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

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? (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: For the case of $n = 4$, the best 1-leaf node would make 2 mistakes for the cases $\{0, 0, 0, 0\}$ and $\{0, 0, 0, 1\}$. For the cases $n > 4$, there would be a total of $2^(n - 3)$ errors.

(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: No; no matter which single attribute we split on, among all possible inputs, the number of errors in total for both resulting leaf nodes would still be 2.

(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 of the output label Y would be $H(Y) = -(2/16)log_2(2/16) - (14/16)log_2(14/16) = 0.544$.

(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: We can split the set by any of X_1, X_2, or X_3. The resulting conditional entropy of Y would be $H(Y_new) = (1/2)(-(2/8)log_2(2/8) - (6/8)log_2(6/8)) + (1/2)(-(0/8)log_2(0/8) - (8/8)log_2(8/8)) = 0.406$.

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

(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: Say that the percentage of positive examples of the original entire set S is Q. The entropy of S would be $H(S) = B(Q)$. IF S were to be split into k subsets while maintaining the exact same percentage of positive examples, the entropy of each subset would still be $B(Q)$, no matter what the size of each subset is. The total conditional entropy, the sum of the entropies of each subset, would still end up being $B(Q)$, thus the information gain would be 0.

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.

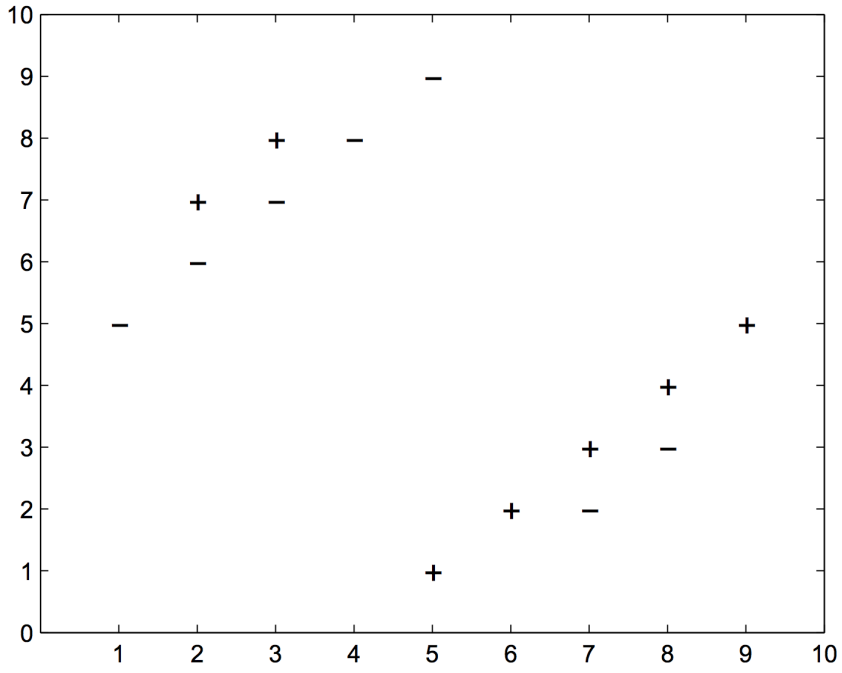


Figure 1: Dataset for KNN binary classification task.

(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: positive (+)

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

Solution: $K = 1$. The resulting training error would be 0 as all the points would be self-labelling.

(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 the value of k be too small (like $k = 1$), we may run into the problem of over-fitting (little generalization). If we set the value of k to be too large (like $k = 12$ in this case), we would run into problems of under-fitting (will generalize the most of, if not the entire, data set).

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

Solution: The values $k = 5$ and $k = 7$ both minimize leave-one-out cross-validation error. The resulting error is (4/12) for both.

