Mini-project 1
CMPSCI 689 Spring 2015
Due: ~~Wednesday, March 4, at 11:55 pm~~
Thursday, March 5, at 4:00 pm

Guidelines

Submission. Submit ~~a *single pdf document* via moodle~~ a hardcopy at the CS main office that includes your solutions, figures and printouts of code. For readability you may attach the code printouts at the end of the solutions. There will be a "drop box" at the main office in the reception area where you should place your report. Submissions may be 48 hours late with 50% deduction. Submissions more than 48 hours after the deadline will be given zero. Late submissions should be emailed to the TA as a pdf.

Plagiarism. We might reuse problem set questions from previous years, covered by papers and webpages, we expect the students not to copy, refer to, or look at the solutions in preparing their answers. Since this is a graduate class, we expect students to want to learn and not google for answers.

Collaboration. The homework must be done individually, except where otherwise noted in the assignments. 'Individually' means each student must hand in their own answers, and each student must write their own code in the programming part of the assignment. It is acceptable, however, for students to collaborate in figuring out answers and helping each other solve the problems. We will be assuming that, as participants in a graduate course, you will be taking the responsibility to make sure you personally understand the solution to any work arising from such a collaboration.

Using other programming languages. All of the starter code is in Matlab which is what we expect you to use. You are free to use other languages such as Octave or Python with the caveat that we won't be able to answer or debug non Matlab questions.

Changelog

- 3/1/15: Updated instructions for submission and changed deadline to Thursday.

1 Decision Trees

1.a Entropy and classification error

In class we used classification error to pick the best attribute to split the data. A different criteria is to use the Entropy. Given a random variable Y with probability distribution $p(y)$ the entropy $H(Y)$ is defined as

$$H(Y) = -\sum_y p(y) \log_e p(y).$$

1. *[0.5 points]* Show that for a binary random variable $Y \in \{yes, no\}$ with $p(Y = yes) = \theta$, the entropy is
$$H(Y) = -\theta \log_e(\theta) - (1-\theta) \log_e(1-\theta).$$

2. *[0.5 points]* What is the best classification error ($\mathbb{E}[\hat{y} \neq Y]$) of any predictor for a binary variable Y with $P(Y = yes) = \theta$?

3. *[1 point]* Plot the above entropy and classification error as a function of θ on the *same* figure. Scale the maximum y-value of both the plots to 0.5 to make them comparable. You may find the Matlab commands: `figure; hold on; plot(x, y)` useful.

From the above it should be apparent that entropy and classification error are similar (up to a scale) and an alternate way to select features to split is to pick one that offers the highest information gain (or reduction in Entropy) i.e.,

$$\text{InfoGain}(X) = H(Y) - H(Y|X)$$

This is the criteria that for the ID3 algorithm in the Quinlan 1986 paper.

1.b Train a decision tree

Untergang der Titanic by Willy Stöwer, 1912

Below is a dataset of the 2201 passengers and crew aboard the RMS Titanic, which disastrously sunk on April 15th, 1912. For every combination of three variables (Class, Gender, Age), we have the counts of how many people survived and did not. We've also included rollups on individual variables for convenience.

Next to the table is a *mosaic plot*[1], which simply visualizes the counts as proportional areas.

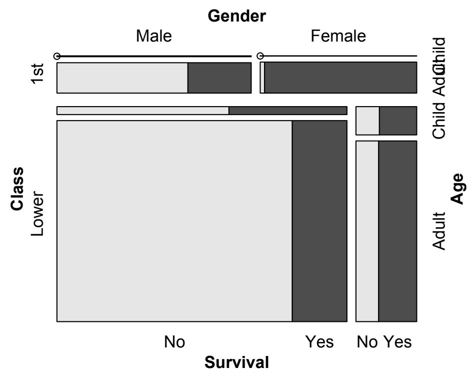
[5 points] We are interested in predicting the outcome variable Y, survival, as a function of the input features C, G, A.

