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 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 $n \geq 4$.)

The best 1-leaf decision tree would be a tree that always predicts Y as 1; since it does not split the data even once, the best solution would be to always classify as 1 since that is the most common label. The best 1-leaf tree makes 2^{n-3} mistakes, an error rate of 1/8.

(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.)

No, since we can put any variable X_i to split at the root, and the rate of mistakes will still be 1/8. Splitting on any X_i for i = 1,2,3 will split the data into one leaf that contains only 1s (if X_i = 1) and one leaf where the proportion of 1s is 3/4 (since when X_i=0 where i=1,2,3, there are 3/4 1s and 1/4 0s). Splitting on X_i where i > 3 then the proportion of 1s will still be 7/8, and so both branches will predict 1. In both leaves, the tree predicts 1, so it makes the same number of errors as the 1-leaf decision tree that always predicts 1.

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

$$H[Y] = -\frac{7}{8}log_2\frac{7}{8} - \frac{1}{8}log_2\frac{1}{8} \approx 0.543$$

(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.)
Splitting at X1, X2, or X3 reduces the entropy of output Y by a nonzero amount.
i = 1,2,3

$$H(Y|X_i) = \frac{1}{2}(0) + \frac{1}{2}(-\frac{3}{4}log_2\frac{3}{4} - \frac{1}{4}log_2\frac{1}{4}) \approx 0.406$$

For each split of X1, X2, or X3, X_i is 1 half of the time and 0 the other half. When X_i is 1, then Y is 1 all the time, so there is no entropy; when X_i is 0, Y is 1 3/4 of the time and 0 1/4 of the time

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 $p = n$.

For a Bernoulli random variable, P(X=1) can range from 0 to 1, where q=0 is when the event never occurs and q=1 is when the event always occurs. In the above example, q=0 when there are no positive examples (p=0 and n>0). Therefore, the event of a positive example never occurs, so there is no uncertainty, and so the entropy is 0.

$$B(0) = -0 \log 0 - (1 - 0) \log(1 - 0) = 0.$$

The maximum entropy occurs when p=n (there are an equal number of positive and negative examples, so we are completely uncertain). When q=1, there are no negative examples (n=0 and p>0). Likewise, the entropy is 0.

The derivative of the entropy equation is $-log_2 q + log_2(1 - q)$. Setting it equal to 0, we get q = ½, and see that the maximum entropy is when q=½, or when p = n.
When p=n, there is an equal number of positive and negative examples; then

$$H(S) = B(\frac{n}{n+n}) = B(\frac{1}{2}) = -0.5 \log 0.5 - 0.5 \log 0.5 = 0.5 + 0.5 = 1$$

(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.

GAIN[Y, split] = H[Y] - H[Y | split] = 0

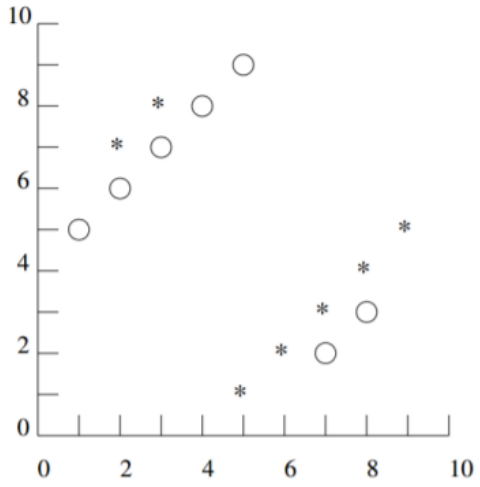
$$H(S) = B(\frac{p}{p+n}) = x$$

Each subset's entropy is equal (since the ratio $\frac{p_k}{p_k+n_k}$ is the same for all k), and if each subset has the same ratio, then the original set of p and n examples has the same ratio of p and n examples. So, no information is gained from a split. Since each subset's entropy is the same, then the conditional entropy given the split is $\frac{p_k+n_k}{p+n}x + \cdots + \frac{p_k+n_k}{p+n}x$ for k subsets, which is just equal to $\frac{p+n}{p+n}x = x$, the original entropy with no split. So, no information is gained:

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

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?

k=1 minimizes the training set error for this data set. The resulting training set error is 0 (since when k=1, each data point location is its own neighbor, and thus each data point is classified as its own class). Training set is not a reasonable estimate of test set error because when we are given unseen data, we cannot classify the unseen data point as its own class because it does not have a class yet.