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: $H(Y) = -(1/3)log_2(1/3) - (2/3)log_2(2/3) = 0.918$

(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: $H(Y|A) = (8000/12000)(1) + (4000/12000)(0) = 0.667$ and $G(Y, A) = 0.918 - 0.667 = 0.251$

(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: $H(Y|B) = (9000/12000)(-(6000/9000)log_2(6000/9000) - (3000/9000)log_2(3000/9000)) + (3000/12000)(-(2000/3000)log_2(2000/3000) - (1000/3000)log_2(1000/3000)) = 0.861$ and $G(Y, B) = 0.918 - 0.861 = 0.057$

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

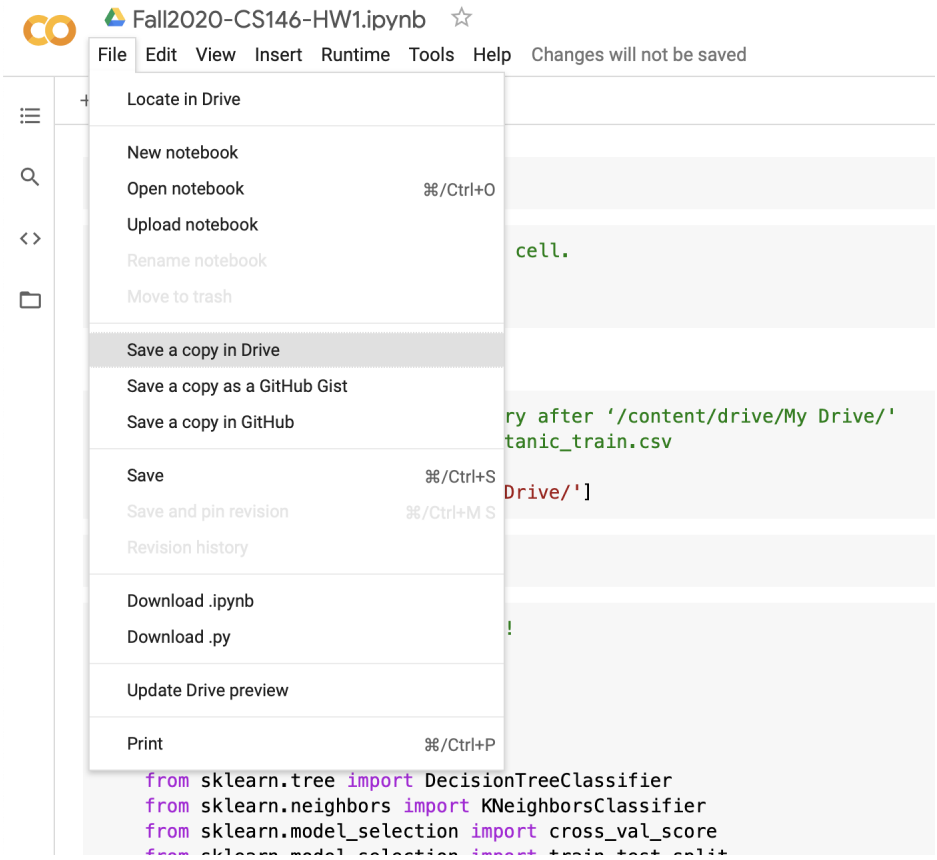
Solution: Attribute B

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:

workclass: There seems to be no samples with a workclass of 0 with values <50k, and a significantly larger proportion of those wit a workclass of 1 (among which 3/4 of the samples are <50k). The remaining samples with workclass >1 are about equal and small in number.

education: There seems to be a large number of samples with education of 0 or 1 (among which 2/3 and 4/5 of the samples are <50k, respectively). The remaining samples with education >1 are about equal and small in number.

marital-status: There seems to be a large number of samples with values of 1, 2, or 3 (with still most samples being <50k). The remaining samples are smaller in number and seem highly variable (0 and 5, none for 6, all of which are <50k).

occupation: There seems to be a large number of samples with values of 0, 1, or 2 (with

a large peak at 4 for <50k and peaks at 4 and 6 for ≥50k).

