Problem Set 1

Graded

Student

BRANDON LO

Total Points

82.5 / 85 pts

Question 1

Splitting Heuristic for Decision Trees — **20** / 20 pts

1.1 **a** — **5** / 5 pts

✔ **– 0 pts** Correct

1.2 **b** — **5** / 5 pts

✔ **– 0 pts** Correct

1.3 **c** — **5** / 5 pts

✔ **– 0 pts** Correct

1.4 **d** — **5** / 5 pts

✔ **– 0 pts** Correct

Question 2

Entropy and Information — **4.5** / 5 pts

2.1 **a** — **1.5** / 2 pts

✔ **– 0.5 pts** The definition of entropy is not sufficient to prove $0 \leq H(S) \leq 1$, missing derivative

2.2 **b** — **3** / 3 pts

✔ **– 0 pts** Correct

Question 3

k-Nearest Neighbor — **10** / 10 pts

3.1 a — **3** / 3 pts

✔ **– 0 pts** Correct

3.2 b — **3** / 3 pts

✔ **– 0 pts** Correct

3.3 c — **4** / 4 pts

✔ **– 0 pts** Correct

Question 4

Programming exercise — **48** / 50 pts

4.1 a — **5** / 5 pts

✔ **– 0 pts** Correct

4.2 b — **0** / 0 pts

✔ **– 0 pts** Correct

4.3 c — **10** / 10 pts

✔ **– 0 pts** Correct

4.4 d — **5** / 5 pts

✔ **– 0 pts** Correct

4.5 e — **10** / 10 pts

✔ **– 0 pts** Correct

4.6 f — **3** / 5 pts

✔ **– 2 pts** Incorrect/missing observation

4.7 g — **5** / 5 pts

✔ **– 0 pts** Correct

4.8 h — **5** / 5 pts

✔ **– 0 pts** Correct

4.9 i — **5** / 5 pts

✔ **– 0 pts** Correct

Questions assigned to the following page: 1.1, 1.2, and 1.3

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.)

The best 1-leaf decision tree would predict 1 for any input (output will be 1 more often than 0). Therefore, we make a mistake when $Y=0$, which occurs when $x_1=x_2=x_3=0$. This will be 2^{n-3} mistakes because the other $n-3$ features can be one of two options (0 or 1).

(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.)

No, If there is a split on x_i where $i \geq 4$, Y will equal 1 $7/8$ of the time. If you choose a split on x_i where $1 \leq i \leq 3$, Y will be 1 when $x_i = 1$ and 1 $3/4$ of the time when $x_i = 0$. Both still predict 1, and hence have the same amount of mistakes

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

$$P(0) = \frac{2^{n-3}}{2^n} = \frac{2^n \cdot 2^{-3}}{2^n} = \frac{1}{8}$$

$$P(1) = 1 - \frac{1}{8} = \frac{7}{8}$$

$$H(Y) = -\frac{7}{8}\log\left(\frac{7}{8}\right) - \frac{1}{8}\log\left(\frac{1}{8}\right) = 0.544$$

Question assigned to the following page: 1.4

(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.)

We can split at x_i where $1 \leq i \leq 3$

When $x_i = 1$, $Y=1$, so entropy is 0
When $x_i = 0$,

$P(0) = \frac{1}{4}$
$P(1) = \frac{3}{4}$

$H(Y \mid x_i) = \frac{1}{2}(0) + \frac{1}{2}\left(-\frac{3}{4}\log\left(\frac{3}{4}\right) - \frac{1}{4}\log\left(\frac{1}{4}\right)\right) = 0.406$ Yes

Questions assigned to the following page: 2.1 and 2.2

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 $p = n$.