(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?

k=5 or k=7 minimizes LOOCV error.

With k=5 and leave-one-out, the error rate is 4/14 (it misclassifies the 2 asterisks at (2,7) and (3,8) and the 2 circles at (7,2) and (8,3), while classifying the 5 grouped circles and 5 grouped asterisks correctly).

With k=7, the error rate is 4/14 (it misclassifies the same points, and classifies the 5 grouped circles and 5 grouped asterisks correctly since each of the 5 circles' 7 nearest neighbors are 4 circles and 3 asterisks with leave-one-out, and likewise for the 5 grouped asterisks).

Cross-validation is a better measure of test set performance because performance may be different for each split so we want the average performance to maximize the generalizability of the model to unseen 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?

The highest k for this data set is 13 (every remaining point in the training set). With LOOCV and there being only 7 asterisks and 6 circles or 7 circles and 6 asterisks when 1 is left out, every point would be misclassified (it just classifies every point as the most frequent class, and with LOOCV, the most frequent class will always be the wrong class). Using too large of a k leads to underfitting.

The lowest k for this data set is 1. The error rate for k=1 is 10/14; only the asterisks at (5,1) and (9,5) and the circles at (1,5) and (5,9) are classified correctly. As we can see, the error rates are high compared to k=5 or k=7; this is because the highest k misclassifies every point and having a low k leads to overfitting.

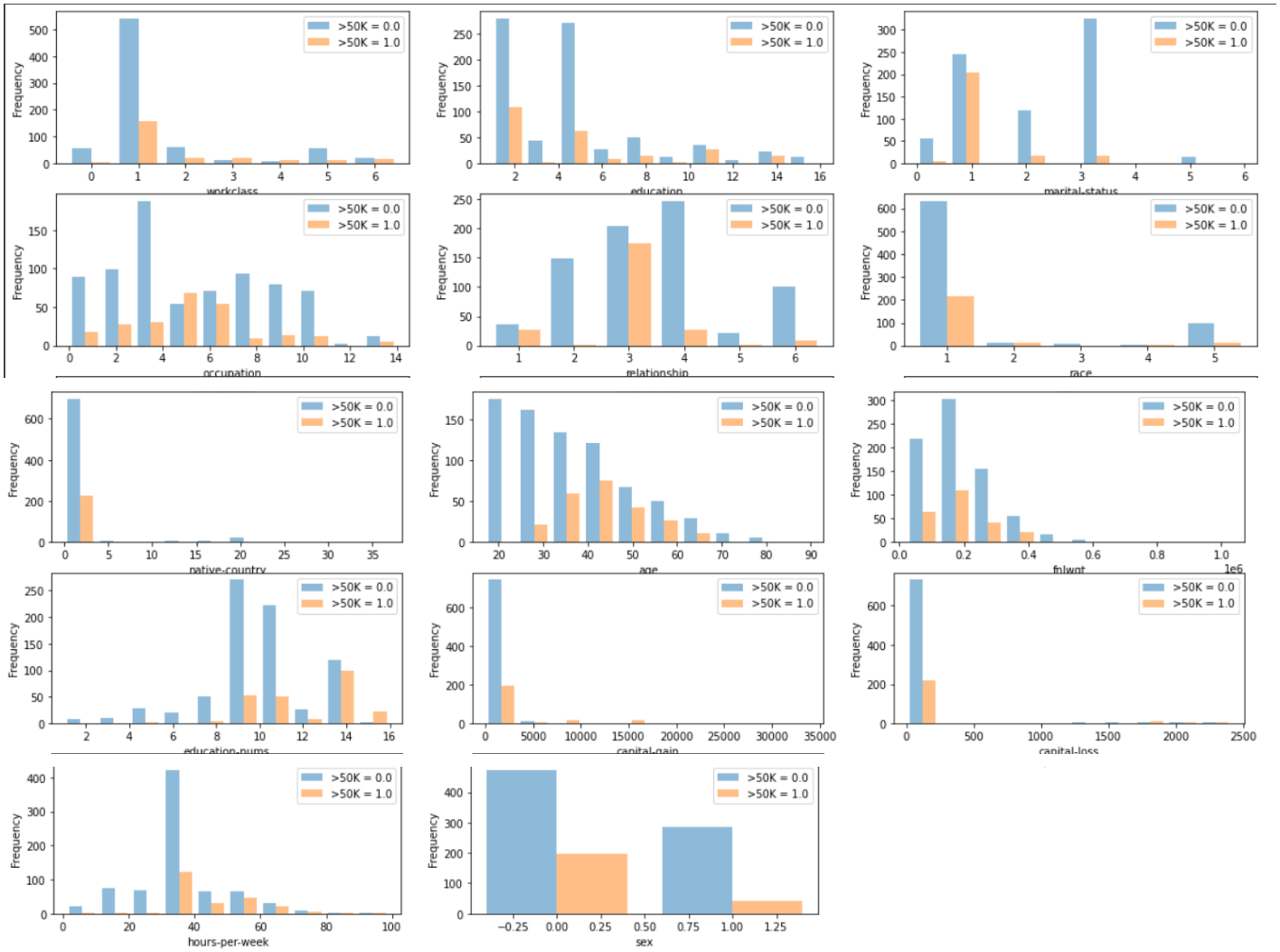
4 Programming [50 pts]