relationship: For >50k, the curve seems to peak around 3 or 4, while for ≥50k, the curve seems to peak around 3 only. The <50k values seems to be larger than the /*geq*50k values for the same given relationship values.

race: For both >50k and /*geq*50k, the curve seems to peak around 0. The <50k values seems to be larger than the /*geq*50k values for the same given relationship values. There also seems to be a slight spike at 5 for <50k.

native-country: For both >50k and /*geq*50k, the very sharp curve seems to peak around 0. The <50k value seems to be significantly larger than the /*geq*50k value at 0.

age: For >50k, there is a downward curve from 0, while for ≥50k, the curve seems to peak around 40.

fnlwgt; For both >50k and /*geq*50k, the curve seems to peak around 0.2. The <50k values seems to be larger than the /*geq*50k values for the same given relationship values.

education-nums: For >50k, the curve peaks strongly at 10, while for ≥50k, the curve seems to peak around 16.

capital-gain: For both >50k and /*geq*50k, the very sharp curve seems to peak around 0. The <50k value seems to be significantly larger than the /*geq*50k value at 0.

capital-loss: For both >50k and /*geq*50k, the very sharp curve seems to peak around 0. The <50k value seems to be significantly larger than the /*geq*50k value at 0.

hours-per-week: For >50k, the curve peaks strongly at 40, while for ≥50k, the curve seems to peak around 40, but not as strongly. The <50k values seems to be larger than the /*geq*50k values for the same given relationship values.

sex: There are only 2 values, 0 and 1. For both >50k and /*geq*50k, their values for 0 are greater than their corresponding values for 1.

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:** Our training error using the DecisionTreeClassifier is 0.000. This makes sense as, without any further restrictions like number of layers, the decision tree will attempt to make a model that will fit for all samples in the training set (given that no sample can be classified as both, the error has to be 0).

(d) **(5 pts)** Similar to the previous question, train and evaluate a `KNeighborsClassifier` (using the class from `scikit-learn` and referring to the documentation as needed). Use k=3, 5 and 7 as the number of neighbors and report the training error of this classifier. **Solution:** Again, using "leave-one-out", our error for 3NN is 0.153, for 5NN it is 0.195, and for 7NN it is 0.213.

(e) **(10 pts)** So far, we have looked only at training error, but as we learned in class, training error is a poor metric for evaluating classifiers. Let's use cross-validation instead.

Implement the missing portions of `error(...)` according to the provided specifications. You may find it helpful to use `StratifiedShuffleSplit(...)` from `scikit-learn`. To ensure that we always get the same splits across different runs (and thus can compare the classifier results), set the `random_state` parameter to be the same (e.g., 0).

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

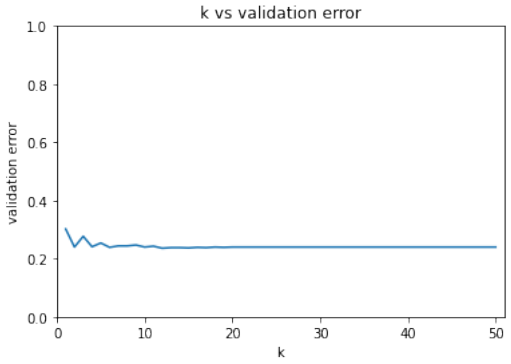
[1] Note that almost all of the model techniques in `scikit-learn` share a few common named functions, once they are initialized. You can always find out more about them in the documentation for each model. These are `some-model-name.fit(...)`, `some-model-name.predict(...)`, and `some-model-name.score(...)`.

