CM146, Winter 2020
Problem Set 1: Decision trees, Nearest neighbors
Due Jan 27, 2020 at 11:59 pm

1 Splitting Heuristic for Decision Trees [14 pts]

Recall that the ID3 algorithm iteratively grows a decision tree from the root downwards. On each iteration, the algorithm replaces one leaf node with an internal node that splits the data based on one decision attribute (or feature). In particular, the ID3 algorithm chooses the split that reduces the entropy the most, but there are other choices. For example, since our goal in the end is to have the lowest error, why not instead choose the split that reduces error the most? In this problem, we will explore one reason why reducing entropy is a better criterion.

Consider the following simple setting. Let us suppose each example is described by n boolean features: $X = \langle X_1, \ldots, X_n \rangle$, where $X_i \in \{0, 1\}$, and where $n \geq 4$. Furthermore, the target function to be learned is $f : X \to Y$, where $Y = X_1 \vee X_2 \vee X_3$. That is, $Y = 1$ if $X_1 = 1$ or $X_2 = 1$ or $X_3 = 1$, and $Y = 0$ otherwise. Suppose that your training data contains all of the 2^n possible examples, each labeled by f. For example, when $n = 4$, the data set would be

X_1	X_2	X_3	X_4	Y
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	0	1
0	0	1	0	1
1	0	1	0	1
0	1	1	0	1
1	1	1	0	1

X_1	X_2	X_3	X_4	Y
0	0	0	1	0
1	0	0	1	1
0	1	0	1	1
1	1	0	1	1
0	0	1	1	1
1	0	1	1	1
0	1	1	1	1
1	1	1	1	1

(a) **(4 pts)** How many mistakes does the best 1-leaf decision tree make over the 2^n training examples? (The 1-leaf decision tree does not split the data even once. Make sure you answer for the general case when $n \geq 4$.) **Solution:** A sample X is labeled 0 if and only if $X_1 = X_2 = X_3 = 0$. The number of such binary vectors is given by 2^{n-3} because there are two choices for each of the remaining $n - 3$ features. Since $2^n - 2^{n-3}$ is on the order of 2^n, which is much larger than 2^{n-3}, the best 1-leaf decision tree predicts 1 for every input and makes 2^{n-3} mistakes. This corresponds to making an error $2^{n-3}/2^n = 1/8^{\text{th}}$ of the time.

Common mistakes: Many people only answered this question for the specific case when $n = 4$. The question asked about the case when $n \geq 4$.

(b) **(3 pts)** Is there a split that reduces the number of mistakes by at least one? (That is, is there a decision tree with 1 internal node with fewer mistakes than your answer to part (a)?)

Parts of this assignment are adapted from course material by Andrea Danyluk (Williams), Tom Mitchell and Maria-Florina Balcan (CMU), Stuart Russell (UC Berkeley) and Jessica Wu (Harvey Mudd).

Why or why not? (Note that, as in lecture, you should restrict your attention to splits that consider a single attribute.) **Solution:** No. No matter what variable you put at the root, the error rate will remain 1/8. Splitting on X_i with $i \geq 4$ will split the data so that the proportion of ones in each leaf is 7/8. In both leaves, the tree will predict 1, so it makes the same number of errors as the single-leaf tree that always predicts 1. Splitting on X_1, X_2, or X_3 will split the data into one leaf that is contains only 1's and one leaf where the proportion of 1's is 3/4. Again, this tree will predict 1 in both leaves, so it makes the same number of mistakes as the single-leaf tree that always predicts 1.

Common mistakes: Many people suggested that the tree with a single internal node would make more mistakes than the single-leaf tree that always predicts 1. But, since the tree with an internal node can always predict 1 in both of its leaves, it should never make more mistakes (as long as we choose the predictions in each leaf greedily).

(c) **(3 pts)** What is the entropy of the output label Y for the 1-leaf decision tree (no splits at all)? **Solution:** $Y \sim Bernoulli(7/8)$. Thus, $H(X) = (1/8)\log(8) + (7/8)\log(8/7) = 0.543$ bits.

(d) **(4 pts)** Is there a split that reduces the entropy of the output Y by a non-zero amount? If so, what is it, and what is the resulting conditional entropy of Y given this split? (Again, as in lecture, you should restrict your attention to splits that consider a single attribute.) **Solution:** Yes, splitting with any of X_1 or X_2 or X_3 gives a tree with entropy $H(Y|X_i) = (1/2)[0] + 1/2[(1/4)\log(4) + (3/4)\log(4/3)] = 0.406$ bits.

