CIS519: Applied Machine Learning **Spring 2018**

Problem Set 1

Handed Out: January 25, 2018 *Due: February 9,* 2018

- 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 include at the top of your document the list of people you consulted with in the course of working on the homework.
- 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.
- Please use Piazza if you have questions about the homework. Also please come to the TAs recitations and to the office hours.
- Please try to keep the solution brief and clear.
- Please, no handwritten solutions. Consult the class' website if you need guidance on using Latex. You will submit your solution manuscript as a single pdf file (in addition to the package with your code; see instructions in the body of the assignment).
- The homework is due at 11:59 PM on the due date. We will be using Canvas for collecting the homework assignments. Please submit your solution manuscript as a pdf file via Canvas. Please do NOT hand in a hard copy of your write-up. Post on Piazza and Contact the TAs if you are having technical difficulties in submitting the assignment.

1. **Understanding Decision Tree Learning – 20 points**

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

`Attribute`	`Value`	`Play outside = yes`	`Play outside = no`
Sunny	yes	20	1
Sunny	no	15	14
Snow	no	10	5
Snow	yes	25	10

Table 1: The `Study Pattern` data set

The data set consists of two binary attributes (`Sunny, Snow`) and a binary **label** (`Play outside`). Notice that, even though you are not given the examples in the data set, you are given enough information to compute the information gain.

For example, you can see that there are 50 instances in the data set, 35 of which are positive (`Play outside` = yes) and the remaining 15 are negative (`Play outside`= no). From Table 1, we can see that there are 20 such instances when people play outside when it is sunny (i.e., 20 instances with `Sunny` = yes and `Play outside`= yes), 1 such instance with `Sunny` = yes and `Play outside` = no), etc. Use this information to determine the root attribute.

(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`
Blue	Small	Stretch	Adult	**F**
Blue	Small	Stretch	Child	**F**
Blue	Small	Dip	Adult	**F**
Blue	Small	Dip	Child	**F**
Blue	Large	Stretch	Adult	**F**
Blue	Large	Stretch	Child	**T**
Blue	Large	Dip	Adult	**T**
Blue	Large	Dip	Child	**T**
Red	Small	Stretch	Adult	**F**
Red	Small	Stretch	Child	**T**
Red	Small	Dip	Adult	**T**
Red	Small	Dip	Child	**T**
Red	Large	Stretch	Adult	**F**
Red	Large	Stretch	Child	**T**
Red	Large	Dip	Adult	**T**
Red	Large	Dip	Child	**T**

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 :
  if feature_1 = y :
    if feature_2 = z :
      class = T
```

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

```
      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 guarantee a globally optimal decision tree? By optimality, we mean a decision tree that perfectly fits the training data and also has a minimal depth. Justify your answer shortly.

2. **Experimenting with Decisions Trees and SGD – 80 points**

2.1. Getting Started with Scikit-learn

Before starting these programming exercises, you will need to make certain that you are working on a computer with the following particular software:

- python 3.6
- numpy (http://www.numpy.org/)
- scikit-learn (http://scikit-learn.org/stable/)

If you are using Pycharm to run your python projects, these libraries should already be installed. To make sure that you have the libraries, run the following code in the python interpreter (you should just be able to cut & paste the code):

```
from sklearn import tree
X = [[0, 0], [1, 1]]
y = [0, 1]
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X,y)
clf.predict([[2.,2.]])
```

If this code runs without error and gives you the following output:

```
array([1])
```

then everything should be configured correctly for this homework.

You can substitute the 4th line with `linear_model.SGDClassifier(...)` for training an SGD model.

Also, try to explore the method `tree.export_graphviz(...)` to print the decision tree graphically.

2.2. Decision Trees as Features

The goal of this problem is to use the decision tree algorithm to generate features for learning a linear separator. In the course of doing it you will first run several versions of the scikit-learn, Decision Tree Algorithm, the scikit-learn SGD algorithm finally, use learned Decision Trees as features for the SGD algorithm.

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 - both the first and last names. The labels (+ or -) are generated according to a new function. The new data set is available from the homework page in a file called badges.zip. The archive contains a file called badges.modified.data.train which has all the examples. Additionally, this archive contains five files named badges.modified.data.fold1-5 with roughly equal splits of the data. You will use these to perform five-fold cross validation. Furthermore, the directory also has a file named badges.modified.data.test. You have to label these names using your best performing model and submit that in the same format as the training file, i.e. each line should start with the label, followed by a space and followed by the actual name.

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.

