CS M146
PS1

February 1, 2025

Paul Kallarackel
906554018

---

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. 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. How many mistakes does the best 1-leaf decision tree make over the 2n 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.)

My solution:

For $Y = 0$ then all of the following $X_1 = X_2 = X_3 = 0$ aswell. Thus there

are 2^{n-3} instances for the other features to be either {0,1}. The best 1-leaf decision tree will output the majority class to minimise mistakes and since for n $\geq$ 4:

$$2^n - 2^{n-3} > 2^{n-3}$$

Therefore the outputted class will be $Y = 1$. This means a total of 2^{n-3} mistakes, e.g. $\frac{1}{8}$ incorrect predictions of total sample.

b. 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.)

My solution:

No, there is no split that reduces the number of mistakes by at least one.

From part (a), the best 1-leaf decision tree predicts the majority class ($Y = 1$), resulting in 2^{n-3} mistakes (all cases where $Y = 0$).

When we consider splitting on a single attribute (e.g., $X_1, X_2, \ldots, X_n$):

- Splitting on X_1, X_2, or X_3: For each of these splits, the total number of mistakes is 2^{n-3}, which is the same as the 1-leaf tree. This is because the branches $X_1 = 1$, $X_2 = 1$, or $X_3 = 1$ correctly classify their subsets with no mistakes, while the $X_1 = 0$, $X_2 = 0$, or $X_3 = 0$ branches result in the same number of mistakes as the 1-leaf tree as there's no dependence on the other n-3 features.
- Splitting on $X_4, X_5, \ldots, X_n$: These attributes do not affect Y (which depends only on X_1, X_2, X_3), so these splits also do not reduce the number of mistakes.

Thus, there is no single-attribute split that reduces the number of mistakes compared to the best 1-leaf decision tree.

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

My solution:

In part (a), the probability of $Y = 0$ was found to be $\frac{1}{8}$, therefore $p = P(Y = 1) = \frac{7}{8}$

$$\begin{aligned} H[s] &= -p\log_2(p) - (1-p)\log_2(1-p) \\ &= -\frac{7}{8}\log_2\left(\frac{7}{8}\right) - \left(1-\frac{7}{8}\right)\log_2\left(1-\frac{7}{8}\right) \\ &\approx 0.543 \end{aligned}$$

d. 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.)

My solution:

Yes, there is a split that reduces the entropy of the output Y. Splitting on X_1 (or X_2 or X_3, due to symmetry) reduces the entropy.

The conditional entropy is calculated as follows:

$$H(Y|X_1) = p(X_1 = 1)H(Y|X_1 = 1) + p(X_1 = 0)H(Y|X_1 = 0)$$

1. **Entropy in the Branch $X_1 = 1$:**

When $X_1 = 1$, $Y = 1$ for all examples. Hence, there is no uncertainty, and:

$$H(Y|X_1 = 1) = 0$$

2. **Entropy in the Branch $X_1 = 0$:**

When $X_1 = 0$, $Y = X_2 \vee X_3$. The probabilities are:

$$p(Y = 1|X_1 = 0) = \frac{3}{4}, \quad p(Y = 0|X_1 = 0) = \frac{1}{4}$$

The entropy for this branch is:

$$H(Y|X_1 = 0) = -\frac{3}{4}\log_2\left(\frac{3}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right)$$

Using $\log_2\left(\frac{3}{4}\right) \approx -0.415$ and $\log_2\left(\frac{1}{4}\right) = -2$, we have:

$$H(Y|X_1 = 0) = -\frac{3}{4}(-0.415) - \frac{1}{4}(-2) \approx 0.811$$

3. **Conditional Entropy:**

The probabilities for each branch are $p(X_1 = 1) = \frac{1}{2}$ and $p(X_1 = 0) = \frac{1}{2}$. Thus, the total conditional entropy is:

$$H(Y|X_1) = \frac{1}{2}(H(Y|X_1 = 1)) + \frac{1}{2}(H(Y|X_1 = 0))$$

$$H(Y|X_1) = \frac{1}{2}(0) + \frac{1}{2}(0.811) \approx 0.406$$

4. **Reduction in Entropy:**

The original entropy of Y is:

$$H(Y) \approx 0.544$$

The reduction in entropy is:

$$\Delta H = H(Y) - H(Y|X_1) \approx 0.544 - 0.406 = 0.138$$

Final Answer: Yes, there is a split that reduces the entropy. Splitting on X_1 reduces the entropy to $H(Y|X_1) \approx 0.406$.