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:
MajorityVoteClassifier:
– training error: 0.240
– test error: 0.240
RandomClassifier:
– training error: 0.375
– test error: 0.382
DecisionTreeClassifier:
– training error: 0.000
– test error: 0.205
KNeighborsClassifier:
– training error: 0.202
– test error: 0.259

(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:



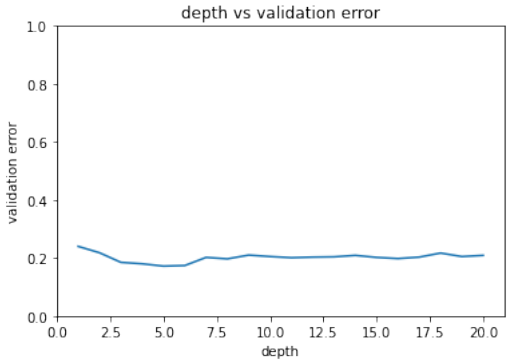
There doesn't seem to be too much of a change between using 1 and using 50 (or any other value in-between). The best value for k is 12 (with a validation error of 0.236, which is only slightly less than the other values).

(g) **(5 pts)** One problem with decision trees is that they can *overfit* to training data, yielding complex classifiers that do not generalize well to new data. Let's see whether this is the case.

One way to prevent decision trees from overfitting is to limit their depth. Repeat your cross-validation experiments but for increasing depth limits, specifically, $1, 2, \ldots, 20$. 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:

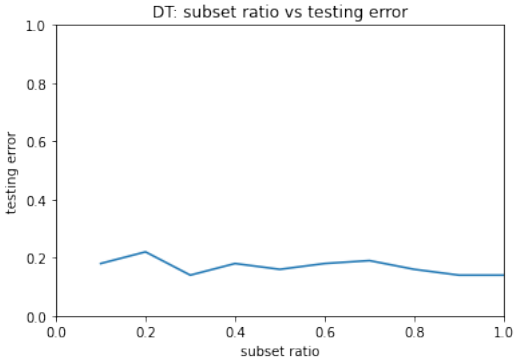


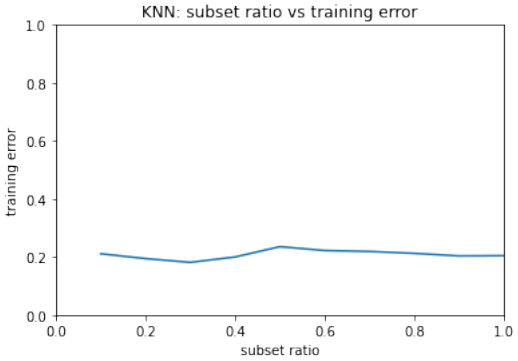
According to the data, the best maximum depth to use is 5 (although there doesn't seem to be too much variation). There does seem to be some over-fitting, as the error rate does increase to the original value (for depth = 1) after the lowest point at depth = 5.

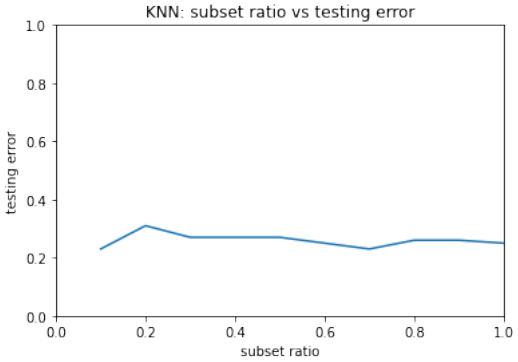
(h) **(5 pts)** Another useful tool for evaluating classifiers is *learning curves*, which show how classifier performance (e.g. error) relates to experience (e.g. amount of training data). For this experiment, first generate a random 90/10 split of the training data and do the following experiments 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:







For training error, the decision tree does seem to have an increasing training error (as more samples would mean more difficulty in generalizing in a decision tree) while KNN seems to have a relatively constant training error. For test error, the decision tree does seem to have a slightly decreasing curve, while KNN also seems have it be mostly constant.

(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: Most of the errors and curves from the previous parts seem to be about the same. The most noticeable differences are the decreased training errors for KNN for all values of k, and the slight flattening for the learning curves (both training and test errors).