# CS273P Homework #4
## Machine Learning & Data Mining: Spring 2018
### Due: Thursday May 24th, 2018

**Write neatly (or type) and show all your work!**

Download the provided Homework 4 code, to replace / add to last week's code (several new functions have been added).

## Problem 1: Decision Trees (30 pts)

In order to reduce my email load, I decide to implement a machine learning algorithm to decide whether or not I should read an email, or simply file it away instead. To train my model, I obtain the following data set of binary-valued features about each email, including whether I know the author or not, whether the email is long or short, and whether it has any of several key words, along with my final decision about whether to read it ($y = +1$ for "read", $y = -1$ for "discard").

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $y$ |
|-------|-------|-------|-------|-------|-----|
| know author? | is long? | has 'research' | has 'grade' | has 'lottery' | $\Rightarrow$ read? |
| 0 | 0 | 1 | 1 | 0 | -1 |
| 1 | 1 | 0 | 1 | 0 | -1 |
| 0 | 1 | 1 | 1 | 1 | -1 |
| 1 | 1 | 1 | 1 | 0 | -1 |
| 0 | 1 | 0 | 0 | 0 | -1 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | -1 |

In the case of any ties, we will prefer to predict class +1.

(a) Calculate the entropy of the class variable $y$

(b) Calculate the information gain for each feature $x_i$. Which feature should I split on first?

(c) Draw the complete decision tree that will be learned from these data.

## Problem 2: Decision Trees on Kaggle (40 pts)

In this problem, we will use Kaggle in-class competition data to test decision trees on real data. You can download the data here `https://www.kaggle.com/c/uci-s2018-cs273p-hw4`. Note that this is not same site that you use for Project 1. The `X_train.txt` and `X_test.txt` data are the same, but the target variable `Y` is here classification rather than regression. You should be able to download the data and upload your predictions without registering. When uploading your predictions, create a name for you submissions as a concatenation of your last name and the **last** 4 digits of your student id.

In this problem, we will build a simple decision tree model to make predictions on the data. You can use the `treeClassify` class provided to build your decision trees.

**Note:** Kaggle competitions only let you submit a fixed number of predictions per day, $\approx 5$ in our case, so be careful. We'll use a validation split to decide what hyperparameter choices we think are most promising, and upload only one model.

(a) Load the training data, **X_train.txt** and **Y_train.txt**. There are quite a lot of data available; for the homework you can just use the first 10,000 samples. Split out a validation set as well, say samples $10001 - 20000$.

(b) Learn a decision tree classifier on the data. To avoid any potential recursion limits, specify a max depth of 50, e.g.,

```
learner = ml.dtree.treeClassify(Xt,Yt, maxDepth=50)
```

(This may take a minute or two.) Compute your model's training and validation error rates.

(c) Now, try varying the maximum depth parameter (**maxDepth**), which forces the tree to stop after at most that many levels. Test values **0, 1, ..., 15** and compare their performance (both training and test) against the full depth. Is complexity increasing or decreasing with the depth cutoff? Identify whether you think the model begins overfitting, and if so, when. If you use this parameter for complexity control, what depth would you select as best?

(d) Now, using high maximum depth ($d = 50$), use **minLeaf** to control complexity. Try values **2.∧[2:12]=[4,8,16,...,4096]**. Is complexity increasing or decreasing as **minLeaf** grows? Identify when (if) the model is starting to overfit, and what value you would use for this type of complexity control.

(e) **(Not graded)** A related control is **minParent**; how does complexity control with **minParent** compare to **minLeaf**?

(f) Our Kaggle competition measures performance using the ROC curve, specifically the AUC (area under the curve) score. Compute and plot the ROC curve for your trained model (using e.g. the **roc** member function), and the area under the curve (**auc**).

(g) Using your best complexity control value (either depth or number of leaf data), re-train a model (you may use the first 10k data, or more, as you prefer). Load the test features **X_test.txt**, and make predictions on all test points. Output your predictions in the format expected by Kaggle,

```
Ypred = learner.predictSoft( Xte )
# Now output a file with two columns, a row ID and a confidence in class 1:
np.savetxt('Yhat_dtree.txt',
   np.vstack( (np.arange(len(Ypred)) , Ypred[:,1]) ).T,
   '%d, %.2f',header='ID,Prob1',comments='',delimiter=',');
```

Upload them and report your model's performance. Compare its performance to the AUC score you estimated using the validation data.

Note the use of **predictSoft** here; while you can upload "hard" predictions (class values), your ROC score will usually be much better if you include your confidence level. Thus we typically upload our confidence that the data is class 1; this allows Kaggle to order the data by our confidence and give a smoother ROC curve (and thus usually higher area under the curve).

## Problem 3: Random Forests (30 pts)

Random Forests are bagged collections of decision trees, which select their decision nodes from randomly chosen subsets of the possible features (rather than all features). You can implement this easily in **treeClassify** using option **'nFeatures'**=$n$, where $n$ is the number of features to select from (e.g., $n = 50$ or $n = 60$ if there are 90-some features); you'll write a for-loop to build the ensemble members, and another to compute the prediction of the ensemble.

In Python, it is easy to keep a list of different learners, even of different types, for use in an ensemble predictor:

```
ensemble[i] = treeClassify(Xb,Yb,...)  # save ensemble member "i" in a cell array
# ...
ensemble[i].predict(Xv,Yv);            # find the predictions for ensemble member "i"
```

(a) Using your validation split, learn a bagged ensemble of decision trees on the training data and evaluate validation performance. (See the pseudocode from lecture slides.) For your individual learners, use little complexity control (depth cutoff 15+, minLeaf 4, etc.), since the bagging will be used to control overfitting instead. For the bootstrap process, draw the same number of data as in your training set after the validation split ($M' = M$ in the pseudocode). You may find **ml.bootstrapData()** helpful, although it is very easy to do yourself. Plot the training and validation error as a function of the number of learners you include in the ensemble, for (at least) $1, 5, 10, 25$ learners. (You may find it more computationally efficient to simply learn 25 ensemble members first, and then evaluate the results using only a few of them; this will give the same results as only learning the few that you need.)

(b) Now choose an ensemble size and build an ensemble using at least 10k training data points, make predictions on the test data, and upload to Kaggle. Report your preformance, along with your estimated AUC score on your validation data.

Note: for your ensemble's "soft" predictions, you can either use the fraction of members of your ensemble that predicted class 1, or take the average of your ensemble members' soft prediction scores (confidences). You can try either, or try both and let us know which worked better.