Use the information gain criterion to choose which of the three features C, G or A to use at the root of the decision tree. In fact, your task here is to learn a depth 1 decision tree that uses only this root feature to classify the data (such depth-1 decision trees are often called "decision stumps").

Please show all work, including the information gain calculations for each candidate feature.

[1] http://en.wikipedia.org/wiki/Mosaic_plot

Class	Gender	Age	No	Yes	Total
1st	Male	Child	0	5	5
1st	Male	Adult	118	57	175
1st	Female	Child	0	1	1
1st	Female	Adult	4	140	144
Lower	Male	Child	35	24	59
Lower	Male	Adult	1211	281	1492
Lower	Female	Child	17	27	44
Lower	Female	Adult	105	176	281
		totals:	1490	711	2201



Class	No	Yes	Total
1st	122	203	325
Lower	1368	508	1876

Gender	No	Yes	Total
Male	1364	367	1731
Female	126	344	470

Age	No	Yes	Total
Child	52	57	109
Adult	1438	654	2092

Hint: to make information gain easier to calculate, you may wish to use this formula for conditional entropy:

$$-H(Y|X) = \sum_{x,y} p(x,y) \log_e p(y|x)$$

1.c Evaluation

1. *[1 point]* What is the accuracy rate of your decision stump (depth 1 decision tree) on the training data?

2. *[1 point]* If you grew a complete decision tree that used all three variables, what would its accuracy be over the training data? [Hint: you don't actually need to grow the tree to figure out the answer.]

1.d Decision Trees and Equivalent Boolean Expressions

[1 point] The decision tree is a function $h(C, G, A)$ that outputs a binary value. Therefore, it can be represented as a boolean logic formula.

Write a decision tree that is equivalent to the following boolean formula (i.e., a decision tree that outputs 1 when this formula is satisfied, and 0 otherwise).

$$(C \wedge \neg A \wedge \neg G) \vee (C \wedge A) \vee (\neg C \wedge G)$$

1.e Model complexity and data size

Let's think about a situation where there is a true boolean function underlying the data, so we want the decision tree to learn it. We'll use synthetic data generated by the following algorithm. To generate an $(\vec{x}, y)$ pair, first, six binary valued $x_1, \ldots, x_6$ are randomly generated, each independently with probability 0.5. This six-tuple is our $\vec{x}$. Then, to generate the corresponding y value:

$$f(\vec{x}) = x_1 \ \vee \ (\neg x_1 \ \wedge \ x_2 \ \wedge \ x_6) \quad (1)$$

$$y = f(\vec{x}) \text{ with prob } \theta \text{, else } (1 - f(\vec{x})) \quad (2)$$

So Y is a possibly corrupted version of $f(X)$, where the parameter θ controls the noisiness. $\theta = 1$ is noise-free. $\theta = 0.51$ is very noisy.

1. *[0.5 points]* What is $P(Y = 1 \mid (X_1 \vee (\neg X_1 \wedge X_2 \wedge X_6)) = 1)$?
2. *[0.5 points]* What is $P(Y = 1 \mid \neg((X_1 \vee (\neg X_1 \wedge X_2 \wedge X_6))) = 1)$?
3. *[1 point]* Does $P(Y = 1|X_2 = 1) = P(Y = 1)$? Why?
4. *[1 point]* Does $P(Y = 1|X_4 = 1) = P(Y = 1)$? Why?
5. *[1 point]* Consider learning a decision tree classifier h. Assume the learning algorithm outputs a decision tree h that exactly matches f (despite the noise in the training data, it has so much data that it still learns f correctly). Assume the training data was generated by the above process. What should h's accuracy rate be on the training data?
6. *[1 point]* Assume new test data is also generated from the same process. What should its accuracy rate be on this new test data (assuming plenty of test data)?
7. *[1 point]* Decision trees can overfit, so let's think about controlling the tree's model complexity. Instead of using pruning like we learned in lecture, here we use a maximum depth parameter.

