.

1. **Compute the multipliers α_t [2 pts]**
   Complete the function [alpha] = computeAlpha(eps). $eps \in [0, 1]$ is the *weighted error* of a decision stump s, as computed by the function chooseBestStump. $\alpha \in \mathbb{R}$ is the resulting multiplier.

2. **Compute the new dataset weights D_{t+1} [3 pts]**
   Complete the function [Dnew] = computeNewWeights(X, Y, D, alpha, s). $Dnew$ is a $n \times 1$ vector of new weights computed based on the old weights D, the new decision stump s and the associated multiplicative factor $alpha$. Note that D and $Dnew$ are both normalized: $\sum_{i=1}^{n} D(i) = \sum_{i=1}^{n} Dnew(i) = 1$.

3. **Predict using the Adaboost model H [3 pts]**
   Complete the function [H] = predictAdaBoost(X, alphas, stumps). $alphas$ is a $T \times 1$ vector of multiplicative weights, $stumps$ is a $T \times 1$ cell array of decision stumps, and H is a $n \times 1$ vector of Adaboost classification labels for the data points X.

4. **Train the Adaboost model [10 pts]**
   Finally putting everything together! Complete the function [alphas, stumps] = trainAdaboostModel(X, Y, Tmax). The function should train an Adaboost model for a maximum number of iterations $Tmax$, and stop if at any point the weighted error eps of the best decision stump is equal to 0.5. $alphas$ is a $T \times 1$ vector of multiplicative weights and $stumps$ is a $T \times 1$ cell array of decision stumps, where T is the total number of iterations.