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

My solution:

The entropy of set S is defined as:

$$H(S) = -\frac{p}{p+n} \log_2 \left(\frac{p}{p+n}\right) - \frac{n}{p+n} \log_2 \left(\frac{n}{p+n}\right)$$

1. Show that $0 \leq H(S) \leq 1$:

- When all examples belong to a single class (either $p = 0$ or $n = 0$), the entropy is:
$$H(S) = -1 \cdot \log_2(1) - 0 \cdot \log_2(0) = 0$$
This represents perfect certainty, so the entropy is 0.
- The entropy reaches its maximum when the distribution is uniform, i.e., when $\frac{p}{p+n} = \frac{n}{p+n} = \frac{1}{2}$. In this case:
$$H(S) = -\frac{1}{2} \log_2 \left(\frac{1}{2}\right) - \frac{1}{2} \log_2 \left(\frac{1}{2}\right) = -\frac{1}{2}(-1) - \frac{1}{2}(-1) = 1$$

Thus, $0 \leq H(S) \leq 1$.

2. Show that $H(S) = 1$ when $p = n$:

$$\text{If } p = n, \quad \frac{p}{p+n} = \frac{1}{2}.$$

Substituting into the entropy formula:

$$H(S) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1.$$

Hence, $H(S) = 1$ when $p = n$.

b. 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.

My solution:

The entropy of the entire set S is:

$$H(S) = B\left(\frac{p}{p+n}\right)$$

where $B(q) = -q\log_2(q) - (1-q)\log_2(1-q)$.

When we split S into k disjoint subsets S_k, the weighted average entropy of the subsets is:

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

Since the ratio $\frac{p_k}{p_k+n_k}$ is the same for all k, we have:

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

Because $\sum_k \frac{p_k+n_k}{p+n} = 1$, this simplifies to:

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

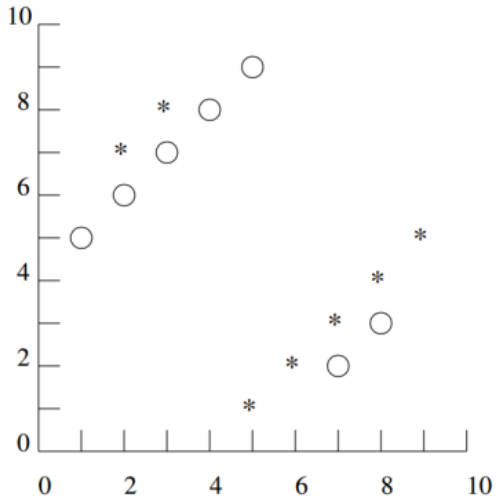
The information gain is the difference between the entropy before and after the split:

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

Therefore, if the ratio $\frac{p_k}{p_k+n_k}$ is the same for all k, the information gain is 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. 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?

My solution:

The value of k that minimizes the training set error is $k = 1$. This results in a training set error of 0 because each data point is its own nearest neighbor, perfectly classifying itself.

However, training set error is not a reasonable estimate of test set error, especially when $k = 1$, because it leads to **overfitting**. The model memorizes the training data rather than learning general patterns, making it sensitive to noise and outliers. This lack of generalization results in poor performance on unseen data, leading to a higher test set error despite perfect accuracy on the training set.

b. 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?

My solution:

The value of k that minimizes the leave-one-out cross-validation (LOOCV) error for this data set is $k = 5$. The resulting error is $\frac{4}{14}$.