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.

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)



workclass: The majority of people are self-employed-not-inc. Notably, almost everyone that works in the private sector make less than 50k, and people that work in federal or local government have a higher chance to make more than 50k than less than 50k.
education: Most people with education level Bachelors or 11th grade make less than 50k. The ratio of people who make less than and more than 50k is much closer to 1 for HS grad, Prof school, Assoc-voc, and 7th-8th
marital status: Divorced people are most likely to make more than 50k; for all other marital statuses, one is much more likely to make less than 50k. occupation: Sales and Exec-managerial are nearly equally likely to make more than 50k as they are to make less than; for all other occupations, one is much more likely to make less than 50k.
relationship: Husbands are nearly equally likely to make more than 50k as they are to make less than; for all other relationship statuses, one is more likely to make less than.
race: White and black people are much more likely to make less than 50; for other races (Asians and Other) are more equally likely to make more or less than 50k.
native country: Almost all people's native country is the US in this data set; one is more likely to make less than 50k.

age: As one gets older, one is less likely to make less than 50k; the age groups that are most likely to make more than 50k are 30, 40, and 50 year olds.
final weight: The census believes each entry represents less than 300k people most often; the entry that is most likely to make more than 50k out of all entries is the one that represents ~200k people.
education number: As the education number increases, one is more likely to make more than 50k.
capital gain: As the capital gain increases, one is more likely to make more than 50k.
capital loss: As the capital loss increases, one is more likely to make more than 50k.
hours per week: People that work less than 40 hours a week are very unlikely to make more than 50k. People that work 40-60 hours are more likely to make more than 50k than people who work other hours.
sex: Males are unlikely to make more than 50k, while females are somewhat more likely to make more than 50k.

(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 `>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.

```
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?

```
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.

