CM146, Winter 2024
Problem Set 1: Decision trees, Nearest neighbors
Due Jan 30, 2024 at 11:59 pm PST

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 $f : X \to Y$, 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 (X_i for $i \geq 4$ is not considered). 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) **(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. Justify and answer for the general case when $n \geq 4$ for full credit.)

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 $n \geq 4$.

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).

(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? (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) **(5 pts)** What is the entropy of the label Y?

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

(d) **(5 pts)** Is there a split that reduces the entropy of 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. Please use logarithm in base 2 to report Entropy.)

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 [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 $H(S) = B\left(\frac{p}{p+n}\right)$. 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) **(2 pts)** Show that $0 \leq H(S) \leq 1$ and that $H(S) = 1$ when $p = n$.

Solution: First to show that $H(s) \leq 1$, we must establish that H is a concave function (either imply this or show $\frac{\partial^2 B}{\partial q^2} < 0$ in $0 \leq q \leq 1$). Then the only maxima of this function is when the first derivative is zero: $\frac{\partial B}{\partial q} = log(\frac{1-q}{q}) = 0$, which occurs when $q = \frac{1}{2}$ and plugging back into $H(s)$, we get $B(\frac{1}{2}) = 1$. Therefore we know that the function is less than or equal to 1. Since H is concave, either boundaries must be the global minima: where $q = 1$, and $q = 0$. $B(1) = 0$ and $B(0) = 0$, therefore $0 \leq H(s)$ (plugging in numbers without this argumentation will receive point deductions).
Where $p = n$, $q = \frac{1}{2}$ which is where $H(s) = B(\frac{1}{2}) = 1$

(b) **(3 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

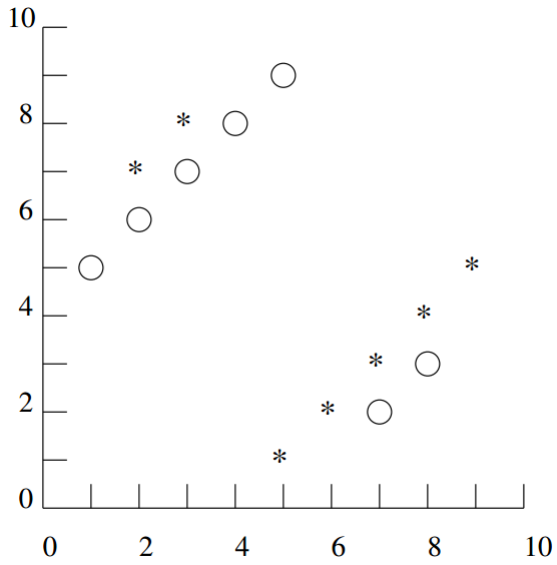
$$\sum_k \frac{p_k + n_k}{p + n} B\left(\frac{p_k}{p_k + n_k}\right) = \frac{\sum_k p_k + n_k}{p + n} B\left(\frac{p}{p + n}\right) = \frac{p + n}{p + n} B\left(\frac{p}{p + n}\right) = B\left(\frac{p}{p + n}\right)$$

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.

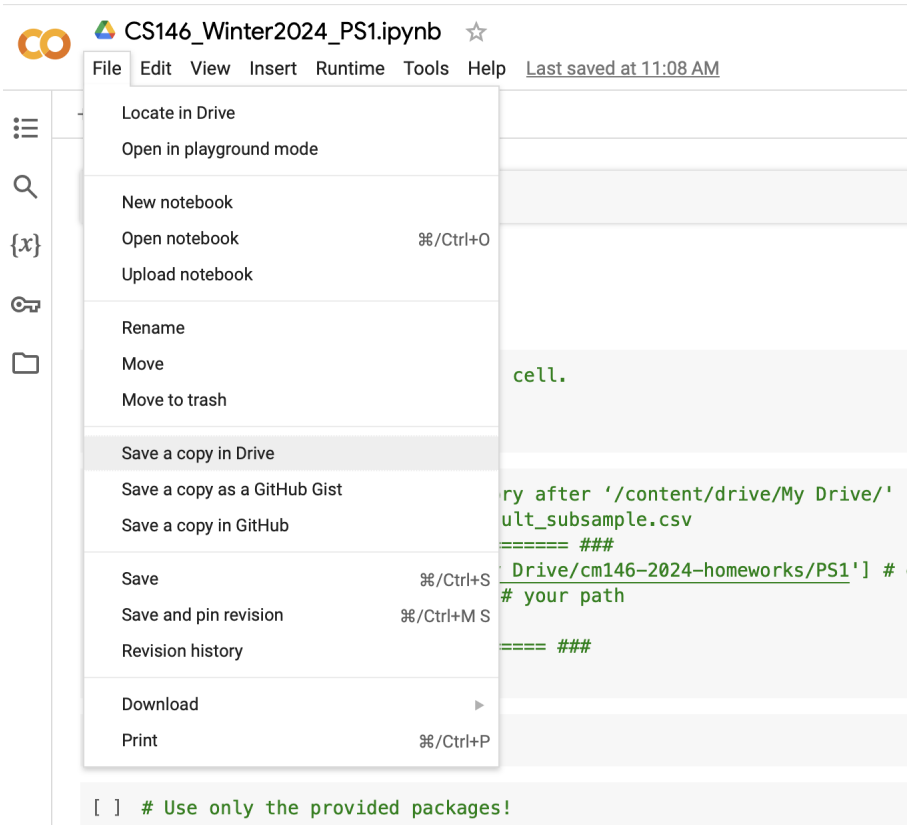
Using too small or too large a value for k reduces test set accuracy because the former leads to overfitting and the latter to underfitting.