Cross-validation, particularly LOOCV, is a better measure of test set performance because it evaluates the model's ability to generalize to unseen data. In LOOCV, each data point is used once as a validation point while the remaining data is used for training. This process is repeated for all data points, providing an unbiased and realistic estimate of the model's test performance. Unlike training set error, LOOCV prevents overfitting and offers insight into how the model behaves on new data.

c. 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?

My solution:

For the lowest $k = 1$, the LOOCV error is high due to **overfitting**. The model becomes overly sensitive to noise and individual data points, leading to poor generalization on unseen data.

For the highest k (where $k = 14$, the total number of data points), the LOOCV error is also high due to **underfitting**. The model predicts the majority class for all points, ignoring local structures and differences between classes.

Using too small a value of k (e.g., $k = 1$) causes overfitting, where the model memorizes the training data but fails to generalize to new data. Conversely, using too large a value of k causes underfitting, where the model is too simple and fails to capture meaningful patterns, leading to poor performance on both training and test sets.

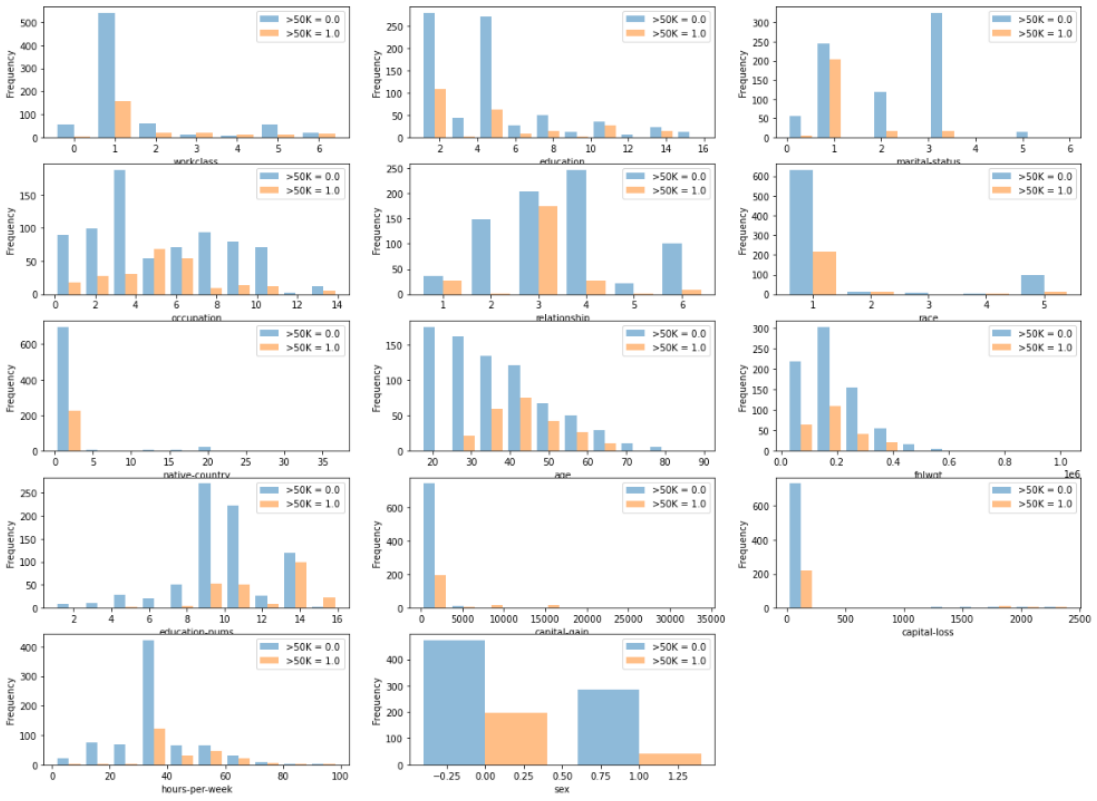
4.1 Visualization [5 pts]

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. Make histograms for each feature, separating the examples by class (e.g., income greater than 50k or smaller than or equal to 50k). 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? (Please only describe the general trend. No need for more than two sentences per feature)

My solution:



- **Workclass:** Most people are self-employed and governmental workers, who are the most likely to make more than 50k.
- **Education:** The more education each person has, the more likely they are to make above 50k.
- **Marital-status:** Divorced people are the most likely to make more than 50k a year.
- **Occupation:** Sales and exec-managerial are the most likely to make more than 50k.
- **Relationship:** Husbands are most likely to make more than 50k.
- **Race:** Most people in this dataset are White. Asians have a high fraction of people making more than 50k.
- **Native-country:** Most people in this study are from the U.S. Most of those people make less than 50k.
- **Age:** Younger people tend to not make above 50k. Individuals aged 40-50 have the highest chance of making above 50k.
- **Fnlwgt:** The fraction of people who make more than 50k is relatively constant. There are more people who make less than 50k.
- **Education-num:** Most people have 8-16 years of education. People with more years of education have a higher chance to make more than 50k.
- **Capital-gain:** Most people have low capital gain. People with high capital gain tend to make more than 50k.
- **Capital-loss:** Most people have low capital loss. People with high capital loss are evenly spread above and below 50k.
- **Hours-per-week:** Most people in this study work around 40 hours per week. People who work more hours a week tend to earn more than 50k.
- **Sex:** There are more women in this study. Women in this study have a higher chance of making more than 50k.

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.[2]

b. 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 $\geq$50k = 0 and 15% have $\geq$50k = 1, then, when applied to a test set, RandomClassifier should randomly predict 85% of the examples as $\geq$50k = 0 and 15% as $\geq$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.

My solution:

```
Classifying using Random...
        -- training error: 0.374
```

c. 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?

My solution:

```
Classifying using Decision Tree...
        -- training error: 0.000
```

d. 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.

My solution:

```
Classifying using k-Nearest Neighbors...
        -- training error for k=3: 0.153
        -- training error for k=5: 0.195
        -- training error for k=7: 0.213
```

e. 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 training error and (cross-validation) test error and test micro averaged F1 Score (If you dont 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?

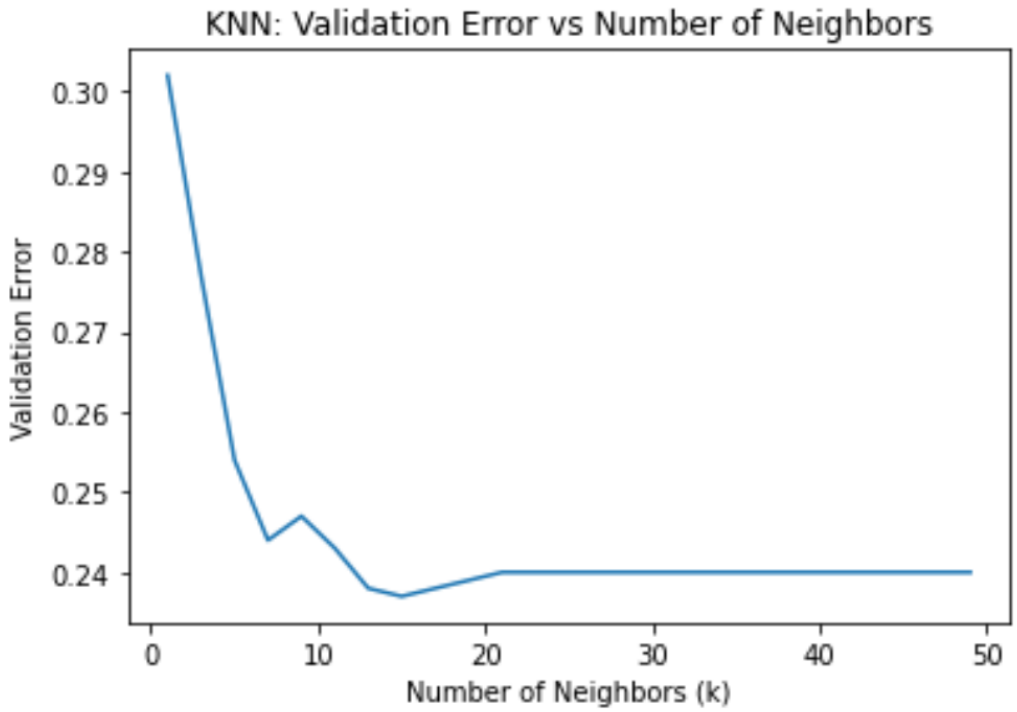
My solution:

```
Investigating various classifiers...
MajorityVote -- Train Error: 0.240, Test Error: 0.240, F1 Score: 0.760
Random -- Train Error: 0.378, Test Error: 0.386, F1 Score: 0.614
DecisionTree -- Train Error: 0.000, Test Error: 0.202, F1 Score: 0.798
KNN (k=5) -- Train Error: 0.202, Test Error: 0.266, F1 Score: 0.735
```

