CM146, Fall 2020
Problem Set 1: Decision trees and k-Nearest Neighbors
Due Oct. 29, 2020 at 11:59 pm

Ian Conceicao, ID:505153981

Submission instructions

- Submit your solutions electronically on the course Gradescope site as PDF files.
- Please package your code (.py) for Problem 5 and submit it to CCLE.
- If you plan to typeset your solutions, please use the LaTeX solution template. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app.

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

1 Splitting Heuristic for Decision Trees [20 pts]

Recall that the ID3 algorithm iteratively grows a decision tree from the root downwards. On each iteration, the algorithm replaces one leaf node with an internal node that splits the data based on one decision attribute (or feature). In particular, the ID3 algorithm chooses the split that reduces the entropy the most, but there are other choices. For example, since our goal in the end is to have the lowest error, why not instead choose the split that reduces error the most? In this problem, we will explore one reason why reducing entropy is a better criterion.

Consider the following simple setting. Let us suppose each example is described by n boolean features: $X = \langle X_1, \ldots, X_n \rangle$, where $X_i \in \{0, 1\}$, and where $n \geq 4$. Furthermore, the target function to be learned is $f : X \rightarrow Y$, where $Y = X_1 \vee X_2 \vee X_3$. 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) **(5 pts)** How many mistakes does the best 1-leaf decision tree make over the 2^n training examples?https://www.overleaf.com/project/5f9785dcd5e4ea0001d6442e (The 1-leaf decision tree does not split the data even once. Make sure you answer for the general case when $n \geq 4$.)

Solution: The best 1-leaf solution has Y=1 for every input. This solution is only wrong when X_1, X_2, and X_3 are all 0. This occurs $2^{(n-3)}$ number of times because 2^3 number of possible permutations are fixed.

(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?

Solution: Yes. The internal node $X_1 = 1$ where $True$ is the $Y = 1$ leaf and $False$ is the $Y = 0$ leaf would lead to 0 mistakes. Because our boolean operation: "$Y = 1$ if $X_1 = 1$ or $X_2 = 1$ or $X_3 = 1$, and $Y = 0$ otherwise" will always return 1 if $X_1 = 1$, no matter the size of n. We will have 0 errors in our decision tree.

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

Solution: The entropy is: 0.544 We arrive to that solution because $E[S] = -P_+ log(P_+) - P_- log(P_-)$. We find P_+ is 14/16 and P_- is 2/16

(d) **(5 pts)** 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?

Solution: Yes, the split described in part b would be an entropy of 0. In our definition, $0log(0) = 0$. All the 1's and 0's are completely separated.

2 Entropy and Information [5 pts]

The entropy of a Bernoulli (Boolean 0/1) random variable X with $p(X = 1) = q$ is given by

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

Suppose that a set S of examples contains p positive examples and n negative examples. The entropy of S is defined as $H(S) = B\left(\frac{p}{p+n}\right)$.

(a) **(5 pts)** Based on an attribute X_j, we split our examples into k disjoint subsets S_k, with p_k positive and n_k negative examples in each. If the ratio $\frac{p_k}{p_k+n_k}$ is the same for all k, show that the information gain of this attribute is 0.

Solution: The current entropy is: $-\frac{p}{p+n}\log(\frac{p}{p+n}) - (1 - \frac{p}{p+n})\log(1 - \frac{p}{p+n})$. Each subset has entropy of $-\frac{p_k}{p_k+n_k}\log(\frac{p_k}{p_k+n_k}) - (1 - \frac{p_k}{p_k+n_k})\log(1 - \frac{p_k}{p_k+n_k})$ Taking the weighted average would be same as multiplying this expression by k, and then diving it by k, so therefore this is the weighted average. If the ratio of $\frac{p_k}{p_k+n_k}$ is the same for every subtree, then $\frac{p_k}{p_k+n_k}$ must equal $\frac{p}{p+n}$. Information gain is the entropy before, subtracted by the weighted average, therefore information gain is 0 because we are subtracting a value by itself.

3 k-Nearest Neighbors and Cross-validation [10 pts]

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

(a) **(2.5 pts)** What will be the label of point (7,3) in Fig 1 using K-NN algorithm with majority voting when K=3?

Solution: Negative. Using Euclidian distance, the 3 closest points are $(7, 3)$, $(7, 2)$ and $(8, 3)$. 2 of these points are negative and 1 is positive, therefore the label will be negative.

(b) **(2.5 pts)** What value of k minimizes the training set error for this dataset? What is the resulting training error?