4 Programming exercise : Applying decision trees and k-nearest neighbors [50 pts]

To work on this HW: you need to download two files (i) nutil.py (ii) adult_subsample.csv from here. Then copy/upload them to you own Google drive. If you cannot open the notebook with Colab or don't see an option, click on "Open with" and select "Connect more apps", where you can add Colab.

Next, for all the coding, please refer to the following colab notebook CS146-Winter2024-PS1.ipynb.

Before executing or writing down any code, please make a copy of the notebook and save it to your own google drive by clicking the File → Save a copy in Drive



You will then be prompted to log into your google account. Please make sure all the work you implement is done on your own saved copy. You won't to able to make changes on the the original notebook shared for the entire class. Running the first two cells will further mount your own google drive so that your copy of the Colab notebook will have access to the two files (nutil.py and adult_subsample.csv) you have just uploaded.

The notebook has marked blocks where you need to code.

$$\#\#\# \text{ ========= } TODO : START \text{ ========= } \#\#\#$$

$$\#\#\# \text{ ========= } TODO : END \text{ ========== } \#\#\#$$

For the questions please read below.

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.

Note: We have already converted all the categorical features to numerical ones. The target column is the last one: ">50k", where 1 and 0 indicate >50k or $\leq$ 50k respectively. The feature "fnlwgt" describes the number of people the census believes the entry represents. All the other feature names should be self-explanatory. If you want to learn more about this data please click here

(a) **(5 pts)** Make histograms for each feature, separating the examples by class by running the function plot_histograms in the notebook. This should produce fourteen 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 if any? (Please only describe the general trend. No need for more than two sentences per feature)

Solution: Anything goes as long as it's within reason.

4.2 Evaluation [45 pts]

Now, let's 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) **(0 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 85% of the examples in the training set have `>50k = 0` and 15% have `>50k = 1`, then, when applied to a test set, `RandomClassifier` should randomly predict 85% of the examples as `>50k = 0` and 15% as `>50k = 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 0.374 or 0.385.

(c) **(10 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

(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:
$k = 3$: 0.153
$k = 5$: 0.195
$k = 7$: 0.213

(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's use cross-validation instead.

Implement the missing portions of `error(...)` according to the provided specifications. You may find it helpful to use `StratifiedShuffleSplit(...)` 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 same (e.g., 0).

[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(...)`.