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

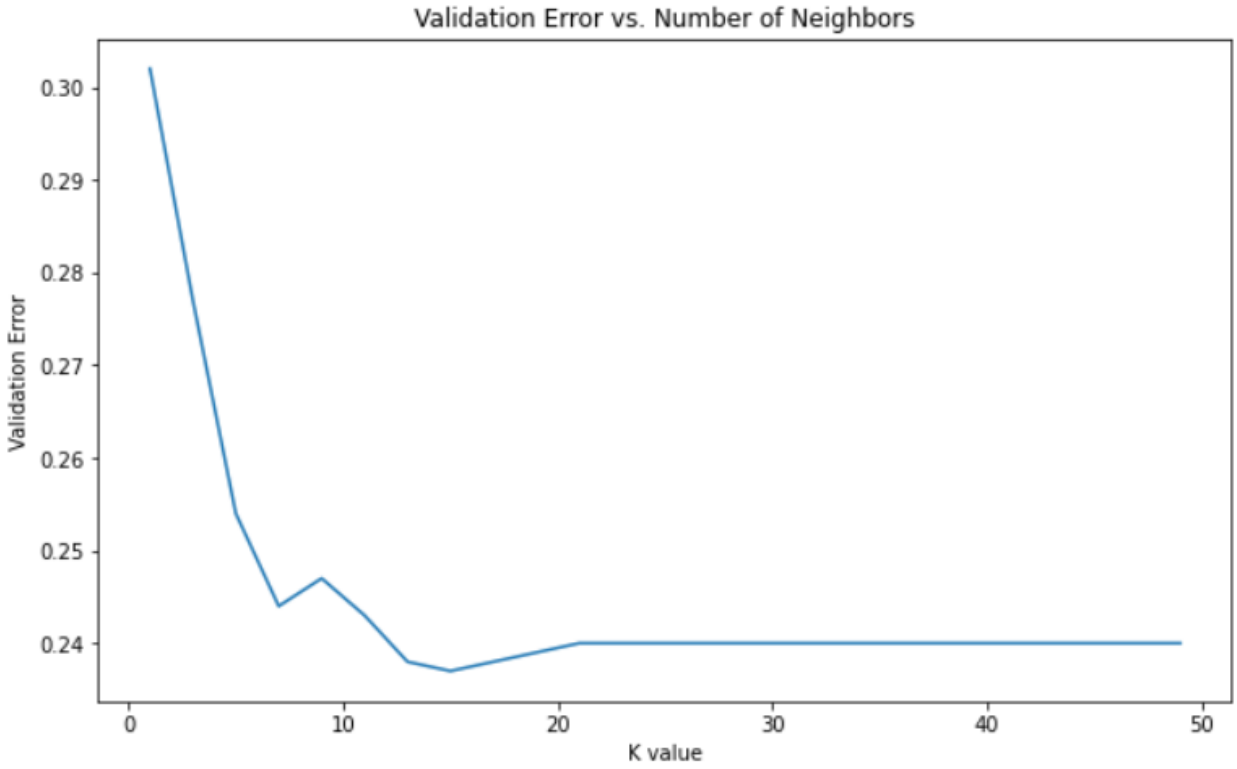
(e) So far, we have looked only at training error, but as we learned in class, training error is a poor metric for evaluating classifiers. Let's use cross-validation instead.

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?

```
Investigating various classifiers...
Cross-validating Majority Classifer...
        -- train error: 0.240
        -- test error: 0.240
        -- f1 score: 0.760
Cross-validating Random Classifer...
        -- train error: 0.375
        -- test error: 0.382
        -- f1 score: 0.618
Cross-validating Decision Tree Classifer...
        -- train error: 0.000
        -- test error: 0.205
        -- f1 score: 0.795
Cross-validating 5-Nearest Neighbors Classifer...
        -- train error: 0.202
        -- test error: 0.259
        -- f1 score: 0.741
```

(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?



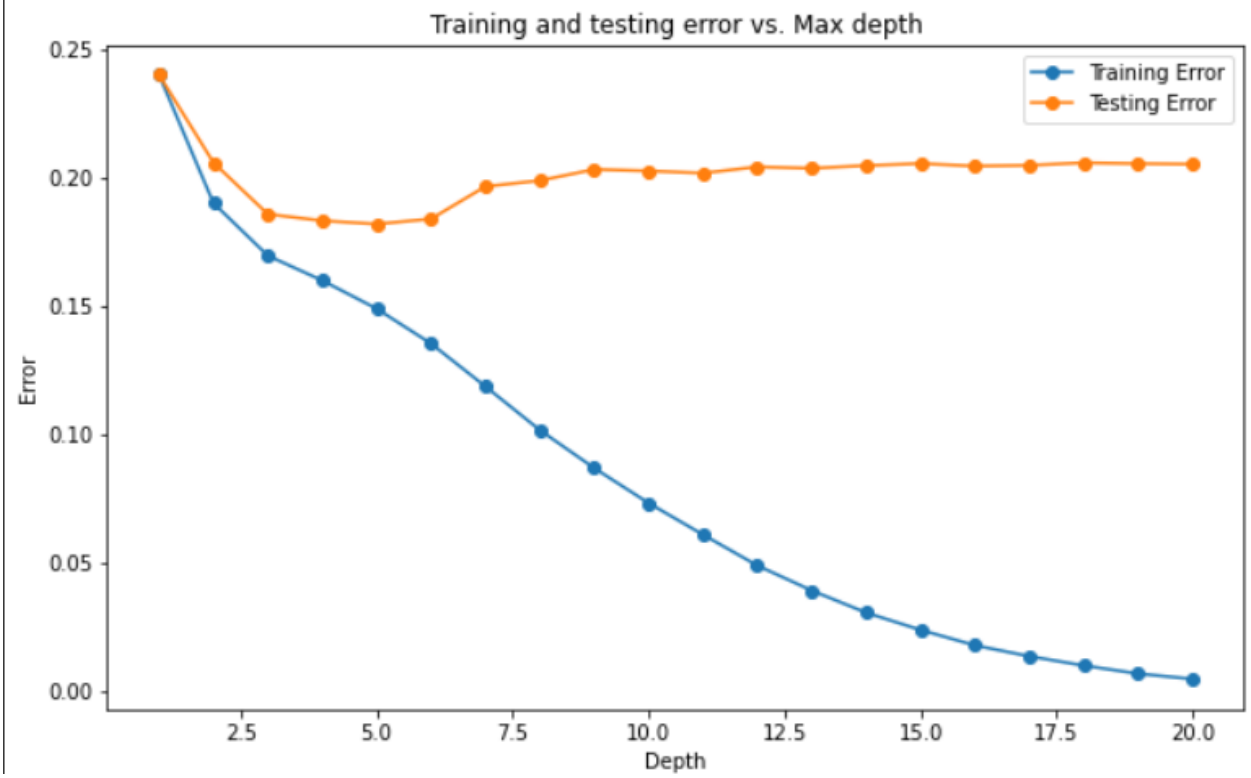

```
-- best k is 15 with an error of 0.2370000000000001
```