Solution: $K = 1$. Because a feature can be it's own nearest neighbor, in the training set, if only the closest point is considered, then each training point only looks at itself, and the resulting training error is 0

(c) **(2.5 pts)** Why might using too large values k be bad in this dataset? Why might too small values of k also be bad?

Solution: If K is too large, then the model will be underfitted. Labels that are farther away will be considered when they should not be, and spacial meaning will be lost. Also, if K is too small, the model will be overfitted. The model, as explained in part b, will perform extremely

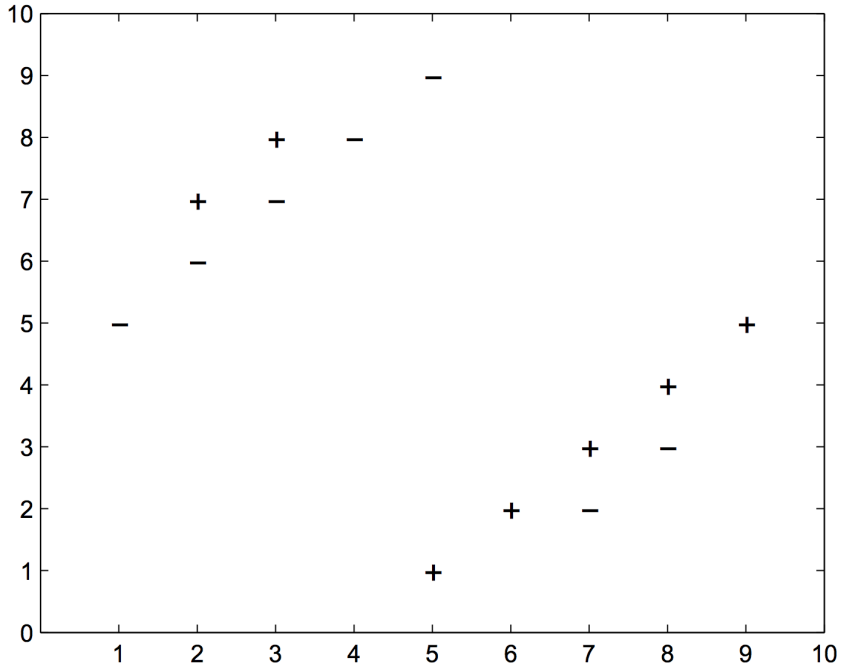


Figure 1: Dataset for KNN binary classification task.

well for the training data, but will not consider enough data to fit test data or other data well.

(d) **(2.5 pts)** What value of k minimizes leave-one-out cross-validation error for this dataset? What is the resulting error?

Solution: $K = 5$ minimizes leave-one-out cross-validation error for this dataset. The error would be $4/14$ because points $(2, 7), (4, 8), (7, 2), (8, 3)$ would return errors but the others would not.

4 Decision Tree [15 pts]

In a binary classification problem, there are 4000 examples in the class with label 1 and 8000 examples in the class with label 0. Recall that the information gain for target label Y and feature X is defined as $Gain = H[Y] - H[Y|X]$, where $H[Y] = -E[\log_2 P(Y)]$ is the entropy.

(a) **(2 pts)** What is the entropy of the class variable Y?

Solution: Remember Entropy is: $-\frac{p}{p+n}\log(\frac{p}{p+n}) - (1 - \frac{p}{p+n})\log(1 - \frac{p}{p+n})$ Let $p = 4000$ and $n = 8000$. Entropy is 0.918

(b) **(5 pts)** Let's consider a binary feature A for this problem. In the negative class (with label 0), the number of instances that have $A = 1$ and $A = 0$ respectively: $(4000, 4000)$. In the positive class (with label 1), these numbers are: $(4000, 0)$. Write down conditional entropy

and information gain of A relative to Y?

Solution: When A=1: $-\frac{4000}{8000}\log\frac{4000}{8000}-\frac{4000}{8000}\log\frac{4000}{8000}$ Without calculating this, we can observe entropy is 1, because it is an even split. When A=0: $-\frac{4000}{4000}\log\frac{4000}{4000}-\frac{4000}{4000}\log\frac{4000}{4000}$ Without calculating this, we can observe entropy is 0, because the output is homogeneous. Next, we take the weighted average: $\frac{8{,}000}{12{,}0000}*1+\frac{4{,}000}{12{,}0000}*0=\frac{2}{3}$ This value is our *conditional entropy.* Therefore our *information gain* is $\frac{2}{3}$ because $1-\frac{2}{3}=\frac{1}{3}$

(c) **(5 pts)** Let's consider another binary feature B. In the negative class (with label 0) , the number of instances that have $B=0$ and $B=1$ respectively are: $(6000, 2000)$. In the positive class (with label 1), these numbers are: $(3000, 1000)$. Write down conditional entropy and information gain of B relative to Y?