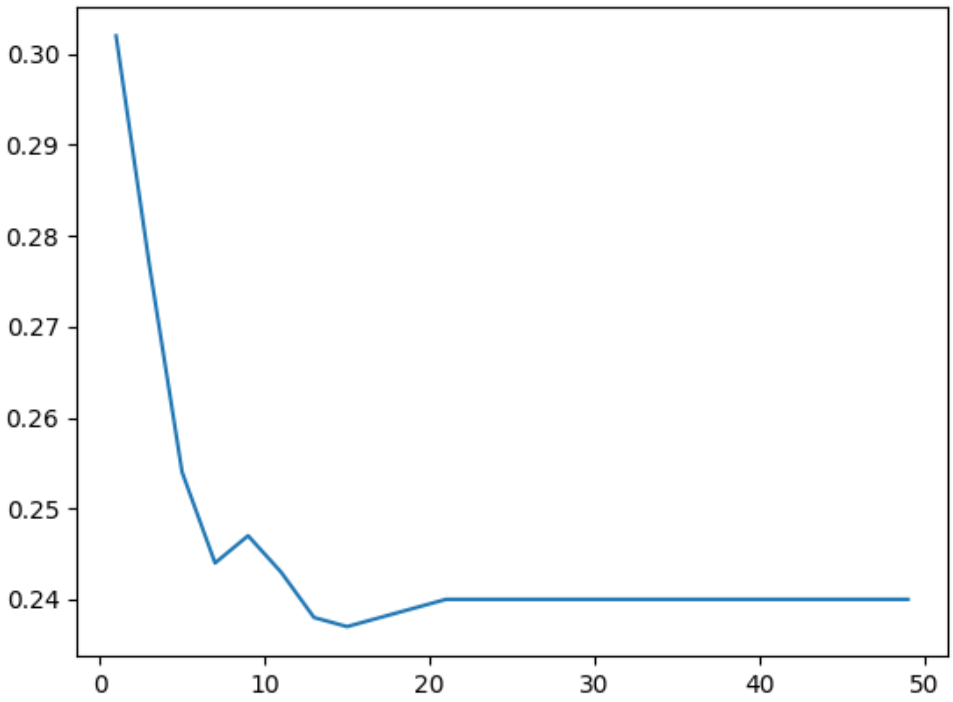
Next, use your `error(...)` function to evaluate the average cross-validation training error, test error and test micro averaged F1 Score (If you dont́ know what is F1, please click here) of each of your four models (for the `KNeighborsClassifier`, use k=5). To do this, generate a random 80/20 split 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 adult_subsample data set?

Solution:

classifier	train error	test error	F1 score
majority	0.24	0.24	0.76
random	0.374775 or 0.36855	0.382 or 0.3489	0.618 or 0.6511
decision tree	0	0.20475	0.79525
kNN	0.201675	0.25915	0.74085

(f) **(5 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?

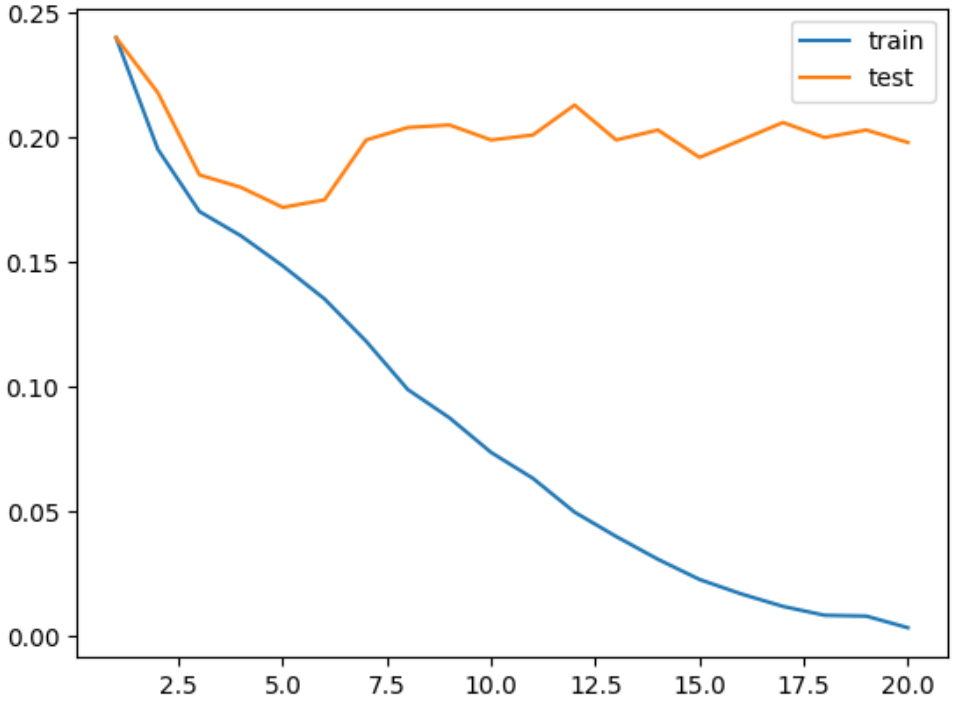
Solution: The best value of k is **15**. Students should observe underfitting with larger k values and overfitting with smaller k values.



(g) **(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's see whether this is the case.

One way to prevent decision trees from overfitting is to limit their depth. Repeat your cross-validation experiments but for increasing depth limits, specifically, $1, 2, \ldots, 20$. You may find `cross_validate(...)` from `scikit-learn` helpful. 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.