$H(S) = -\left(\frac{p}{p+n}\right)\log\left(\frac{p}{p+n}\right) - \left(1-\frac{p}{p+n}\right)\log\left(1-\frac{p}{p+n}\right)$

Split into cases:

$p=0$ $\quad -0\log(0) - 1\log(1) = 0$

$p<n$ $\quad$ ex: $p=1, n=3$

$-\left(\frac{1}{4}\right)\log\left(\frac{1}{4}\right) - \left(\frac{3}{4}\right)\log\left(\frac{3}{4}\right) = 0.811$

$p=n$ $\quad -\frac{1}{2}\log\left(\frac{1}{2}\right) - \frac{1}{2}\log\left(\frac{1}{2}\right) = 1$

$p>n$ $\quad$ ex: $p=3, n=1$

$-\left(\frac{3}{4}\right)\log\left(\frac{3}{4}\right) - \left(\frac{1}{4}\right)\log\left(\frac{1}{4}\right) = 0.811$

Therefore, $0 \leq H(S) \leq 1$ and $H(S) = 1$ when $p = n$

(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.

Gain: $B\left(\frac{p}{p+n}\right) - \sum_{k=1}^{x} \frac{p_k+n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right)$

$p = \sum_{k=1}^{x} p_k$, $n = \sum_{k=1}^{x} n_k$

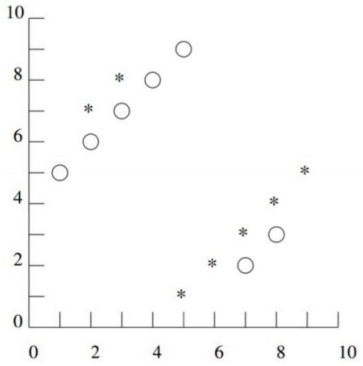
$B\left(\frac{p}{p+n}\right) - \frac{p+n}{p+n} B\left(\frac{p}{p+n}\right)$

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

Questions assigned to the following page: 3.1 and 3.2

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?

The value of k that minimizes training set error is k=1, which gives a training set error of 0. This is not a reasonable estimate because we do not know how we wish to classify different objects, and this could lead to overfitting instead of finding a general solution we are looking for.

(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?

The value of k that minimizes the leave-one-out cross-validation is k=5, which gives an error of 4/14. The only points misclassified would be (7,2), (8,3), (2,7), and (3,8) due to the amount of points of the opposite type nearby. Cross-validation is a better measure of test set performance because it gives a more reliable assessment of a model's performance in comparison to a single train-test split. It also aids in determining a model's ability to generalize data and perform well on unseen data.

Question assigned to the following page: 3.3

(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?

The lowest value of k for this data set is k=1. This would misclassify all points except (1,5), (5,9), (5,1), and (9,5), so the error is 10/14. This is because the misclassified points' nearest point is of the opposite type.

The highest value of k for this data set is k=13 (you leave one out, so 14-1=13). This would misclassify all points because there is an equal amount of each type of points, therefore whichever one is left out will be misclassified as there will be less of that type. The error is therefore 14/14, or 100%.

Using too large of a value for k is bad because it can lead to overfitting, as it takes into account more data than is needed or is relevant. Using too small a value for k is bad because it can lead to underfitting, as it takes into account too little data than is needed to make an accurate prediction.

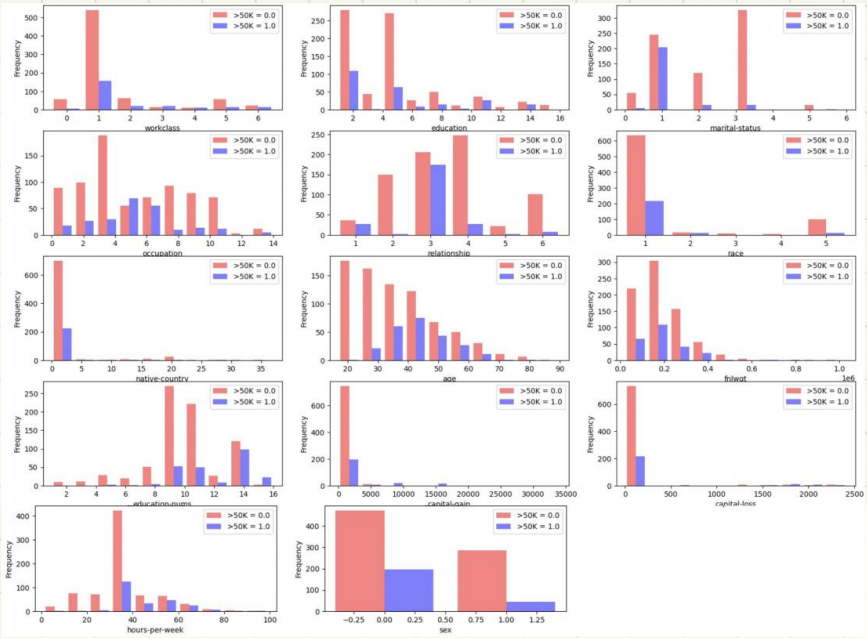
Question assigned to the following page: 4.1