A small k leads to a high validation error; this is because the model is overfitting with a small k and so the model is not generalizable to unseen data. A large k also is not the best k because it leads to underfitting; the decision boundary is too smooth. The best k is 15, with an error of 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 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.

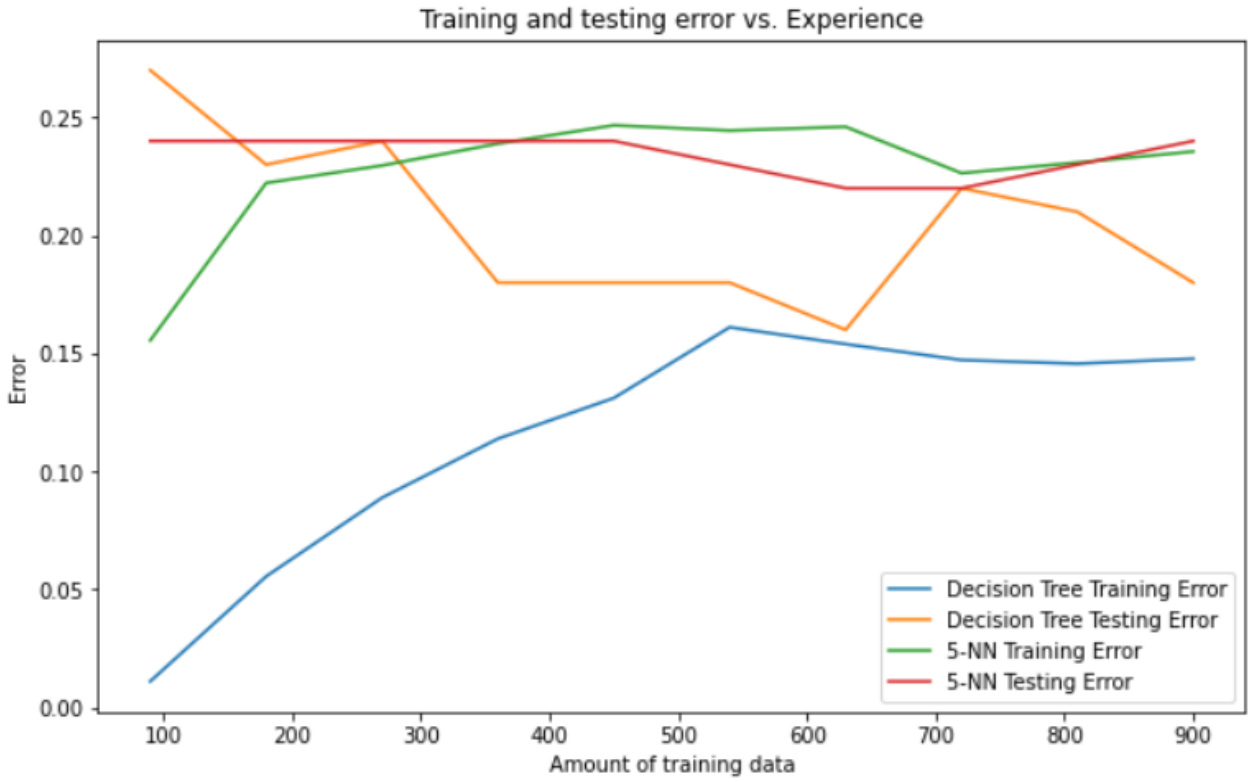
```
Investigating depths...
```



The best depth to use is 5; too large of a max depth leads to overfitting, specifically greater than a depth of 5. We can see that there is overfitting because as the max depth increases, the training error decreases while the test error increases and then stagnates at around 0.21.