f. 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?

My solution:



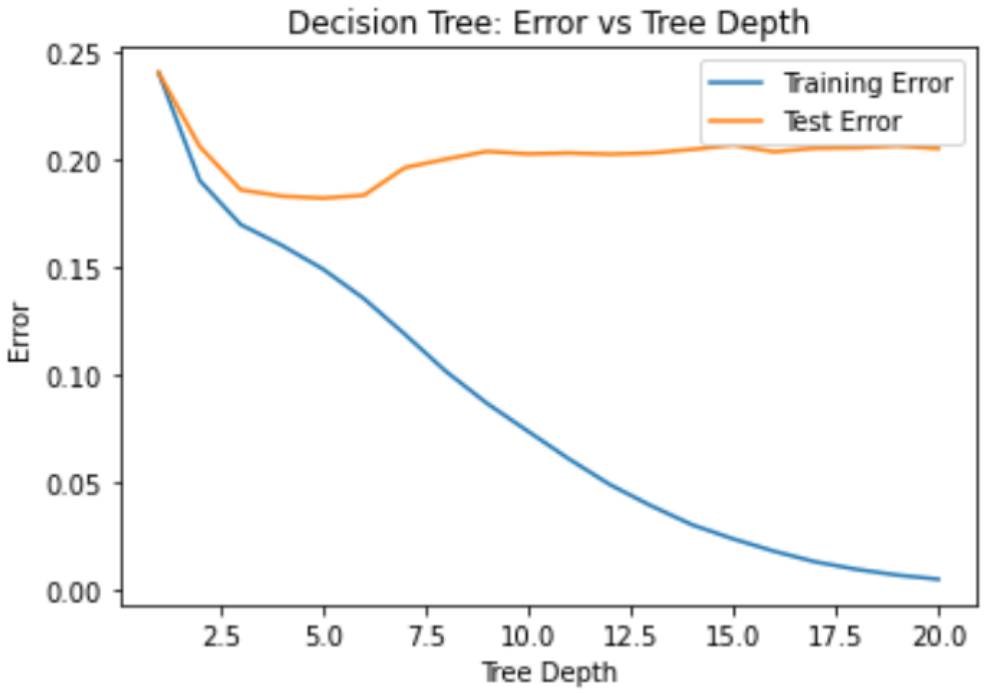

```
Investigating depths...
               Best value of k: 15, error: 0.2370000000000001

```

As we increase k the validation error decreases but then slightly increases and forms a trough. The best number of k the minimizes error is 15 with an error of approximately 0.237.

g. 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 crossvalidation experiments but for increasing depth limits, specifically, 1, 2, . . . , 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.

My solution:



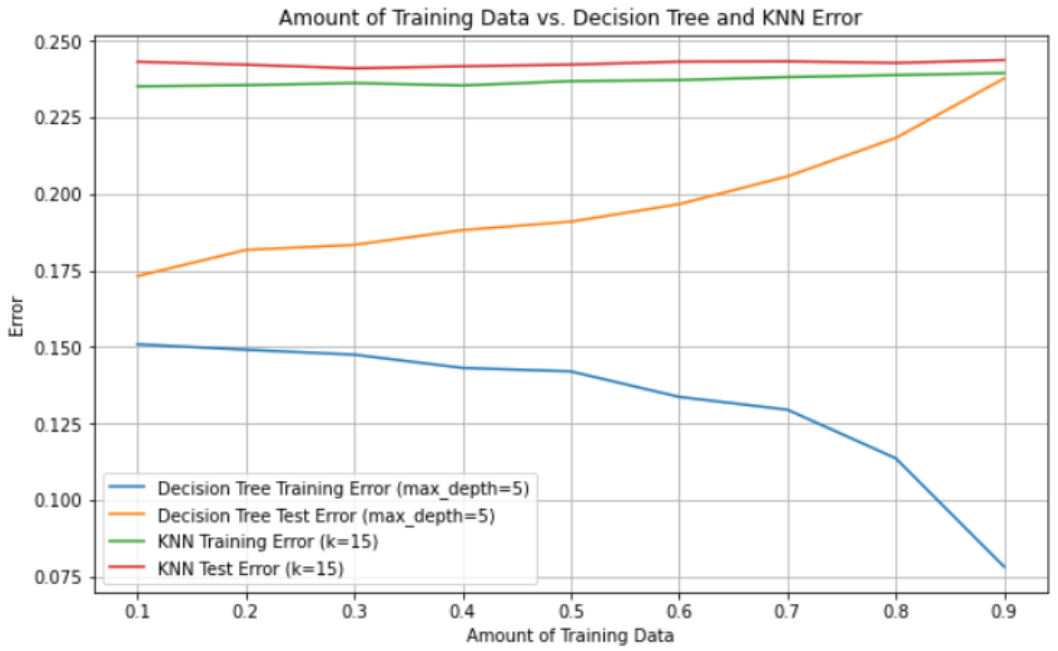

```
            Investigating depths...
Best value of depth: 5, test error: 0.18180000000000004

```

The best depth limit for this dataset is 5, with a test error of 0.1818. Increasing the depth beyond 5 leads to overfitting, as the test error begins to rise, indicating the model is fitting too closely to the training data and losing generalization.

h. 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.

My solution:



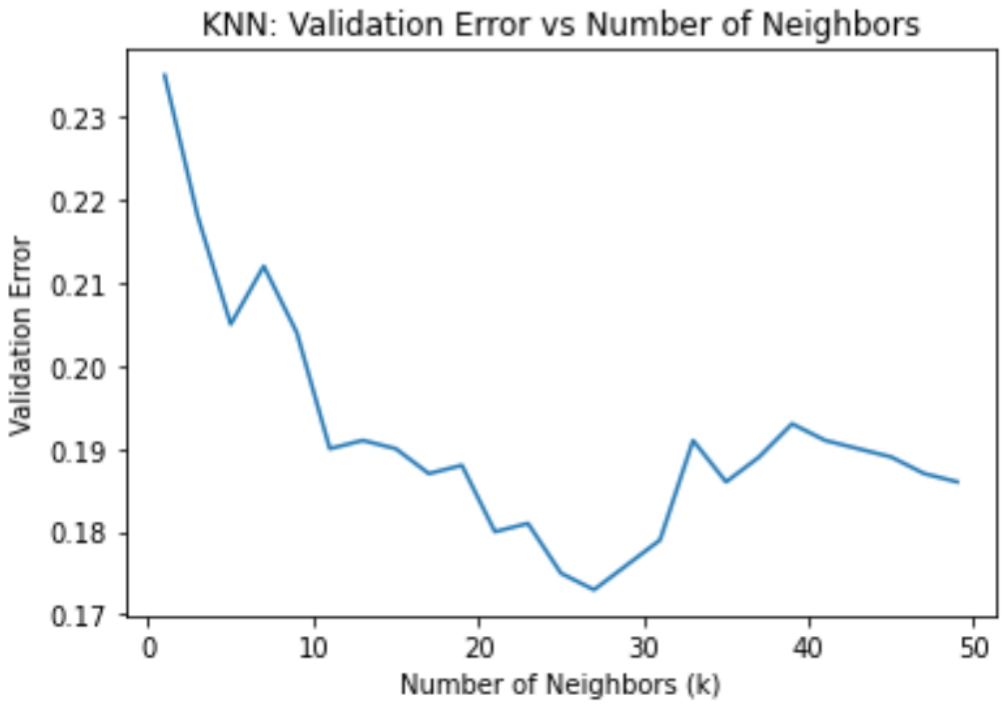
The Decision Tree error increases as the amount of training data increases, with the training error consistently lower than the testing error. In contrast, KNN error remains relatively stable regardless of the training data size.