Solution: When B=1: $-\frac{6000}{9000}\log\frac{6000}{9000}-\frac{3000}{9000}\log\frac{3000}{9000}$

Let this be known as $E_1=0.918$.

When A=0: $-\frac{2000}{3000}\log\frac{22000}{3000}-\frac{1000}{3000}\log\frac{1000}{3000}$

Let this be known as $E_2=0.918$

Next, we take the weighted average: $\frac{9{,}000}{12{,}0000}*0.918+\frac{3{,}000}{12{,}0000}*0.918=0.918$. This value is our *conditional entropy.* Therefore our *information gain* is 0.082 because $1-0.918=0.082$

(d) **(3 pts)** Using information gain, which attribute will the ID3 decision tree learning algorithm choose at first?

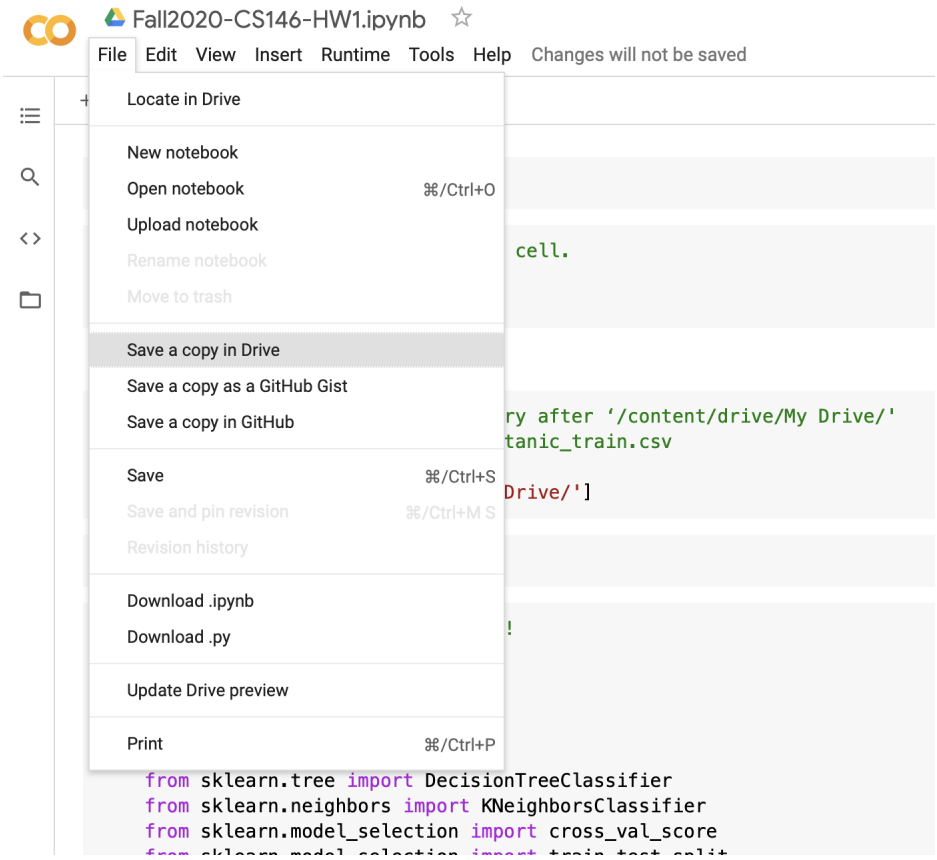
Solution: The ID3 decision tree learning algorithm will choose attribute A because it has a larger information gain.

5 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.

Next, for all the coding, please refer to the following colab notebook Fall2020-CS146-HW1.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've just uploaded.

The notebook has marked blocks where you need to code.

$========= TODO : START =========$

$========= TODO : END ==========$

Submission instructions for programming problems

- Please export the notebook to a `.py` file by clicking the "File" → "Download.py" and upload to CCLE.

  Your code should be commented appropriately. The most important things:
  - Your name and the assignment number should be at the top of each file.
  - Each class and method should have an appropriate doctsring.
  - If anything is complicated, it should include some comments.

  There are many possible ways to approach the programming portion of this assignment, which makes code style and comments very important so that staff can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.