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 ≤ 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)



Trends:

Workclass: Far more people are Self-emp-not-inc(1) than any other workclass. The most people who make >50k are part of this workclass while you are most likely to make >50k as Federal-gov (3).

Education: Most people who make >50k have a bachelor's degree(1). With more education you have a higher chance of making >50k, such as masters (10) and doctorate (13).

Marital Status: Most people who make >50k are divorced, and they have the highest chance of making >50k.

Occupation: The occupations who have most people who make >50k are Sales (3) and Exec-managerial (4). Sales has the highest chance to make >50k.

Question assigned to the following page: 4.1

Relationship: Most people are husbands (3) and they are also most likely to make >50k

Race: Most people are white (1). It also seems they are most likely to make >50k based on this data.

Native Country: Most people are from the United-States. A little under $\frac{1}{4}$ make over >50k, while there is too little data on other countries to make conclusions.

Age: All people who make >50k are 30-70 years old, while 40-60 year olds have the greatest chance to make >50k.

fnlwgt: The fraction of people who make more than 50k is fairly constant. Their are also more people who make less than 50k

education-nums: Most people who make >50k have more than 8 years of education. You have the best chance of making >50k if you have 14 years of education

capital-gain: Most people have low capital gain and those with high capital gain tend to make >50k

capital-loss: Most people have low capital loss. People with high capital loss seem fairly evenly distributed between making >50k and <50k.

hours-per-week: Most people work 40 hours a week. You have the highest chance of making >50k working 40-80 hours a week.

sex: There are more women than men, and women have a higher chance of making >50k

Questions assigned to the following page: 4.2, 4.3, and 4.4

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.

Screenshot of training error:

```
Random:          train error =   0.37477500000000014
```

(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?

Screenshot of training error:

```
Decision Tree:   train error =   0.0
```

(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.

Screenshot of training errors:

```
training error when k = 3 : 0.15300000000000002
training error when k = 5 : 0.19499999999999995
training error when k = 7 : 0.21299999999999997
```

Questions assigned to the following page: 4.6 and 4.5

(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).

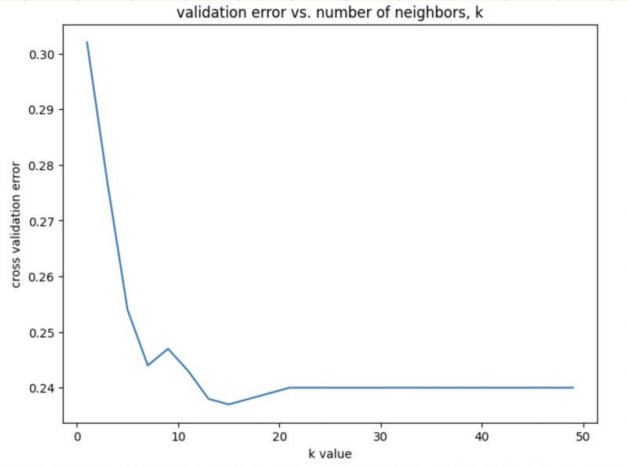
Next, use your `error(...)` function to evaluate the average cross-validation training error, test error and test micro averaged F1 Score (If you dont́ know what is F1, please click here) of each of your four models (for the `KNeighborsClassifier`, use k=5). To do this, generate a random 80/20 split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

Screenshot:

```
Majority: train error = 0.2399999999999996
Majority: CV test error = 0.2399999999999996
Majority: F1 score = 0.7600000000000002
Random: train error = 0.37477500000000014
Random: CV test error = 0.38199999999999984
Random: F1 score = 0.6180000000000002
Decision Tree: train error = 0.0
Decision Tree: CV test error = 0.20475
Decision Tree: F1 score = 0.7952500000000002
KNN: train error = 0.20167499999999997
KNN: CV test error = 0.25915000000000005
KNN: F1 score = 0.7408499999999998
```

(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?