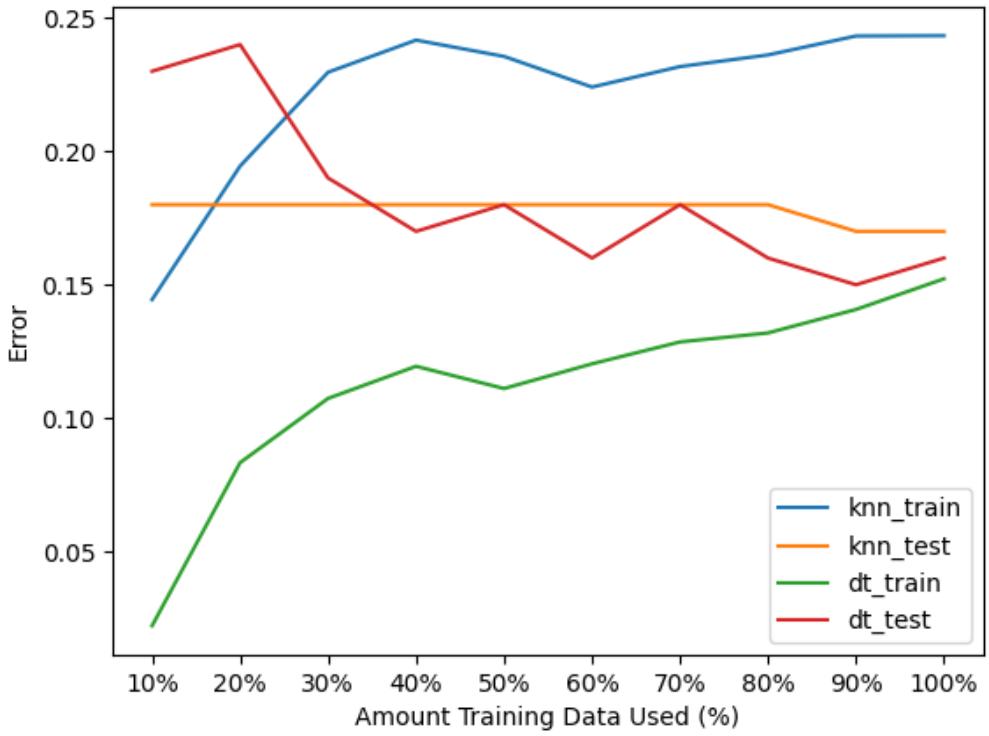
Solution: The best depth limit is **5**. Students should observe underfitting with smaller depth limits and overfitting with larger depth limits.



(h) **(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). For this experiment, first generate a random 90/10 split of the training data using `train_test_split` from `scikit-learn` with `random_state` set to 0. Then, 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: Anything goes as long as it's within reason. An example plot is provided, but values can differ greatly on this one.



(i) **(5 pts)** Pre-process the data by standardizing it. See the sklearn.preprocessing.StandardScaler package for details. After performing the standardization such as normalization please run all previous steps part (b) to part (h) and report what difference you see in performance.

Solution: The majority and random classifiers are not affected at all by the standardization. The decision tree classifier performs the same with standardization. The kNN classifier seems to perform significantly better with standardization. The standardization results are provided below

b. same as part (b)

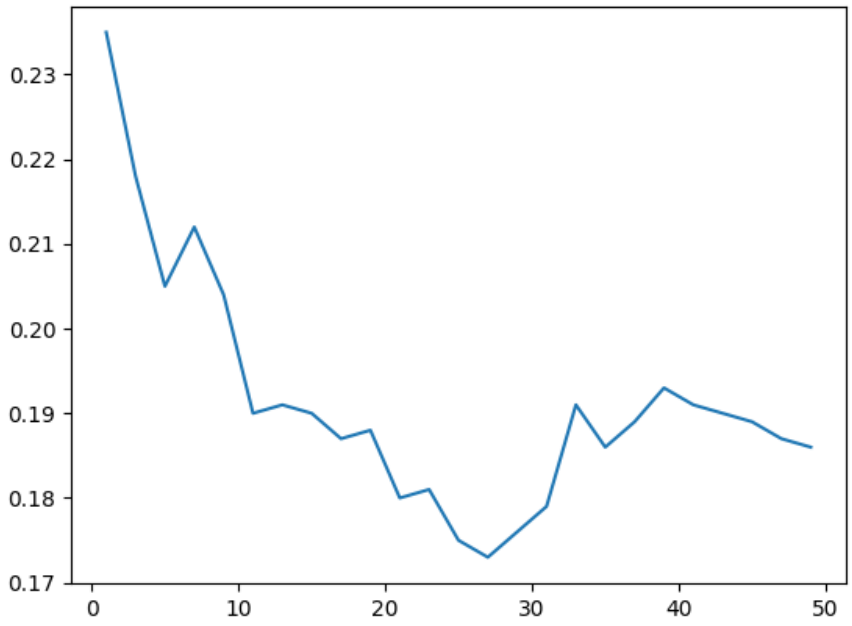
c. same as part (c)