- Please submit all the plots and the rest of the solutions (other than codes) to Gradescope

For the questions please read below.

5.1 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 (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)

Solution:

i. The workclass for most people who earn less than 50k is 1. Very few people are earning more than 50k and are in the 0 workclass group

ii. There are many eduacated people who earn less than 50k and more than 50k

iii. Many people who earn less than 50k are in the 3 marital status group

iv. People who earn less are more likely to have a fewer occupation ranking.

v. Very few people who earn more than 50k are in the 4 group for relationships

vi. People in the 1 group of race earn less than people who are not

vii. People who are native to 0 earn more than people who are native other places.

viii. Earning over 50k linearly decreases with the age rating

ix. The lower the fnl more likely they are earn to more money it appears

x. the education nums is most lkely to be around 9-10 for people who less than 50k. And more likely to be 14 for those who earn more

xi. The capital gain of almost everyone who earns less than 50k is low.

xii. People who have less than 50k have about the same capital-loss as those with greater than 50k

xiii. People who earn less than 50k are very likely to work 40 hours per week

xiv. It is tough to draw a conclusion about sex.

5.2 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.

(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:** The training error of this classifier is 0.000

(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:**

The following output shows the training error:

Classifying using k-Nearest Neighbors...

– training error with K=3: 0.153

– training error with K=5: 0.195

– training error with K=7: 0.195

(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 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?

Solution: The following are the training, test errors, and the f1 scores for all of my classifiers

– training error for RandomClassifier: 0.375
– testing error for RandomClassifier: 0.382
– F1 score for RandomClassifier: 0.618

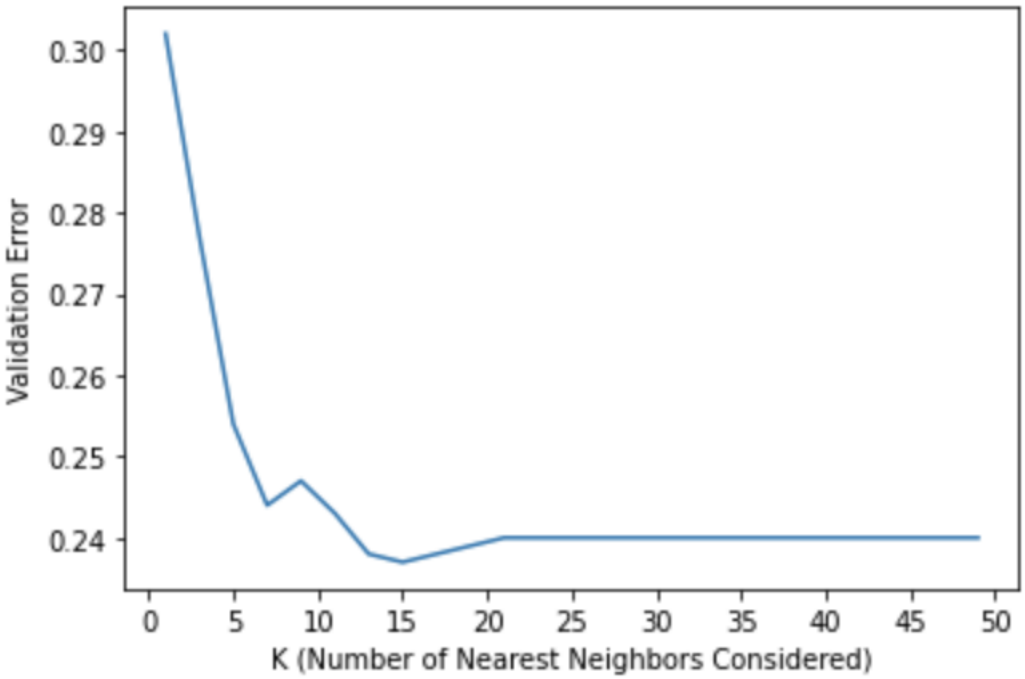
– training error for DecisionTree: 0.000
– testing error for DecisionTree: 0.205
– F1 score for DecisionTree: 0.795

– training error for 3NearestNeighbors: 0.157
– testing error for 3NearestNeighbors: 0.284
– F1 score for 3NearestNeighbors: 0.716

– training error for 5NearestNeighbors: 0.202
– testing error for 5NearestNeighbors: 0.259
– F1 score for 5NearestNeighbors: 0.741

– training error for 7NearestNeighbors: 0.202
– testing error for 7NearestNeighbors: 0.259
– F1 score for 7NearestNeighbors: 0.741