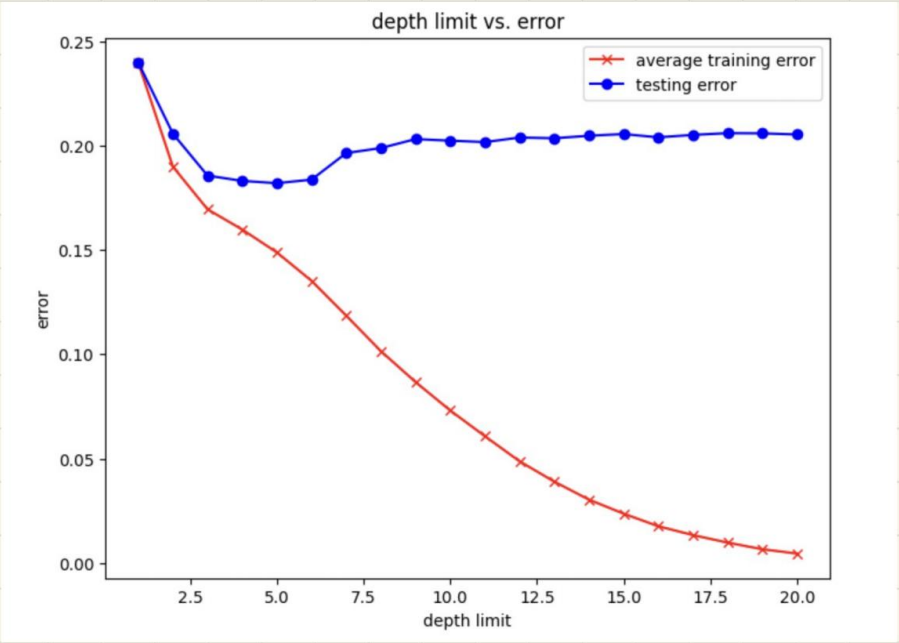


The lowest cross validation error occurs when k=15, therefore this is the best value of k.

Question assigned to the following page: 4.7

(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.

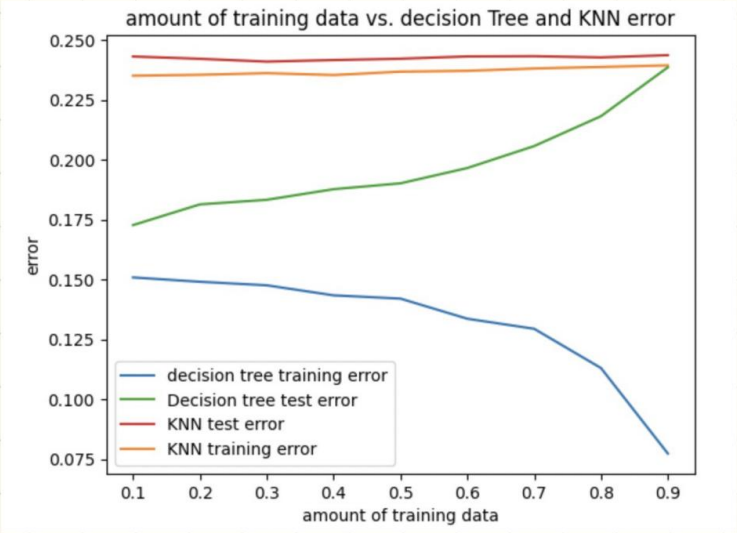


the best depth limit to use for this data is 5 because the testing error is lowest there. There is overfitting when depth is greater than 5 because the testing error increases,

Question assigned to the following page: 4.8

(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.