   Assuming a very large amount of training data, what's the smallest maximum-depth setting necessary to perfectly learn the generating function f?

1.f Train/Test Experiments

Now we experimentally investigate the relationships between model complexity, training size, and classifier accuracy. Get code and test data from: `http://www-edlab.cs.umass.edu/~smaji/cmpsci689/proj/p1_code.tar.gz`. The code for this part is inside the `dt` folder.

We provide a Matlab implementation of ID3 featuring a maxdepth parameter: *train_tree(trainX, trainY, maxdepth)*. It returns an object representing the classifier, which can be viewed with *print_tree(tree)*. Classify new data via *classify_with_tree(tree, testX)*. We also provide the simulation function to generate the synthetic data: *generate_data(N, theta)*, that you can use to create training data. Finally, there is a fixed test set for all experiments (generated using $\theta = 0.9$).

See `tt1.m` for sample code to get started.

Include printouts of your code and graphs.

1. *[1 point]* For a depth=3 decision tree learner, learn classifiers for training sets size 10 and 100 (generate using $\theta = 0.9$). At each size, report training and test accuracies.
2. *[8 points]* Let's track the learning curves for simple versus complex classifiers.

   For maxdepth=1 and maxdepth=3, perform the following experiment. For each training set size $\{2^1, 2^2, \ldots, 2^{10}\}$, generate a training set, fit a tree, and record the train and test accuracies. For each (depth,trainsize) combination, average the results over 20 different simulated training sets.

   Make three learning curve plots, where the horizontal axis is training size, and vertical axis is accuracy. First, plot the two testing accuracy curves, for each maxdepth setting, on the same graph. For the second and third graphs, have one for each maxdepth setting, and on each plot its training and testing accuracy curves. Place the graphs side-by-side, with identical axis scales. It may be helpful to use a log-scale for training data size.

   Next, answer several questions with *no more than three sentences* each:
3. *[1 point]* When is the simpler model better? When is the more complex model better?
4. *[1 point]* When are train and test accuracies different? If you're experimenting in the real world and find that train and test accuracies are substantially different, what should you do?
5. *[2 points]* For a particular maxdepth, why do train and test accuracies converge to the same place? Comparing different maxdepths, why do test accuracies converge to different places? Why does it take smaller or larger amounts of data to do so?
6. *[3 points]* For maxdepths 1 and 3, repeat the same vary-the-training-size experiment with $\theta = 0.6$ for the training data. Show the graphs. Compare to the previous ones: what is the effect of noisier data?

2 Perceptron

If you are starting with this question you need to get code and test data from: `http://www-edlab.cs.umass.edu/~smaji/cmpsci689/proj/p1_code.tar.gz`. The relevant code for this part is in the `perceptron` folder.

Take a look at the file `toy.m`. The code creates a dataset of size at most 500 points using the function `data = getData('toy', opts)`. If you peek inside the `getData()` function you will see that the labels are being generated using the line $x(1) + x(2) = 0$ as the decision boundary. Moreover, points too close to the boundary are removed, i.e., the data separated by a margin of 0.1 (which is controlled by `opts.margin` parameter). There are other fields in `opts` which we will look into later.

The `data` has fields `data.train.x, data.train.y, data.test.x` and `data.test.y` corresponding to training and test, features and labels. You can visualize the data by using `plotData(x,y)`.

2.a Implement the perceptron algorithm

- *[10 points]* In Matlab, implement a function

  `w = perceptronTrain(x, y, maxiter),`

  which takes features `x`, labels `y`, and the maximum number of iterations `maxiter` as input and returns a weight vector `w`. Assume that there are n points with d features and `x` is a matrix of size $d \times n$, i.e., each *column* of `x` corresponds to a point. `y` is a matrix size $1 \times n$, each with value $\in \{+1, -1\}$.

