**CS446: Machine Learning** **Fall 2012**

# Problem Set 2

*Handed Out: September 17* 2014 *Due: September 25,* 2014

- Feel free to talk to other members of the class in doing the homework. I am more concerned that you learn how to solve the problem than that you demonstrate that you solved it entirely on your own. You should, however, write down your solution yourself. Please try to keep the solution brief and clear.
- Please use Piazza first if you have questions about the homework. Also feel free to send us e-mails and come to office hours.
- Please, no handwritten solutions. You will submit your solution manuscript as a single pdf file.
- A large portion of this assignment deals with programming decision trees to see them in action. While we do provide several pieces of code, you are required to try and test several decision tree algorithms by writing your own code. While we encourage discussion within and outside the class, cheating and copying code is strictly prohibited. Copied code will result in the entire assignment being discarded at the very least.
- The homework is due at 11:59 PM on the due date. We will be using Compass for collecting the homework assignments. Please submit your solution manuscript as a pdf file via Compass (`http://compass2g.illinois.edu`). Please do NOT hand in a hard copy of your write-up. Contact the TAs if you are having technical difficulties in submitting the assignment.

## 1. Learning Decision Trees – 20 points

For this question, you will manually induce a decision tree from a small data set. Table 1 shows the `Balloons` data set from the UCI Machine Learning repository that was first used for an experiment in cognitive psychology[1]. The data consists of four attributes (`Color`, `Size`, `Act`, and `Age`) and a binary label (`Inflated`). You will represent this data as decision trees using two splitting heuristics.

(a) [**7 points**] Use the ID3 heuristic to represent the data as a decision tree.

You can report the decision tree as a series of `if-then` statements in the text, or graphically, it's your choice.

Example:

```
if feature_0 = x :
  if feature_1 = y :
     class = T
  else :
     class = F
else:
  if feature ...
  ...
```

[1] You can learn more about this data set at `http://archive.ics.uci.edu/ml/datasets/Balloons`

| Color | Size | Act | Age | Inflated |
|---|---|---|---|---|
| Yellow | Small | Stretch | Adult | **T** |
| Yellow | Small | Stretch | Child | **T** |
| Yellow | Small | Dip | Adult | **T** |
| Yellow | Small | Dip | Child | **T** |
| Yellow | Large | Stretch | Adult | **T** |
| Yellow | Large | Stretch | Child | **F** |
| Yellow | Large | Dip | Adult | **F** |
| Yellow | Large | Dip | Child | **F** |
| Purple | Small | Stretch | Adult | **T** |
| Purple | Small | Stretch | Child | **F** |
| Purple | Small | Dip | Adult | **F** |
| Purple | Small | Dip | Child | **F** |
| Purple | Large | Stretch | Adult | **T** |
| Purple | Large | Stretch | Child | **F** |
| Purple | Large | Dip | Adult | **F** |
| Purple | Large | Dip | Child | **F** |

Table 1: The `Balloons` data set

(b) [**7 points**] The second heuristic uses the decrease in misclassification rate to choose an attribute to split. If, at some node, we stop growing the tree further and assign the majority label of the remaining examples to that node, then the empirical error on the training set at that node will be

$$MajorityError = min(p, 1 - p)$$

where $p$ is the fraction of examples with label $T$ and, hence, $1 - p$ is the fraction of examples with label $F$. Note that this error can be thought of as a measure of impurity of a node, just like entropy.

Redefine information gain using $MajorityError$ as the measure of impurity and use this to represent the data as a decision tree.

(c) [**6 points**] Use the first twelve examples to manually generate decision trees of maximum depth two with both the heuristics. How well do the trees perform on the remaining four examples? Report the error rate (that is, the ratio of number of errors to the number of examples) for the two algorithms.

2. **Decision Trees as Features – 80 points**

In this question, you will use a variant of the ID3 algorithm from the Weka Machine Learning toolkit[2] to train decision trees. The goal of this problem is to use the decision tree algorithm to generate features for learning a linear separator. You can use any programming language to implement the parts of the assignment that do not require Weka.

[2] `http://www.cs.waikato.ac.nz/ml/weka/`

You will use a data set that is similar to the one from the Badges Game introduced in class. This data has been cleaned so that each name now consists of two lower cased strings – the first and the last names. The labels (+1 or −1) are generated according to a new function. The new data set is available from the homepage page in a file called `data.tar.gz`. The archive contains a file called `badges.train` which has all the examples. It additionally contains a file `badges.test.blind` which you will use to generate the solutions by which you will be graded.