(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.



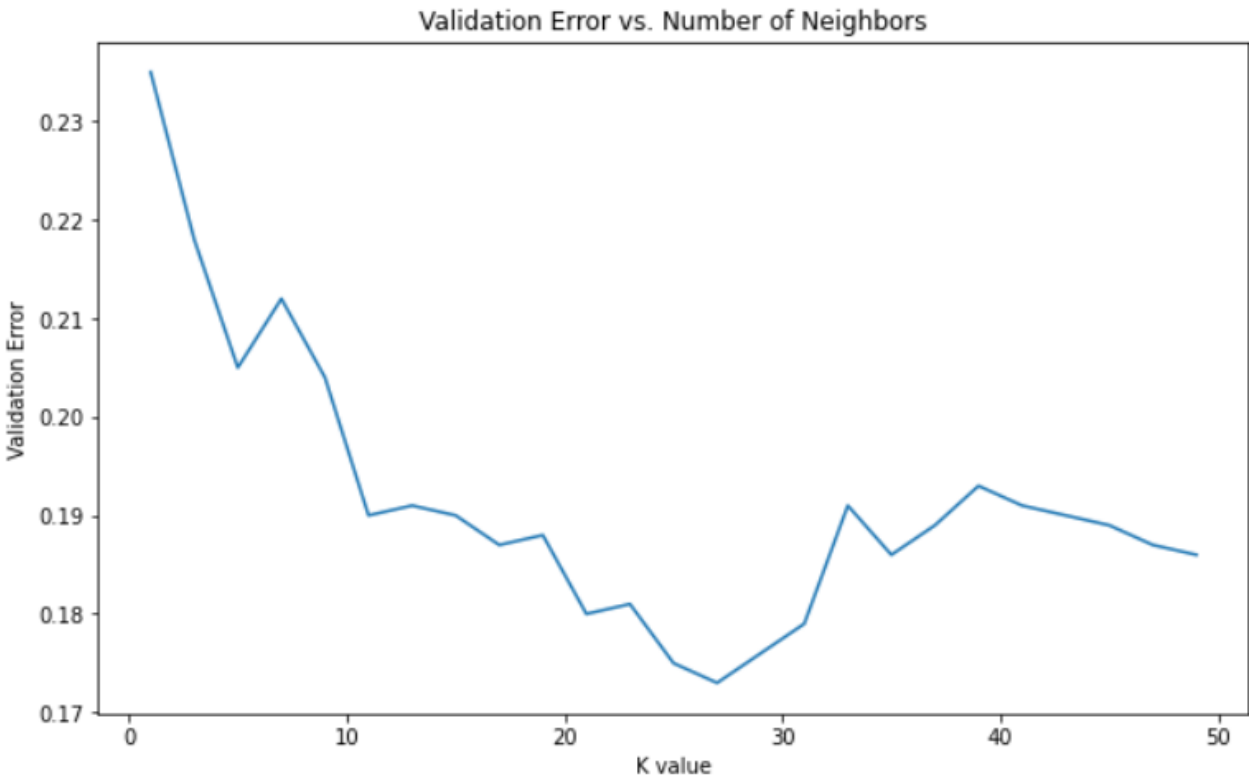
Training experience does not affect the KNN model as much as it affects the decision tree. For both KNN and decision tree, the training error increases as the amount of training data increases until over half of the training data is used, then stagnates at around 0.25 error for KNN and 0.15 error for the decision tree.
For the decision tree, the testing error decreases with more training data, from above 0.25 error to below 0.2.
For KNN, the testing error stays around the same level at just below 0.25.