For example, consider the name "`naoki abe`" from the data set. Suppose you want to extract features corresponding to the first letter "`n`" 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 $0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$ where the 14^{th} element corresponds to the feature `String=First,Position=1,Character=n`. 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 FirstName or LastName, j can be 1, . . . , 5 and k can be `a,b,c, ..., z`.

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, features that indicate whether the letter in the i-th place is one of {a, b, c}, length of the name and so on.

(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 use the stochastic gradient descent algorithm (`see` `sklearn.linear_model.SGDClassifier`) with `log loss`. We recommend that you use five-fold cross validation (CV) to tune the parameters (*learning rate, error threshold*) of the SGD algorithm, i.e., to select the parameter that yields the best averaged accuracy in CV.

ii. **Grow decision tree:** Use the decision tree package to train a decision tree with the `CART` algorithm available in scikit-learn, using the same feature set.

iii. **Grow decision trees of depth 4[2]:** Repeat step (ii), limiting the maximum depth of the decision tree to four.

iv. **Grow decision trees 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 use decision stumps to generate a feature set.
Using the feature set defined in step (a), train hundred different decision stumps of maximum depth eight 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.* The labels for the training data using these 100 different decision stumps will be your new feature set. Now you have to run an SGD classifier on top of these features. 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.

2.3. Evaluation

You should compare the five different algorithms – (a) simple SGD, (b) full decision tree, (c) decision stump of depth four, (d) decision stump of depth eight, and (e) 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. In addition to p_A, record also the performance of your algorithm on the training data, tr_A. This will also be the average you get over the five training 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`).[3]

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]Decision trees with limited depth are also called decision stumps.

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

2.4. What to hand in

- **A report**
  Create a report named **hw1.pdf** 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*).
  - Give 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.

  In addition, briefly comment on the results you see and on whether they match your expectations. In particular, address (i) tr_A vs. p_a, and (ii) the results you see for the different decision tree versions you used.

- **Your code and tree displays**
  Hand in all the code you wrote in a file named **hw1.py**. 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(b). 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 python files, test labels and `README` in a directory called *userID*-`hw1`. Remember to exclude executables and object files. Pack the files together so that when they unpack, the *userID*-`hw1` directory is created with all your files in it. The name of the packed file should be *userID*-`hw1.zip` or *userID*-`hw1.tar.gz`.

- **Test labels**
  Submit the test labels in the format aforementioned in a file named test_labels.txt

- The homework is due at 11:59 PM on the due date. To submit homework 1, transfer all of your files into your SEAS account on eniac. Then, from a command prompt on eniac, use the turnin command to submit all files:

```
turnin -c cis519 -p hw1 hw1.pdf test_labels.txt README hw1.py
```

Note that even students in CIS 419 should submit to the CIS 519 course, as shown in the command above (we have only one course account, CIS 519, for both sections of the course). Make certain that your submitted files are named correctly, and do not submit additional files, or place the files in a folder. You can check that your submission was received by running:

```
turnin -c cis519 -p hw1 -v
```

to list the files submitted. You should see one listing for each file submitted. The turnin command will also send you an email with a subject "cis519: submission received" to verify receipt. Subsequent submissions will overwrite earlier submissions; if you re-submit files after the due date, your submission will be counted as late regardless of whether or not the file contents changed. The official timestamp of your submission will be the most recent (i.e., latest) timestamp of ANY file you submit. Therefore, if you submit even one file late, your entire submission will be considered late.

In addition, you also have to submit the `zip/tar` solution file created above on Canvas.

- *Automatic Testing:* Once you submit your assignment, it will be picked up by the auto-feedback engine and queued for a set of automatic tests. The auto-feedback system will run these tests and then send you a second e-mail with the results once they complete. Please be aware that this is only a limited set of tests and that we cannot provide any further detail than what is given in the report. For final grading, we will run your code on entirely new data sets with a more extensive battery of tests. We will not be releasing grading information (i.e., 10 out of 10 points) until after the homework deadline. If the e-mailed report shows that your code failed the tests or that you were missing any files, you may modify your implementation or writeup and resubmit it. You may re-submit all files as many times as you would like; each time the auto-feedback system will run the tests and e-mail you the results. You can continue to use these comments about possible failures, crashes, or incorrect results to improve your code up until the homework deadline. The auto-feedback system has been tested over several semesters, so we are nearly certain that it is bug-free. Any errors it finds are most likely a result of something being incorrect in your implementation. However, in the unlikely case that the e-mailed reports indicates a bug in the testing script (e.g., your code fails a test that youre absolutely 100% certain it should pass and have double checked), please let us know via a private message on Piazza to the instructors only.

2.5. Grading

- Experimenting with the SGD algorithm [10 points]
- Experimenting with the decision tree and decision stumps. [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]