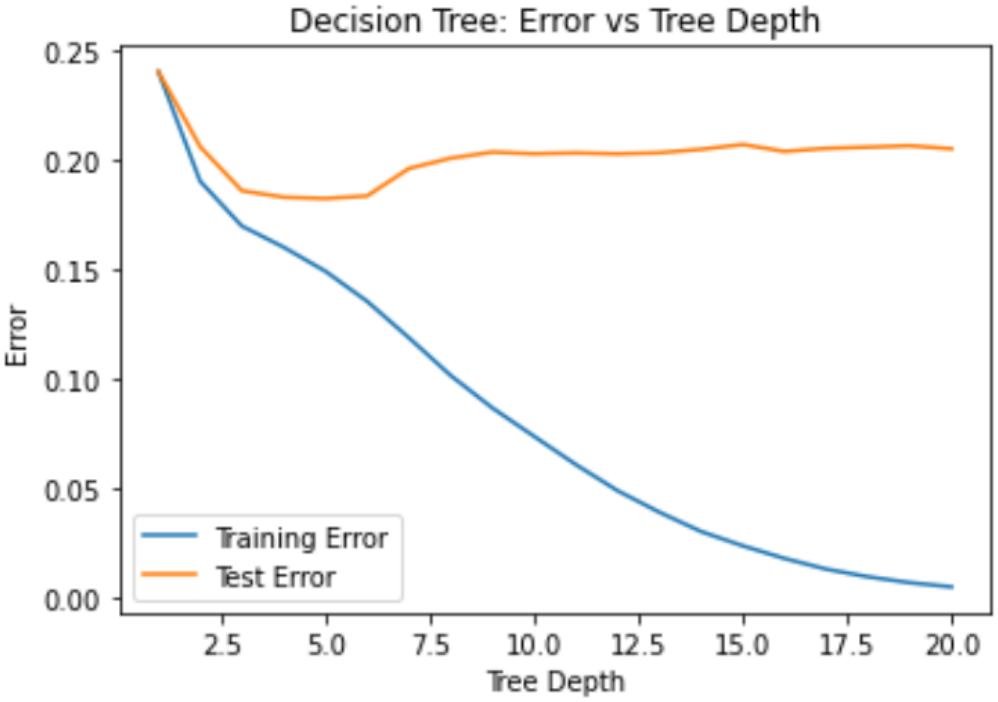
i. Pre-process the data by standardizing it. See the sklearn.preprocessing.StandardScaler package for details. After performing the standardization, run all previous steps from part (b) to part (h) and report the differences in performance.

My solution:

```
Classifying using Majority Vote...
    -- training error: 0.240
Classifying using Random...
    -- training error: 0.374
```

```
Classifying using Decision Tree...
    -- training error: 0.000
Classifying using k-Nearest Neighbors...
    -- training error for k=3: 0.114
    -- training error for k=5: 0.129
    -- training error for k=7: 0.152
Investigating various classifiers...
MajorityVote -- Train Error: 0.240, Test Error: 0.240, F1 Score: 0.760
Random -- Train Error: 0.378, Test Error: 0.386, F1 Score: 0.614
DecisionTree -- Train Error: 0.000, Test Error: 0.202, F1 Score: 0.798
KNN (k=5) -- Train Error: 0.133, Test Error: 0.206, F1 Score: 0.794
Finding the best k...
Best value of k: 27, error: 0.17300000000000004
Investigating depths...
Best value of depth: 5, test error: 0.18210000000000004

```





For the Random, Decision Tree, and Majority classifiers, the performance remains

unchanged after standardization. However, the performance of KNN improves significantly, and the best k value increases to 27.