- *[2 points]* Similarly, implement a function

  `ypred = perceptronPredict(w, x)`

  that returns the predicted labels.

On the toy dataset learn a weight vector by setting `maxiter=10`. If implemented correctly your code should return a weight vector that achieves *zero training error* and a small test error. To get full credit for this part include the following in your submission.

- Include printouts for the code for the `perceptronTrain.m` and `perceptronPredict.m`
- Visualization of predictions on the test data. Use the function `plotData(x, y, ypred)` to plot the points. It shows the misclassified points (`y` $\neq$ `ypred`) as circles.
- The learned weight vector (visualized as a line passing through the origin) on the *same figure* (you can do this by using `hold on;` and then plotting using the `plotLine(w)` function provided in the code). Make sure that the learned classifier agrees with the predictions.

2.b Effect of margin on convergence rate

The number of updates (or mistakes) the perceptron training algorithm makes is bounded by $\frac{R^2}{\delta^2}$ where δ is the margin of the dataset and R is the maximum norm of x. We will empirically verify this relationship on the toy dataset.

[3 points] Modify the `perceptronTrain` function to additionally return the number of updates made during training. Keeping `opts.noise=0`, vary `opts.margin` $\in 10^{\text{linspace}(-5,0,20)}$ (look up `linspace` on Matlab by typing: `help linspace`).

For each value of the margin, sample 10 datasets and plot the *logarithm* of the average number of updates as function of the *logarithm* of the margin (δ). In addition plot the upper bound $\log(\#updates) = \log(R^2) - \log(\delta^2)$ as function of $\log(\delta)$ on the same figure. For the toy dataset $R = \sqrt{2}$. Verify that the number of updates is always below the upper bound. Note that you might have to set the `maxiter` high enough so that the algorithm converges before it is reached.

2.c Implement the averaged perceptron algorithm

[10 points] Implement the averaged perceptron algorithm discussed in class. To get full credit implement the efficient version which maintains a running average instead of storing individual weight vectors and counts.

```
w = averagedPerceptronTrain(x, y, maxiter),
```

Note that you can use the `preceptronPredict(w, x)` for prediction since the output format is identical. Include the printout of the code in the submission, a figure showing the learned weights and predictions on the test data.

2.d Effect of noise on generalization

Noise in the training data may lead to overfitting. Here we will test generalization properties of the perceptron and averaged perceptron as a function of the magnitude of noise in the training data.

[3 points] Keeping `opts.margin=0.1`, vary `opts.noise` $\in$ linspace$(0, 1, 10)$. For each value of noise generate 10 datasets and plot the average test error of the perceptron and averaged perceptron (both of these should be run for `maxiter`=10). How does noise effect the test error for both the algorithms?

Note that when noise is large the dataset may not be separable and the perceptron training is not going to converge.

2.e Handwritten digit classification

You will learn to classify the hand written digits ″3″ from ″8″. Below is a sample of the training set:

The starter code for this part is in `digits.m`. It load the `digits.mat` file, which is a binary classification task separating handwritten images of ″3″ from ″8″. This data is a subset of the MNIST dataset. The format of the data is identical to the toy dataset. Here ″3″ has label = +1 and ″8″ has label = -1.

Each point is 784 dimensional, which is obtained by concatenating the pixel values $\in [0\ 1]$ of each digit which is a 28×28 image. It is hard to visualize the data because it lives in 784 dimensions but you can use function `montageDigits(data.train.x)` to visualize the tiny images in the training set.

1. *[3 points]* Plot the test error as a function of the number of training iterations (i.e., `maxiter=1,2,...,10`), for the perceptron and averaged perception algorithm.
2. *[1.5 points]* Visualize the learned weights for the averaged perceptron using `visualizeWeights(w)` function provided in the code. The code creates an image out of the positive and negative parts of the weights where the intensity is proportional to the magnitude. What parts of the weight vector have high positive weight? What parts have high negative weights? What parts have nearly zero weights? Illustrate with figures.