CM146, Winter 2018
Problem Set 1: Decision trees and k-Nearest Neighbors
Due Oct 29, 2018 at 11:59 pm

1 Splitting Heuristic for Decision Trees [20 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 $f : X \rightarrow Y$, where $Y = X_1 \wedge X_2 \wedge X_3$. That is, $Y = 1$ if $X_1 = 1$ and $X_2 = 1$ and $X_3 = 1$, 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	0
0	1	0	0	0
1	1	0	0	0
0	0	1	0	0
1	0	1	0	0
0	1	1	0	0
1	1	1	0	1

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

(a) **(5 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 $X_1 = X_2 = X_3 = 1$. 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 0 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.

(b) **(5 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)?) Why or why not?

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

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.

(c) **(5 pts)** What is the entropy of the output label Y for the 1-leaf decision tree (no splits at all)?

Solution: $Y \sim \mathit{Bernoulli}(7/8)$. Thus, $H(X) = (1/8)\log(8) + (7/8)\log(8/7) = 0.543$ bits.

(d) **(5 pts)** You've trained a decision tree for classifying emails as being spam or ham and the resulting tree, is getting very bad performance on both the train/test data splits. You're implementation has no bugs so what is causing the problem?

Solution: The decision tree is too shallow, as the resultant tree is getting bad performance on both train and test splits. We first need to produce a tree that can at least fit the training set, then prune to help generalization.

2 Entropy and Information [5 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 $H(S) = B\left(\frac{p}{p+n}\right)$.

(a) **(5 pts)** Based on an attribute X_j, 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

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

3 k-Nearest Neighbors and Cross-validation [15 pts]

In the following questions you will consider a k-nearest neighbor classifier using Euclidean distance metric on a binary classification task. We assign the class of the test point to be the class of the majority of the k nearest neighbors. Note that a point can be its own neighbor.

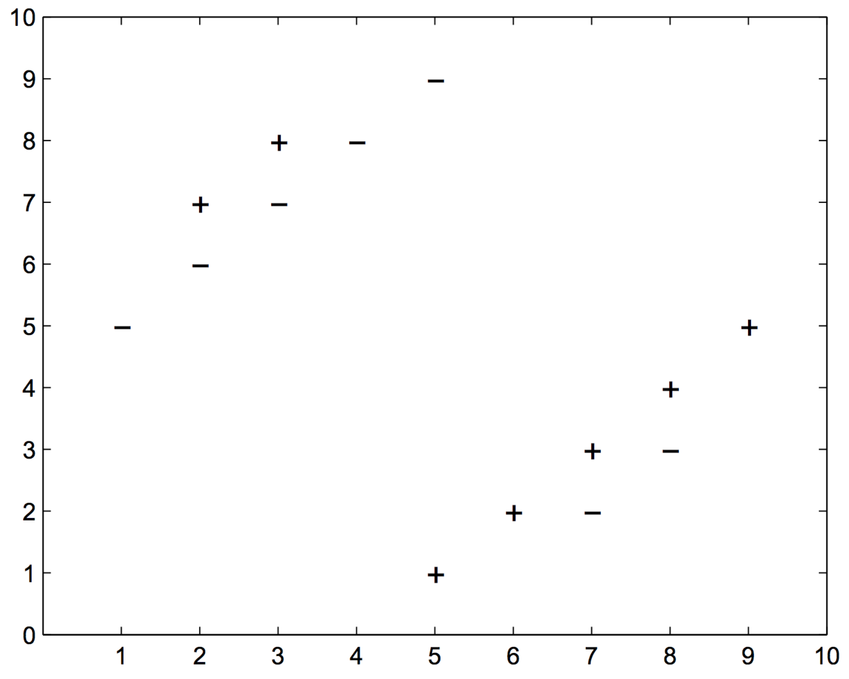


Figure 1: Dataset for KNN binary classification task.

(a) **(5 pts)** What value of k minimizes the training set error for this dataset? What is the resulting training error?

Solution: $k = 0$ or $k = 1$. Note that a point can be its own neighbor. So, k = 1 minimizes the training set error. The error is 0. Depend on your definition of k, the answer can be $k = 0$ or$k = 1$.

(b) **(5 pts)** Why might using too large values k be bad in this dataset? Why might too small values of k also be bad?

Solution: Too big k (k = 13) misclassifies every datapoint (using leave one out cross validation). Too small k leads to overfitting.

(c) **(5 pts)** What value of k minimizes leave-one-out cross-validation error for this dataset? What is the resulting error?

Solution: $k = 5$ or $k = 7$ minimizes the leave-one-out cross-validation error. The error is 4/14.

4 Programming exercise : Applying decision trees and k-nearest neighbors [60 pts]

4.1 Visualization [5 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) **(5 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 [55 pts]

Now, let us use `scikit-learn` to train a `DecisionTreeClassifier` and `KNeighborsClassifier` 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) **(5 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.