**five-fold cross validation:** In five fold cross validation, given 5 disjoint and roughly equal splits of the data, you train on 4 parts and evaluate the accuracy of the resulting classifier on the remaining part. Repeat this five times, once for each of the five choices of the test set. The average accuracy, $p_A$, over these five runs is a good estimate of the algorithm's performance on unseen examples. For all the different training algorithms you implement, report the accuracy on the given 5 folds of the data.

Before you run your programs on the data, you may wish to test it on a small set of examples for which you can construct the tree yourself (e.g., the `Balloons` data set from problem 1 above or the data from Mitchell, exercise 3.2) for debugging purposes.

**Reporting Module** : You will do the following for each subsection (b)-(e).

- ~~Train~~ *Implement* the classifier in question. (If it is not provided)
- *Perform* 5-fold cross-validation on the training set. ( `badges.train` )
- *Report* $p_A$, the mean predictive accuracy over the five folds.
- *Calculate and Report* the 99% confidence interval of this estimate ($p_A$) using Student's t-test. [3]
- *Train* the classifier on the complete training set. Use all the examples in `badges.train` .
- *Predict and submit* labels for the test dataset (`badges.test.blind` ) into a file named `2.X.pred` where `X` is replaced by the subsection. (e.g. `2.b.pred` , `2.c.pred` , *etc.* )
  The format here is the same as the format for `badges.train` . ~~(Here you will want to re-train against the full `badges.train` before predicting, in order to get the best performance.)~~

(a) **Feature Extraction and Instance Generation:** First, you need to extract features from the data. In the file `decision-trees.tar.gz` we have provided you with a minimum feature generator for this step (`src/FeatureGenerator.java` ). This code will generate 270 features all in $\{0, 1\}$, as follows:

The first five letters of the first and last name are each encoded in a "one-hot" representation. That is: for each position of those positions, there are 26 boolean features, only one of which will be "1", based on which letter it was, the others

are “0”.
These features are named like “firstName$X$=$i$” and “lastName$X$=$i$” for $X \in \{0...4\}$ and $i \in \{\text{a}, \text{b}, ...\text{z}\}$ .

Because the Weka implementation of ID3 only works with boolean variables, we have also coded the length of the first and last names using a kind of “one-hot” encoding, thus:
There are features named “first_len=$Y$” and “last_len=$Y$” for $Y \in \{1...5\}$ that take on values in $\{0, 1\}$. If a name is longer than 5 characters, $Y = 5$. There are no empty strings in the input.

Finally, the class label is named “Class” and takes a value in {+1,-1}.

In addition to the feature types described above, feel free to include additional features. For example, you can invent new feature types, such as features to indicate if a letter in the alphabet appears in the name or not, or features based on conjunction of two feature types given above, and so on. You can search for “STUDENTS” to find places that are appropriate to edit, or read the whole code if you like.
You need to write the features generated into the Weka Attribute-Relation file format for the steps that require Weka. For a description of the format, look at `http://weka.wikispaces.com/ARFF+%28stable+version%29` .
The main function of `FeatureGenerator` will do this.
For other steps, if you are not using Java/Weka, this file is easily converted to a .CSV file, or you can implement your own parser.

(b) **Stochastic Gradient Descent (SGD):** As a baseline, you should implement the stochastic gradient descent algorithm for the least mean squares method.
(You can program this in your programming language of choice, but we have also provided you with a stub .java file in the `decision-trees.tar.gz` archive, named `src/SGD.java`, which you can fill in.)
Here treat our class labels as being $y^{(n)} \in \{+1, -1\}$, and our *regression error* as:

$$E_r = \frac{1}{2} \sum_{n=1}^{N} \left( w^T x^{(n)} + w_0 - y^{(n)} \right)^2$$

Classification is then:

$$C(x) = \text{sign}(w^T x + w_0)$$

*Prediction Accuracy* and *Prediction Error* are then:

$$Acc_p = \frac{1}{N} \sum_{n=1}^{N} \mathbb{I} \left[ C(x^{(n)}) = y^{(n)} \right]$$

$$E_p = 1.0 - Acc_p$$

- To learn, minimize $E_r$ using the SGD algorithm.

(c) **Grow decision tree:** Use a decision tree package to train a decision tree with the ID3 algorithm using the same feature set.