(i) 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.

```
Classifying using Random...
        -- training error: 0.374
Classifying using Decision Tree...
        -- training error: 0.000
Classifying using 3-Nearest Neighbors...
        -- training error: 0.114
Classifying using 5-Nearest Neighbors...
        -- training error: 0.129
Classifying using 7-Nearest Neighbors...
        -- training error: 0.152
```

```
 Investigating various classifiers...
 Cross-validating Majority Classifer...
        -- train error: 0.240
        -- test error: 0.240
        -- f1 score: 0.760
 Cross-validating Random Classifer...
        -- train error: 0.375
        -- test error: 0.382
        -- f1 score: 0.618
 Cross-validating Decision Tree Classifer...
        -- train error: 0.000
        -- test error: 0.205
        -- f1 score: 0.795
 Cross-validating 5-Nearest Neighbors Classifer...
        -- train error: 0.133
        -- test error: 0.209
        -- f1 score: 0.791
```

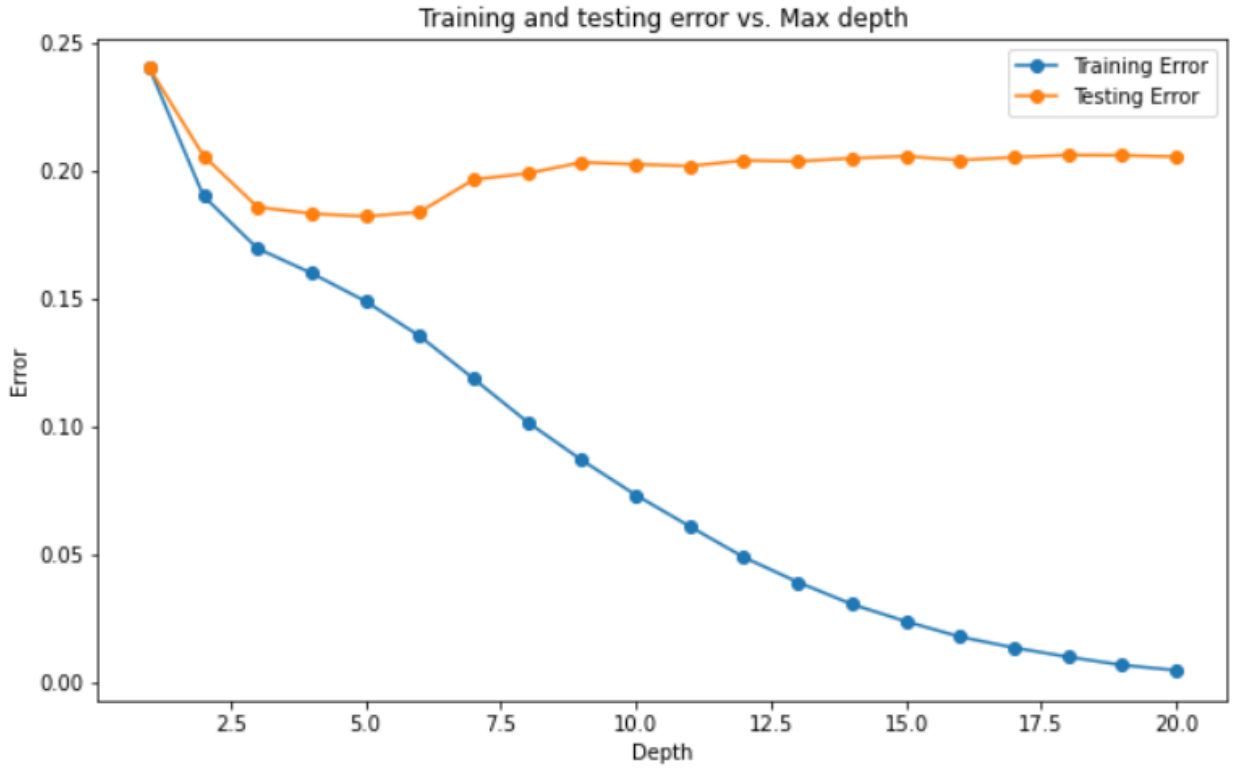
- As we can see, the performance of the random classifier and decision tree stays the same, but the performance of the KNN increases significantly (this is because KNN is negatively affected by data that is not normalized since unnormalized data affects how close the data is)




```
-- best k is 27 with an error of 0.17300000000000004
```

- The new best k is 27

```
Investigating depths...
```



- The decision tree performance is unaffected

- The KNN performance differs in that both the training and test errors are lower than what they were prior to the data being normalized
- Instead of increasing and stagnating, the KNN training error stays at around the same level as it started
- Instead of stagnating at around 0.25, the KNN testing error starts at just below 0.25 and decreases to around 0.15