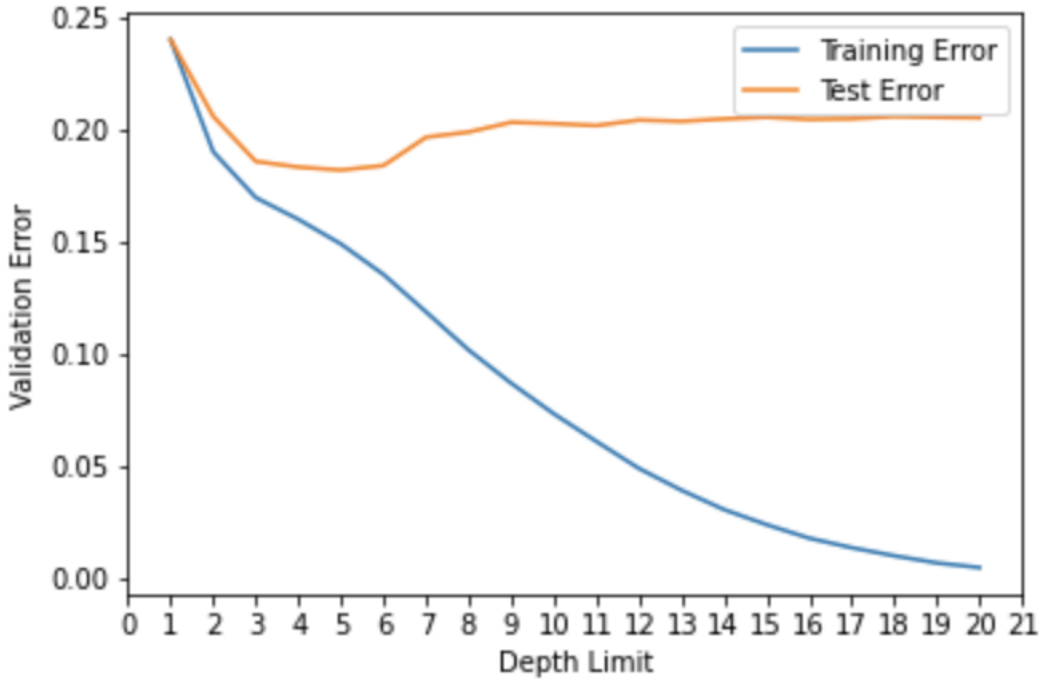
(f) **(5 pts)** One way to find out the best value of k for `KNeighborsClassifier` is n-fold cross validation. Find out the best value of k using 10-fold cross validation. You may find the `cross_val_score(...)` from `scikit-learn` helpful. Run 10-fold cross validation for all odd numbers ranging from 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: I noticed the gain from increasing the number of K's increased greatly until about k =15, where the error then increase and eventually levelled out. The least error was at $K = 15$, and therefore that is the best K value

(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, $1, 2, \ldots, 20$. 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: Yes I see overfitting of the data when the tree is allowed too great of a depth. The best depth for the test data is 5. Any depth greater than 5, continues to decrease the validation error for the training data, but either increases or does not change the test data. Therefore after a depth of 5, the model is overfitting the test data, and losing the ability to generalize to new data.

(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 and do the following experiments considering the 90% fraction as training and 10% for testing.

Run experiments for the decision tree and k-nearest neighbors classifier with the best depth limit and k value you found above. This time, vary the amount of training data by starting with splits of 0.10 (10% of the data from 90% fraction) and working up to full size 1.00 (100% of the data from 90% fraction) in increments of 0.10. Then plot the decision tree and k-nearest neighbors training and test error against the amount of training data. Include this plot in your writeup, and provide a 1-2 sentence description of your observations.

Solution: We notice that both models consistently performed better on training data than on testing data, which is to be expected. However, the more a model learned did not show to effect the error on the testing data at all. The larger a deicison tree's training set, the more trouble it had predicting the training set data, which make sense because there is more to learn.

(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 data normalization did not appear to effect the numbers for the decision tree. The decision (with unlimited depth) still had a training error of 0.000 and a testing error of 0.205. However it did increase its F1 score from 0.716 to 0.795, which is better because the greater the F1 score the best. The improvement is most strongly apparent in the errors for the K-Nearest algorithm. Every single error was decreased by about 0.5. It makes sense the K-Nearest algorithm would benefit from the data being normalized because it makes decisions based on distance between multidimensional coordinates, and dimensions having larger ranges and scales can give them more weight in the decisions.