2 Entropy and Information [6 pts]

The entropy of a Bernoulli (Boolean 0/1) random variable X with $P(X = 1) = q$ is given by

$$B(q) = -q \log q - (1 - q) \log(1 - q).$$

Suppose that a set S of examples contains p positive examples and n negative examples. The entropy of S is defined as $H(S) = B\left(\frac{p}{p+n}\right)$. In this problem, you should assume that the base of all logarithms is 2. That is, $\log(z) := \log_2(z)$ in this problem (as in the lectures concerning entropy).

(a) **(4 pts)** Show that $0 \leq H(S) \leq 1$ and that $H(S) = 1$ when $p = n$.

Solution: To show $0 \leq H(s) \leq 1$, the maximum (it's a concave function) occurs when $\frac{\partial B}{\partial q} = log(\frac{1-q}{q}) = 0$ this occurs when $q = \frac{1}{2}$ and plugging back into $H(s)$, we get $B = 1$. We know the function is less than 1, now check the boundaries i.e. $q = 1$, and $q = 0$.

When $p = n$, $H(s) = B(\frac{1}{2}) = 1$

(b) **(2 pts)** Based on an attribute, we split our examples into k disjoint subsets S_k, with p_k positive and n_k negative examples in each. If the ratio $\frac{p_k}{p_k+n_k}$ is the same for all k, show that the information gain of this attribute is 0.

Solution: Since $p = \sum_k p_k$ and $n = \sum_k n_k$, if $p_k/(p_k + n_k)$ is the same for all k, it must be that $p_k/(p_k + n_k) = p/(p + n) = q$ for all k. Then the entropy of S is $B\left(\frac{p}{p+n}\right)$ and the

weighted average entropy of its children S_k is

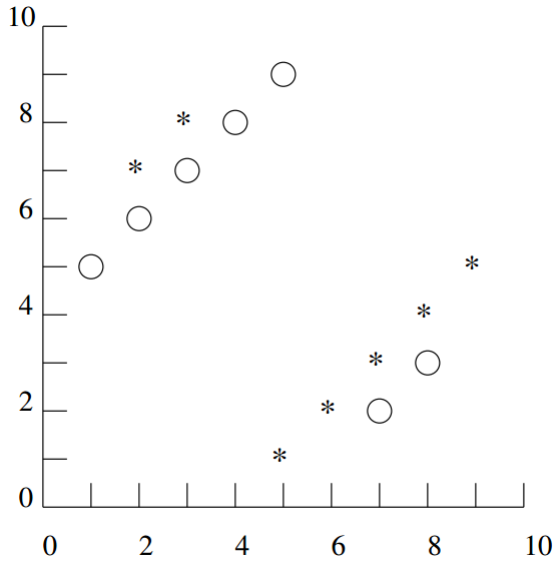
$$\sum_k \frac{p_k + n_k}{p+n} B\left(\frac{p_k}{p_k + n_k}\right) = \frac{\sum_k p_k + n_k}{p+n} B\left(\frac{p}{p+n}\right) = \frac{p+n}{p+n} B\left(\frac{p}{p+n}\right) = B\left(\frac{p}{p+n}\right)$$

so that

$$Gain = B\left(\frac{p}{p+n}\right) - B\left(\frac{p}{p+n}\right) = 0.$$

3 k-Nearest Neighbor [10 pts]

One of the problems with k-nearest neighbor learning is selecting a value for k. Say you are given the following data set. This is a binary classification task in which the instances are described by two real-valued attributes. The labels or classes of each instance are denoted as either an asterisk or a circle.



(a) **(3 pts)** What value of k minimizes training set error for this data set, and what is the resulting training set error? Why is training set error not a reasonable estimate of test set error, especially given this value of k? **Solution:** $k = 1$, assigning each training example to its given class, minimizes the training set error by fitting the data perfectly, thus achieving zero training set error.

When the model overfits the training data, it may increase training set accuracy while lowering test set accuracy. This is definitely the case for $k = 1$, as the model fits the given data perfectly but may not generalize well to new data.