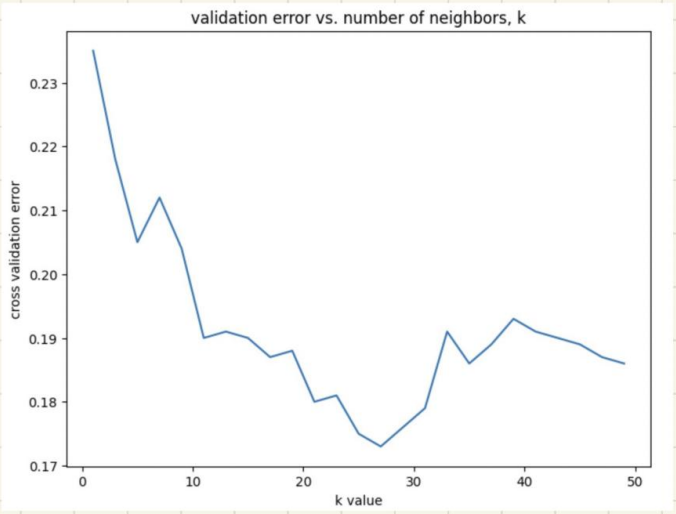
Used depth limit = 5 and $k = 15$.

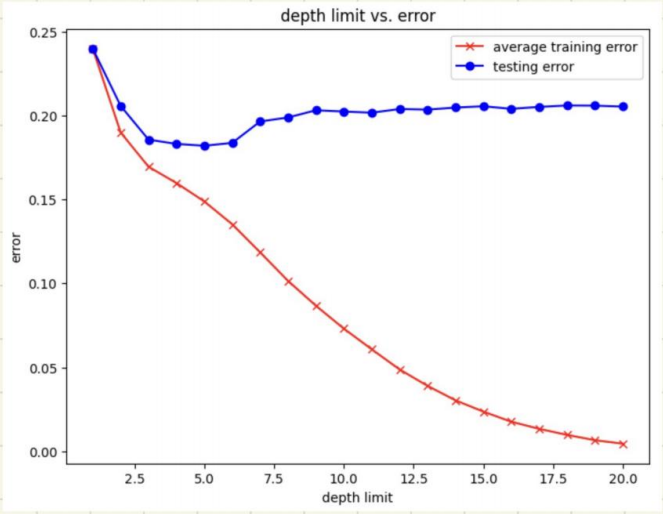
The decision tree training error increases and the decision tree test error decreases as the amount of training data increases. The KNN errors remain relatively constant. Training error is also always less than test error

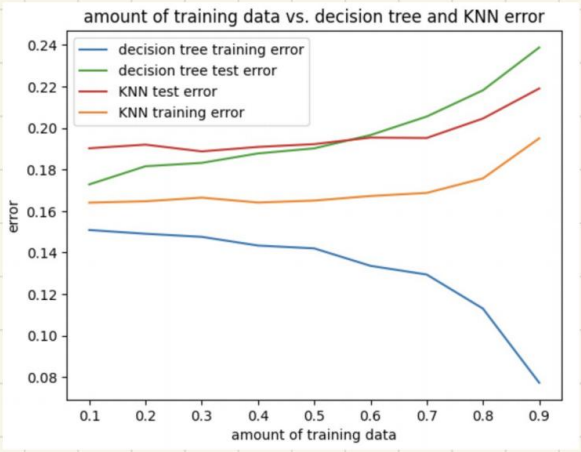
Question assigned to the following page: 4.9

(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.

```
Plotting...
Classifying using Majority Vote...
        -- training error: 0.240
Classifying using Random...
        --training error: 0.374
Classifying using Decision Tree...
training error: 0.0
Classifying using k-Nearest Neighbors...
training error when k = 3 : 0.11399999999999999
training error when k = 5 : 0.129
training error when k = 7 : 0.15200000000000002
Investigating various classifiers...
Majority: train error = 0.2399999999999996
Majority: CV test error = 0.2399999999999996
Majority: F1 score = 0.7600000000000002
Random: train error = 0.37477500000000014
Random: CV test error = 0.38199999999999984
Random: F1 score = 0.6180000000000002
Decision Tree: train error = 0.0
Decision Tree: CV test error = 0.20519999999999994
Decision Tree: F1 score = 0.7947999999999996
KNN: train error = 0.13265000000000002
KNN: CV test error = 0.20900000000000005
KNN: F1 score = 0.7910000000000004
```







The random and decision tree processes remain the same. However, the performance of KNN increased. Looking at the graph of validation error vs. number of neighbors, k, you can see that the best k value increased from k=15 to about k=27.