Introduction to Machine Learning
Homework 6: Trees

Due Thursday December 12 at 11:59PM

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. It's preferred that you write your answers using software that typesets mathematics (e.g. LaTeX, LyX, or MathJax via iPython), though if you need to you may scan handwritten work. You may find the minted package convenient for including source code in your LaTeX document. If you are using LyX, then the listings package tends to work better. For formatting pseudo-code, you could try algorithm2e.

1 Decision Trees

1.1 Determining Splits

Suppose we are trying to understand varieties of mushrooms. We gather some examples of mushrooms, collected in Figure 1, and decide to train a *binary* decision tree to classify them for us (two children per node, i.e., each decision chooses some variable to split on, and divides the data into two subsets). We have one real-valued feature (size) and two categorical features (spots and color). Recall that we do binary splits on a real-valued variable by finding the threshhold with the highest information gain.

y = Poisonous?	x_1 = size (real-valued)	x_2 = spots?	x_3 = color
N	1	N	White
N	5	N	White
N	2	Y	White
N	2	N	Brown
N	3	Y	Brown
N	4	N	White
N	1	N	Brown
Y	5	Y	White
Y	4	Y	Brown
Y	4	Y	Brown
Y	1	Y	White
Y	1	Y	Brown

Figure 1:

Do this problem by hand and show all of your work. Your answer must be a **binary tree** (any branch on x_1 must have an associated threshold).

1. What is the entropy of the target variable, "poisonous"?
2. What is the first attribute a decision tree trained using the entropy or information gain method we discussed in class would use to classify the data?
3. What is the information gain of this attribute?
4. Draw the full decision tree learned from this data set (no pruning, no bound on its size).

1.2 Calculating Error

Consider the data in Figure 2, where we wish to predict the target variable Y. Suppose we train a decision tree (again using information gain, and again with no pruning or bound on size).

Y	A	B	C
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	0
0	0	1	1
1	0	1	1
0	1	0	0
1	1	0	1
1	1	1	0
0	1	1	1
1	1	1	1

Figure 2:

1. What would be the *training* error of our classifier? Give as a percentage, and explain why. You can do this by inspection; there should not be significant calculations required.

2 Random Forests

The Adult Dataset (Blake and Merz, 1998) was extracted from the 1994 Census database. The prediction task here is to determine whether a person makes over 50K a year. You can refer to to `features.txt` for more information on the dataset and attributes.

What makes this dataset interesting is: out of 48842 samples, it has about 24% positive (income >50K) and 76% negative (income <= 50K) samples; some of the features have missing values (marked by '?') and there are 6 continuous and 7 categorical features. These observations make it a good candidate for a Decision Tree based approach to classification.

For this set of experiments, we have pre-processed the original dataset: removed three unused features and partitioned them into training and testing. Please use the following files: `training.csv` and `testing.csv`.

2.1 Software

We will be using Decision Tree and Random Forest implementations of the Machine Learning library `scikit-learn`. See

`http://scikit-learn.org/stable/modules/tree.html#classification`

`http://scikit-learn.org/stable/modules/ensemble.html#random-forests`

Adventurous students are encouraged to implement their own version of these classifiers.

2.2 Experiments

1. The `scikit-learn` implementation of Decision Trees does not support missing values. In this step we will use a very simple method to fill in the missing values. For continuous values - you can substitute with the `mean` or `median` value of the particular feature, for categorical values - you can substitute with the `mode`. Calculation of these statistics should be done only on training data.

2. Another limitation of `scikit-learn's DecisionTreeClassifier` and `RandomTreeClassifier` is that they do not accept non-numeric values. That is, the categorical features cannot be used as is. (The same problem would arise had you wanted to use a SVM for this classification problem.)

   One way to overcome this is to transform the feature space, making one binary valued feature out of each value of the categorical features, while keeping the numeric features intact. We call the transformation a one-hot encoding. Pseudocode goes something like this:

   ```
   INITIALIZE new_features = []
   FOR EACH feat IN original_features:
       IF feat.type == CONTINUOUS or NUMERIC:
           new_features.append(feat.name)
       IF feat.type == CATEGORICAL:
           FOR each feat.value:
               new_feat_name=feat.name + SOME_SEPARATOR + feat.value
               new_features.append(new_feat_name)
   ```

   When transforming the original data record, numeric (i.e. continuous) features remain unchanged, and each of the binary valued features that replace the categorical features is set to 1 or 0 depending on whether the original categorical feature takes the relevant value.

   Alternatively, you could use the pandas function `getdummies`.

3. Randomly assign the data points to the training set (70%) and validation set (30%) using the sklearn function `train_test_split`.

4. Full depth Decision Trees are prone to overfitting. One way to address this is via pruning, but `scikit-learn` does not support pruning. Instead, it has two parameters that you can tune: `max_depth`, which limits the depth of the decision tree, and `min_samples_leaf`, which requires that every leaf has at least this many data points.

   Generate two plots where on the X-axis you vary one of the parameters (see below) and on the y-axis you show classification accuracy. Each plot should have two lines on it, one for accuracy on `training_set` and another for accuracy on `validation_set`:

1. `max_depth = [1,2,3...30]`
2. `min_samples_leaf = [1,2,3...50]`

Save your best choices of parameters.

Create a visualization of the top 3 levels of your Decision Tree obtained using the optimal `max_depth and min_samples_leaf` found above.. In `scikit-learn` you can use `graphviz`. Refer to the tutorial for examples http://scikit-learn.org/stable/modules/tree.html. Note that you should use `conda` to install both `graphi-viz` and `python-graphviz`. Alternatively, you could use Jupyter Hub.

5. Fit a Random Forest Classifier. One of the parameters to tune the Random Forest Classifier is the number of trees in the ensemble (`n_estimators` in `scikit-learn`). As before, generate two plots with classification accuracy (Y-Axis) of `training_set` and `validation_set` vs. (X-Axis):

   1. `n_estimators = [1,2,3...50]`, with all other parameters set to default.
   2. `n_estimators = [1,2,3...50]`, with `max_depth` and `min_samples_leaf` chosen using your best results from Question 4.

6. Train your best configuration of `DecisionTreeClassifier` and `RandomForestClassifier` on the full training data, and report their performance on the test dataset.

Note that for Random Forests you don't need to keep aside a validation set to get an unbiased estimate of the test set error. This is only for Random Forests, which use bagging, not for Decision Trees. Instead, one can use the out of bag error estimate (see http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm). Note also that with ensemble methods you can add as many trees as you can afford computationally, without worrying about overfitting.