(b) **(3 pts)** What value of k minimizes the leave-one-out cross-validation error for this data set, and what is the resulting error? Why is cross-validation a better measure of test set performance? **Solution:** In LOOCV, a test example is classified using a model trained on all other examples. In this problem, minimum LOOCV is achieved when only the asterisks in the top-left and the circles in the bottom-left are misclassified. This can be attained with 5-NN and 7-NN, yielding a LOOCV error of 4/14.

Cross-validation is a better measure of test set performance because it measures performance on data other than the data used to train the model.

(c) **(4 pts)** What are the LOOCV errors for the lowest and highest k for this data set? Why might using too large or too small a value of k be bad? **Solution:** Using LOOCV, $k = 1$ misclassifies all points except the left-most and right-most examples in each group (since the nearest neighbor for the middle points are one-unit away and part of the opposite class), thus achieving a LOOCV error of 10/14. $k = 13$ misclassifies every point (since once any given

point is "left out", there are 7 points of the opposite class and 6 points of the same class), thus achieving a LOOCV error of 14/14.

Using too small or too large a value for k reduces test set accuracy because the former leads to overfitting and the latter to underfitting.

4 Programming exercise : Applying decision trees [24 pts]

4.1 Visualization [4 pts]

One of the first things to do before trying any formal machine learning technique is to dive into the data. This can include looking for funny values in the data, looking for outliers, looking at the range of feature values, what features seem important, etc.

(a) **(2 pts)** Run the code (`titanic.py`) to make histograms for each feature, separating the examples by class (e.g. survival). This should produce seven plots, one for each feature, and each plot should have two overlapping histograms, with the color of the histogram indicating the class. For each feature, what trends do you observe in the data? **Solution:**

feature	trend
`Pclass`	First class had the highest survival rate, and third class had the lowest survival rate.
`Sex`	Females had a higher survival rate than males.
`Age`	Children (below age 10) had the highest survival rate, and adults (between ages 20-40) had the lowest survival rate.
`Sibsp`	Passengers with at least one travelling sibling or spouse had higher survival rates than passengers travelling alone. (The uptick at `Sibsp = 1` is probably due to the prioritization of females on the lifeboats, resulting in wives surviving their husbands – this intuition could be verified using a scatterplot.)
`Parch`	Passengers with at least one travelling parent or child had higher survival rates than passengers travelling alone. (This trend is probably due to the prioritization of mothers and children on the lifeboats.)
`Fare`	Passengers who paid more for their fare ($> \$50$) had a higher survival rate. (This trend echos the trend observed in `Pclass`.)
`Embarked`	Passengers embarking from Cherbourg had the highest survival rate.

4.2 Evaluation [20 pts]

Now, let us use `scikit-learn` to train a `DecisionTreeClassifier` on the data.

Using the predictive capabilities of the `scikit-learn` package is very simple. In fact, it can be carried out in three simple steps: initializing the model, fitting it to the training data, and predicting new values.[1]

(b) **(2 pts)** Before trying out any classifier, it is often useful to establish a *baseline*. We have implemented one simple baseline classifier, `MajorityVoteClassifier`, that always predicts the majority class from the training set. Read through the `MajorityVoteClassifier` and its usage and make sure you understand how it works.

Your goal is to implement and evaluate another baseline classifier, `RandomClassifier`, that predicts a target class according to the distribution of classes in the training data set. For example, if 60% of the examples in the training set have `Survived = 0` and 40% have `Survived = 1`, then, when applied to a test set, `RandomClassifier` should randomly predict 60% of the examples as `Survived = 0` and 40% as `Survived = 1`.

Implement the missing portions of `RandomClassifier` according to the provided specifications. Then train your `RandomClassifier` on the entire training data set, and evaluate its training error. If you implemented everything correctly, you should have an error of 0.485. **Solution:**

(c) **(2 pts)** Now that we have a baseline, train and evaluate a `DecisionTreeClassifier` (using the class from `scikit-learn` and referring to the documentation as needed). Make sure you initialize your classifier with the appropriate parameters; in particular, use the 'entropy' criterion discussed in class. What is the training error of this classifier? **Solution:** 0.014

(d) **(6 pts)** So far, we have looked only at training error, but as we learned in class, training error is a poor metric for evaluating classifiers. Let us use cross-validation instead.

Implement the missing portions of `error(...)` according to the provided specifications. You may find it helpful to use `train_test_split(...)` from `scikit-learn`. To ensure that we always get the same splits across different runs (and thus can compare the classifier results), set the `random_state` parameter to be the trial number.

