CS446: Machine Learning **Fall 2015**

Problem Set 2

Handed Out: September 17, 2015 *Due: September 29,* 2015

- 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 the use of decision tree classifiers. Please note that the code for decision trees and several pieces of code that we provide are the **only external code you are allowed to use**. Other than that, you are required to try and test several decision tree algorithms, and program an SGD algorithm yourself, without using any external resources. While we encourage discussion within and outside the class, cheating and copying code is strictly discouraged. 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

(a) **[7 points]** You will determine the attribute that will be the root of the decision tree if you apply ID3 algorithm on the data set summarized in Table 1. Table 1 reports the information pieces that are required in determining the root attribute, by using the concept of information gain.

Attribute	Value	`Study Today` = yes	`Study Today` = no
`Holiday`	yes	5	10
	no	30	5
`Exam Tomorrow`	yes	15	1
	no	20	14

Table 1: The `Study Pattern` data set

The data set consists of two binary attributes (`Holiday`, `Exam Tomorrow`) and a binary label (`Study Today`). There are 50 instances in the data set, 35 of which are positive (`Study Today` = yes) and the remaining 15 are negative (`Study Today` = no). From Table 1, we can see that there are 5 such instances when students study during a holiday (i.e., 5 instances with `Holiday` = yes and `Study Today` = yes) and, 10 such instances when students don't study during a holiday (i.e., 10 instances with `Holiday` = yes and `Study Today` = no). Similarly, we can see that there are 15 such instances when students study before an examination (i.e., 15 instances with `Exam Tomorrow` = yes and `Study Today` = yes) and there

is a single instance when students do not study before an examination ((i.e., 1 instances with `Exam Tomorrow` = yes and `Study Today` = no).

(b) **[7 points]** For this question, you will manually induce a decision tree from a small data set. Table 2 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 a decision tree using a new splitting heuristics. This new 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.

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 2: The `Balloons` data set

You can report the decision tree as a series of `if-then` statements as the following example shows:

```
if feature_0 = x :
```

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

```
  if feature_1 = y :
    if feature_2 = z :
      class = T
    if feature_2 != z :
      class = F
  if feature_1 != y :
    class = T
if feature_0 != x :
  if feature_1 = y :
    class = T
  if feature_1 != y :
    class = F
```

(c) [**6 points**] Does ID3 find a globally optimal decision tree? Justify your answer shortly.

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.

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 (+ or −) are generated according to a new function. The new data set is available from the homework [3] page in a file called `badges.tar.gz`. The archive contains a file called `badges.modified.data.all` which has all the examples. Additionally, this archive contains five files named `badges.modified.data.fold{1-5}` with roughly equal splits of the data. You will use these to perform five-fold cross validation [4].

In the following few sections we explain the processes you will go through in the course of this problem set.

(a) **Feature Extraction and Instance Generation:** First, you need to extract features from the data. You need to generate ten feature types for each example. These feature types are Boolean in nature, and are indicators for the first five characters from the first and last names.

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

[3] `http://l2r.cs.uiuc.edu/~danr/Teaching/CS446-15/homework.html#hw2`

[4] In all the experiments in this problems set you will use the methodology of 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.

For example, consider the name "`naoki abe`" from the data set. Suppose you want to extract features corresponding to the first letter in the first name, you will have 26 Boolean features, one for each letter in the alphabet. Only the one corresponding to `n` will be 1 and the rest will be 0. This will give us $0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$, where the i^{th} element corresponds to the feature `String=First,Position=1,Character=i`. Note that the features defined earlier are actually *feature types*. In fact, you will have 260 features of the form `String=i,Position=j,Character=k`, where `i` can be `First` or `Last`, `j` can be $1, \ldots, 5$ and `k` can be $1, \ldots, 26$. In addition, you will have the length features and possibly other additional features you come up with.

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 need to write the features generated into the Weka Attribute-Relation file format so that the next steps become easier. For a description of the format, look at `http://weka.wikispaces.com/ARFF+%28stable+version%29` . See the Resources section below for sample feature file and code.

(b) Decision Trees and SGD are the two learning algorithms you will use in this data set. You will use them in multiple ways, and we now explains the ways they will be used.

(i) **Stochastic Gradient Descent (SGD):** As a baseline, you should implement the stochastic gradient descent algorithm for the least means square method. Your implementation should be generic, since you will be running this algorithm on multiple feature sets. We recommend that in your experiments you set aside a fraction of the training set to be a validation set and use it to tune the parameters (*learning rate*, *error threshold*) of the SGD algorithm.

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

(iii) **Grow decision stumps of depth 4:** Repeat step (ii), limiting the maximum depth of the decision tree to four.

(iv) **Grow decision stumps of depth 8:** Repeat step (ii), limiting the maximum depth of the decision tree to eight.

(v) **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 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 this 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 over this data set.

2.1. Evaluation

You should compare the five different algorithms – (i) simple SGD, (ii) full decision tree, (iii) decision stump of depth four, (iv) decision stump of depth eight, and (v) SGD over features derived from 100 decision stumps. 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 experiment with, (i) run five-fold cross validation on the given data set. This will determine an estimate for the algorithm's performance, p_A, on unseen examples. Note that p_A is the average accuracy over the five folds. (ii) Calculate the 99% confidence interval of this estimate using Student's t-test. You will need a table of $t_{n,\alpha}$ values to do this computation (see `t-table.pdf`).[5]

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.

2.2. Resources

The following resources are available on the course homework page:

(a) `decision-trees.rar` 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 a dummy example feature generator that converts the raw text into features in the ARFF format.

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

(c) `badges.example.arff` is an example ARFF feature file that contains features corresponding to the first and last characters in the first name.

(d) `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

[5] `http://en.wikipedia.org/wiki/Student's_t-distribution` also has a Student's t-distribution table.

2.3. What to hand in

- **A report**
  Create a report listing down different observations from your experiments. In particular, provide the following observations in an organized fashion:
  - For each algorithm in order of the ranking you created, describe the feature set and indicate the tree depth and other parameters (specially for SGD, report the *learning rate* and *error threshold*).
  - Report the value of p_A for each algorithm.
  - Provide the 99% confidence interval for this value using Student's-t distribution.

  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 code and tree displays**
  Hand in all the code you wrote. Also, for each algorithm you experimented with (except the last one on decision stumps as features), include the tree created during cross validation that had the best performance. Mention the number of correct and incorrect predictions made by the tree on the corresponding test set. The tree displays can be similar to the one shown in 1(a). You may add the tree displays to the report in a neat fashion.

  Create a `README` file that contains your name and email address, a description of which algorithms correspond to which tree files, and enough information for someone to compile your code and run it.

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

2.4. Grading

- Implementation of SGD algorithm [10 points]
- Implementation of decision tree and decision stumps. Remember that this is the **only** part where you use external commercial code. [10 points]
- Implementation of decision stumps as features [20 points]
- Evaluation report [30 points]
- Other report elements (additional experiments, explanation of implementation and experiments, conclusions, etc.) [10 points]