We have provided you with an implementation in the `decision-trees.tar.gz` archive, named `src/Id3.java`. Look at `src/WekaTester.java` for an example of how to load input data, and train a classifier.

(d) **Grow decision stumps:**

- Repeat step (c), limiting the maximum depth of the decision tree to four. ( Output predictions to `2.d.4.pred` . )
- And again with a maximum depth of eight. (Output predictions to `2.d.8.pred` . )

(e) **Decision stumps as features:** In this part, you will experiment with using decision stumps to generate a feature set.

Using the feature set defined in step (a), train one hundred different decision stumps of maximum depth four on the entire training set. *Note: To get a hundred* **different** *decision stumps, you need to repeatedly sample* 50% *of the training set and train a decision tree on the sub-sample.* These decision stumps will be your new feature set. Make sure that you **only sample from the training set** to generate the decision stumps, otherwise you might contaminate the training set with examples from the test set and this will skew your results.

- Build a data set using the hundred decision stumps (again in the ARFF format, if you need to), as follows: for each example in the data set, the value of the $i^{th}$ feature will be the prediction of the $i^{th}$ decision stump. This will give you a new 100-dimensional feature representation for the data.
- Train a linear separator with the SGD algorithm from (b) over this data set, this is your predictor for the reporting module.
- Note that you must be able to save (even if just in-memory) the trees so that the features you use for your predictions of `badges.test.blind` are the same as those you trained on.

### 2.1. Evaluation

You should compare the five different algorithms in (b) through (e) . Remember that this is the minimum. Feel free to experiment with more parameter combinations (e.g., decision stump depth, learning rate for the SGD, and fraction of the data used to train the decision stumps), or additional feature classes you came up with.

For each algorithm, you should perform the **Reporting Module**.

Rank your algorithms in decreasing order of the performance estimate $p_A$. For each pair of consecutive algorithms in the ranking, show if the difference between the two algorithms' performances is or is not statistically significant.

*Briefly* give some thoughts on the relative performance of these algorithms and the relationship between the (e) and its constituent parts ( (b) and (d) ). What did you notice? This section need not be very long nor arduous, but should show some thought.

### 2.2. Resources

The following resources are available on the course homework page:

(a) `decision-trees.tar.gz` contains an implementation of the ID3 algorithm from the Weka Machine Learning Library. This includes functionality to limit the depth of the tree to a specified depth. There is example code that uses this class. We have also included an example feature generator that converts the raw text into the minimum features in the ARFF format. (You are encouraged to add more.)

(b) `data.tar.gz` contains the modified badges data and five splits for cross validation.

(c) `t-table.pdf`: Student's t-distribution table.

Additionally, you may find the following external resources useful:

(a) `http://www.cs.waikato.ac.nz/ml/weka/` : The Weka toolkit homepage

(b) `http://weka.wikispaces.com/ARFF+%28stable+version%29` : A description of the Attribute-Relation File Format

### 2.3. Deliverables

- **A report**
  Create a report listing down different observations from your experiments. In particular, for each algorithm, provide everything in the *Reporting Module*.

  You may provide these numbers in a table or in a graph with error bars.

  In the end, your conclusion will be that a particular algorithm (or set of algorithms) performed the best. Briefly state the assumptions that this conclusion is based on.

- **Your `*.pred` files**

- **Your code** ~~**and tree displays**~~
  Hand in all the code you wrote.

  Create a `README` file that contains your name and email address, and enough information for someone to compile your code and run it.

- Place all files, including the PDF of your report, your .pred files, your code, and the `README` in a directory called *netID*-`hw2`. Remember to exclude executables and object files. Pack the files together so that when they unpack, the *netID*-`hw2` directory is created with all your files in it. The name of the packed file should be *netID*-`hw2.zip` or *netID*-`hw2.tar.gz`.

  Of course *netID* is replaced with your UIUC NetID.

### 2.4. Grading

- Implementation of SGD algorithm [20 points]
- Implementation of decision tree and decision stumps [20 points]
- Implementation of decision stumps as features [20 points]
- Evaluation report [10 points]
- Other report elements (additional experiments, explanation of implementation and experiments, conclusions, etc.) [10 points]

Implementations will be judged by their meeting a minimum standard (yet to be decided) of accuracy on the prediction portion.

### 2.5. Competitition

We will announce in class the names of students who's predictors from (e) performed the best on the test data.
We *may* even distribute prizes.