Next, use your `error(...)` function to evaluate the training error and (cross-validation) test error of each of your three models. To do this, generate a random 80/20 split of the training data, train each model on the 80% fraction, evaluate the error on either the 80% or the 20% fraction, and repeat this 100 times to get an average result. What are the average training and test error of each of your classifiers on the Titanic data set? **Solution:**

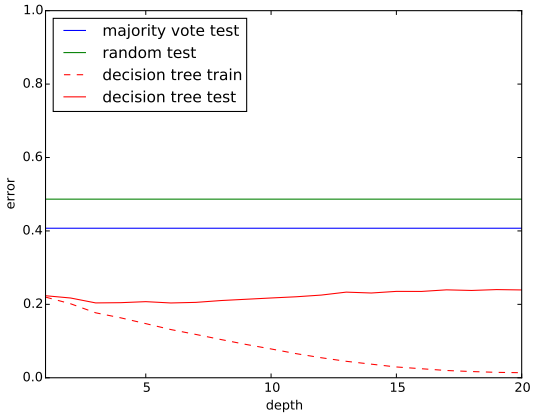
classifier	avg train error	avg test error
Majority Vote	0.404	0.407
Random	0.489	0.487
Decision Tree	0.012	0.241

Note: Errors may differ as a result of the random number generator.

[1] Note that almost all of the model techniques in `scikit-learn` share a few common named functions, once they are initialized. You can always find out more about them in the documentation for each model. These are `some-model-name.fit(...)`, `some-model-name.predict(...)`, and `some-model-name.score(...)`.

(e) **(5 pts)** One problem with decision trees is that they can *overfit* to training data, yielding complex classifiers that do not generalize well to new data. Let us see whether this is the case for the Titanic data.

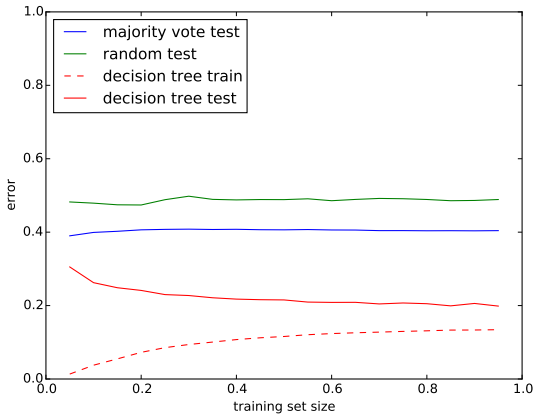
One way to prevent decision trees from overfitting is to limit their depth. Repeat your cross-validation experiments but for increasing depth limits, specifically, $1, 2, \ldots, 20$. Then plot the average training error and test error against the depth limit. (Also plot the average test error for your baseline classifiers. As the baseline classifiers are independent of the depth limit, their plots should be flat lines.) Include this plot in your writeup, making sure to label all axes and include a legend for your classifiers. What is the best depth limit to use for this data? Do you see overfitting? Justify your answers using the plot. **Solution:**



Based on the test error, the optimal depth is 6. Training error decreases with depth, but test error decreases then increases, indicating that the classifier is overfitting.

(f) **(5 pts)** Another useful tool for evaluating classifiers is *learning curves*, which show how classifier performance (e.g. error) relates to experience (e.g. amount of training data).

Run another experiment using a decision tree with the best depth limit you found above. This time, vary the amount of training data by starting with splits of 0.05 (5% of the data used for training) and working up to splits of size 0.95 (95% of the data used for training) in increments of 0.05. Then plot the decision tree training and test error against the amount of training data. (Also plot the average test error for your baseline classifiers.) Include this plot in your writeup, and provide a 1-2 sentence description of your observations. **Solution:**



The learning curve for the decision tree is indicative of a high-variance estimator. Training error increases with the number of training examples but ends at a low-level. Test error

starts high and decreases with the number of training examples but ends at a fairly high level. Crucially, with a large training set size, there is a gap between training error and test error, so adding more data will help. (Or if that is not possible, we could try alternatives such as decreasing the number of features.)

While we did not plot the training error for the baseline classifiers, the test error remains relatively stable and very high across all training set sizes. If we were to plot the training error, we would see both the training error and test error plateau and converge to an unacceptably high level, indicative of a high-bias estimator. In this case, adding more training data will not help, and we should look into generating new features or changing the model structure in some other way.