(c) **(5 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) **(5 pts)** Similar to the previous question, train and evaluate a `KNeighborsClassifier` (using the class from `scikit-learn` and referring to the documentation as needed). Use k=3, 5 and 7 as the number of neighbors and report the training error of this classifier. **Solution:** 0.201

(e) **(10 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 four models (for the `KNeighborsClassifier`, use k=5). 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?

[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(...)`.

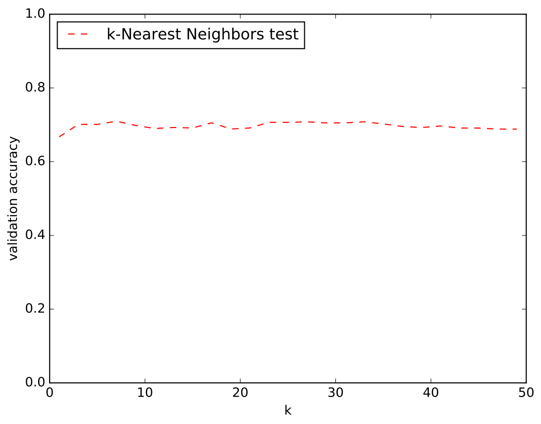
Solution:

classifier	avg train error	avg test error
Majority Vote	0.404	0.407
Random	0.489	0.487
Decision Tree	0.012	0.241
k-Nearest Neighbors	0.212	0.315

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

(f) **(10 pts)** One way to find out the best value of k for `KNeighborsClassifier` is n-fold cross validation. Find out the best value of k using 10-fold cross validation. You may find the `cross_val_score(...)` from `scikit-learn` helpful. Run 10-fold cross validation for all odd numbers ranging from 1 to 50 as the number of neighbors. Then plot the validation error against the number of neighbors, k. Include this plot in your writeup, and provide a 1-2 sentence description of your observations. What is the best value of k?

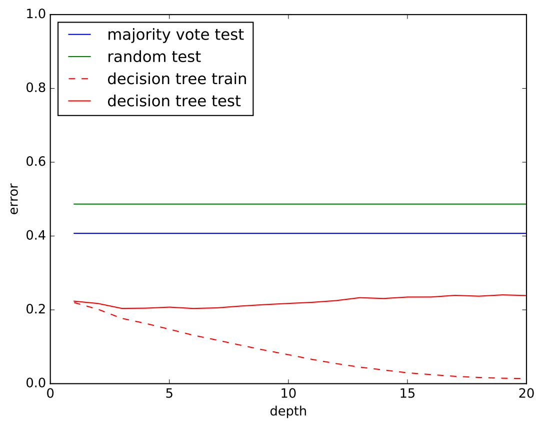
Solution: $k = 7$



(g) **(10 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. 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:

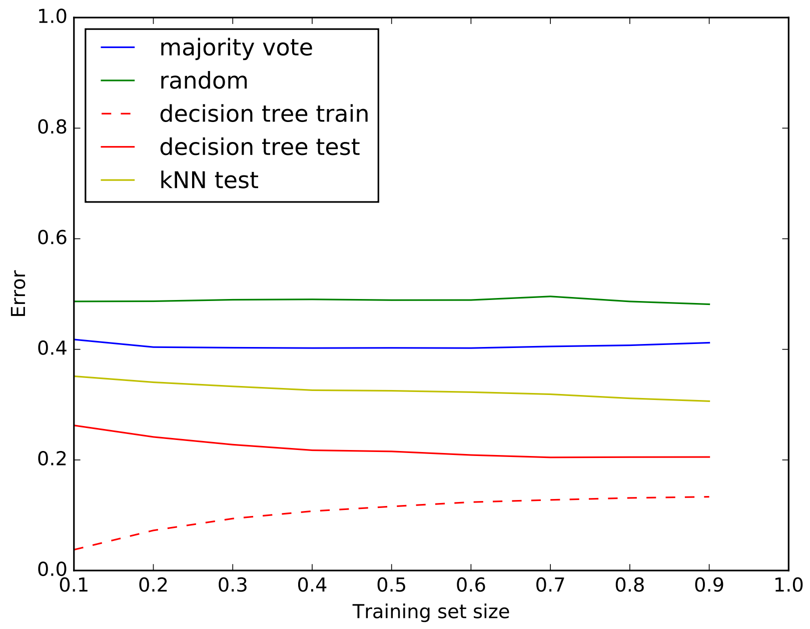


Based on the test error, the optimal depth is 6. Training error decreases with depth, but test error decreases then increases, indicating that the classifier is overfitting.

(h) **(10 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). For this experiment, first generate a random 90/10 split of the training data and do the following experiments considering the 90% fraction as training and 10% for testing.

Run experiments for the decision tree and k-nearest neighbors classifier with the best depth limit and k value you found above. This time, vary the amount of training data by starting with splits of 0.10 (10% of the data from 90% fraction) and working up to full size 1.00 (100% of the data from 90% fraction) in increments of 0.10. Then plot the decision tree and k-nearest neighbors training and test error against the amount of training data. 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.