d. $k = 3$: 0.114
$k = 5$: 0.129
$k = 7$: 0.152

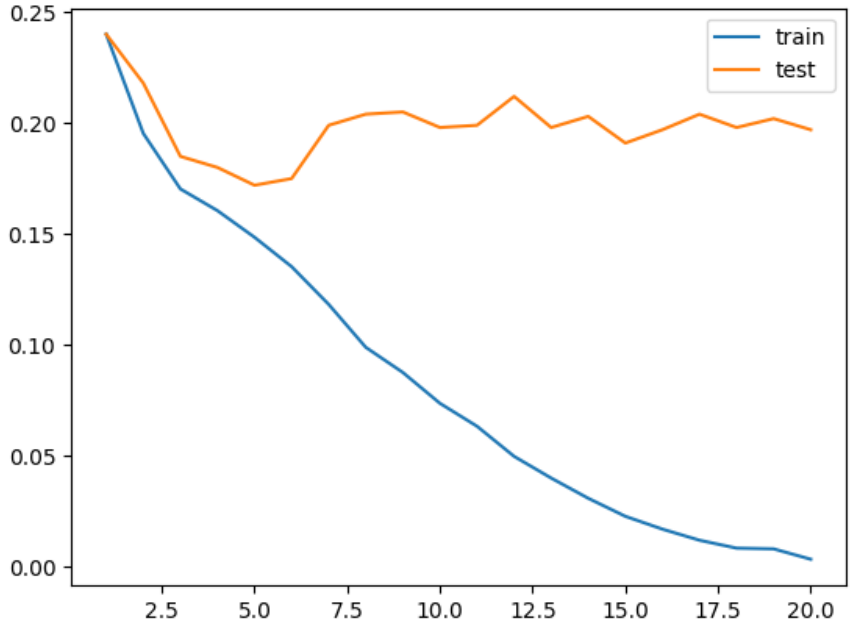
e. majority and random are the same as part (e)

classifier	train error	test error	F1 score
decision tree	0	0.2052	0.7948
kNN	0.13265	0.209	0.791

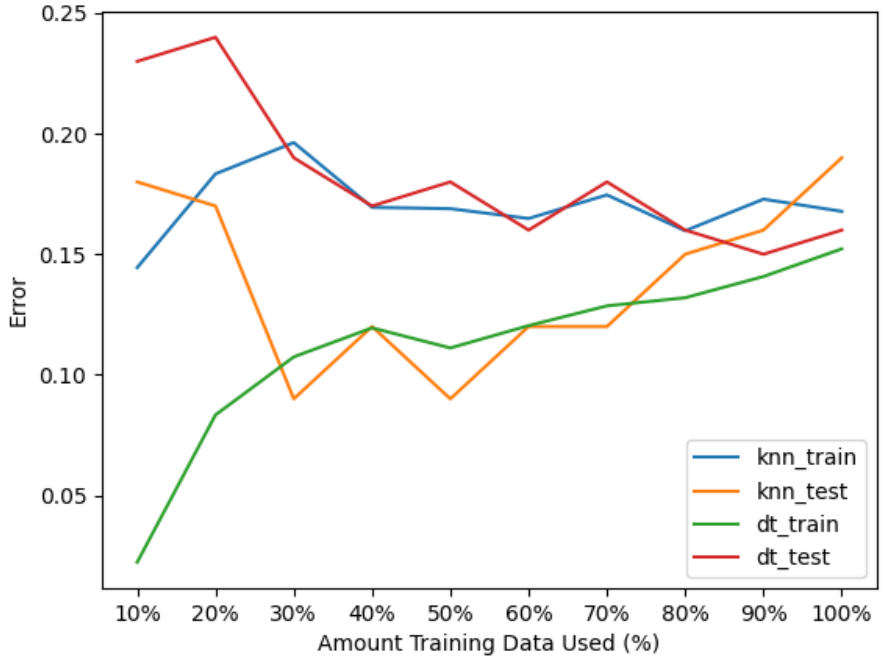
f. The best value of k is **27**.



g. The best depth limit is **5** (same as part (g)).



h. Just like with part (h), values can differ greatly and anything within reason is acceptable.


0.25
0.20
0.15
0.10
0.05
Error
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
Amount Training Data Used (%)
knn_train
knn_test
dt_train
dt_test