CM146, Winter 2023
Problem Set 1: Decision trees, Nearest neighbors
Due Feb 3, 2023 at 11:59 pm PST

Submission instructions

- Submit your solutions electronically on the course Gradescope. Please upload as either a PDF document or Images (.jpeg, .png).
- 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.
- On Gradescope, please carefully mark each page as the corresponding problem or it will not be scored. Responses which are blurry, cut off, or unreadable will also not be scored.
- Submitting code is not required. Please submit only the results and responses related to the coding portion.

Parts of this assignment are adapted from course material by Andrea Danyluk (Williams), Tom Mitchell and Maria-Florina Balcan (CMU), Stuart Russell (UC Berkeley) 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 $f : X \to Y$, where $Y = X_1 \vee X_2 \vee X_3$. That is, $Y = 1$ if $X_1 = 1$ or $X_2 = 1$ or $X_3 = 1$, and $Y = 0$ otherwise (X_i for $i \geq 4$ is not considered). Suppose that your training data contains all of the 2^n possible examples, each labeled by f. For example, when $n = 4$, the data set would be

X_1	X_2	X_3	X_4	Y
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	0	1
0	0	1	0	1
1	0	1	0	1
0	1	1	0	1
1	1	1	0	1

X_1	X_2	X_3	X_4	Y
0	0	0	1	0
1	0	0	1	1
0	1	0	1	1
1	1	0	1	1
0	0	1	1	1
1	0	1	1	1
0	1	1	1	1
1	1	1	1	1

(a) How many mistakes does the best 1-leaf decision tree make over the 2^n training examples? (The 1-leaf decision tree does not split the data even once. Justify and answer for the general case when $n \geq 4$ for full credit.)

Solution: A sample X is considered as 0 iff all its features X_1, X_2, and X_3 are equal to 0. The number of such binary vectors can be calculated as 2^{n-3} because there are two possibilities for each of the remaining $n - 3$ features. Since 2^n is much larger than 2^{n-3}, the (best) 1-leaf decision tree that predicts 1 for every input will result in 2^{n-3} mistakes, which means it will make an error $2^{n-3}/2^n = 1/8^{th}$ of the time.

(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.)

Solution: No matter which root variable is chosen, the error rate will stay the same at $\frac{1}{8}$.

Case 1: If a split is made on a variable X_i with $i \geq 4$, it will divide the data so that the ratio of ones in each leaf is $\frac{7}{8}$. Both leaves will be predicted as 1, resulting in the same number of errors as the single-leaf tree that always predicts 1.

Case 2: Now, if the split is made on X_1, X_2, or X_3, the data will be divided into two leaves with one containing only 1's and the other having a 1's ratio of $\frac{3}{4}$. In this instance, both leaves will be predicted as 1, leading to the same number of errors as the single-leaf tree that always predicts 1.

(c) What is the entropy of the label Y?

Solution:

$$H(X) = \frac{1}{8}\log(8) + \frac{7}{8}\log\left(\frac{8}{7}\right) = 0.543 \text{ bits, since } Y \sim \text{Bernoulli}(\frac{7}{8})$$

(d) Is there a split that reduces the entropy of Y by a non-zero amount? If so, what is it, and what is the resulting conditional entropy of Y given this split? (Again, as in lecture, you should restrict your attention to splits that consider a single attribute. Please use logarithm in base 2 to report Entropy.)

Solution: Yes, by splitting along any of X_1, X_2, or X_3. Using conditional entropy for X_i, $H(Y \mid X_i) = \frac{1}{2}[0] + \frac{1}{2}\left[\frac{1}{4}\log(4) + \frac{3}{4}\log\left(\frac{4}{3}\right)\right] = 0.406$ bits.

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 $p = n$.

Solution: Let $\frac{p}{p+n} = q$. We take the first derivative of B with respect to q (use chain rule) and apply the first derivative test:

$\frac{dB}{dq} = -\log q - q\left(\frac{q}{\ln 2}\right) - \left(-\log(1-q) + (1-q)\left(\frac{1}{(1-q)(\ln 2)(-1)}\right)\right) = -\log q + \log(1-q) = 0$

This gives us $q = 1-q$, providing $q = 0.5$ as a critical point. Now, $B'(0.3) > 0$ and $B'(0.7) < 0$, so $g = 0.5$ is a maximum. The value is $B(0.5) = 1$. We then check the endpoints: the variable q is restricted to the range $[0, 1]$, and we have $B(0) = 0$, and $B(1) = 0$. The only critical point in the interval $[0, 1]$ is 0.5, and it is a maximum, so thus, $0 \leq B(q) \leq 1$, which gives $0 \leq H(S) \leq 1$, as desired.

Then, when $p = n$, $\frac{p}{p+n} = \frac{p}{2p} = 0.5$, and according to the above, $H(S = 0.5) = 1$, as desired.

(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.

Solution: Gain is defined as $H[S]$ - $H[S|split]$. Taking advantage of this since $p = \sum_k p_k$ and $n = \sum_k n_k$, if $p_k/(p_k + n_k)$ is the same for all k, it must be that $p_k/(p_k + n_k) = p/(p + n) = q$ for all k. Then the entropy of S is $B(p/(p + n))$ and the weighted average entropy of its children S_k is: $\sum_k \frac{p_k+n_k}{p+n} B(p_k/(p_k + n_k)) = \frac{p+n}{p+n} B(p/(p + n)) = B(p/(p + n))$.

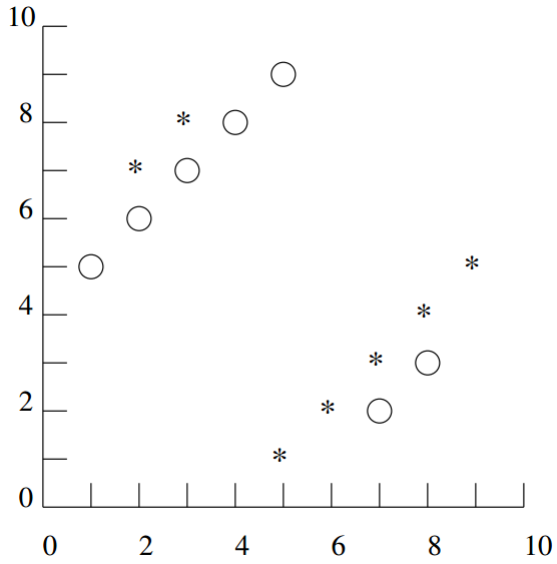
Substituting this expression and the expression for $H[S]$ into the gain equation gives:

$Gain = B(p/(p + n)) - B(p/(p + n)) = 0.$

Thus, we have shown that this particular split results in no information gain.

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?

Solution: The value $k = 1$ minimizes the training set error, and it results in an error of 0. This is because we are simply assigning a label based on the label that it has, as it is its own closest neighbor.

This is not a reasonable estimate of the test set error because we are overfitting the data and not taking any of the data's neighbors into account. When the model overfits the training data, it may increase training set accuracy while lowering test set accuracy. There is a chance that $k = 1$, despite fitting the given data, may not generalize well to new data.

(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?

Solution: Using either $k = 5$ or $k = 7$ minimizes the LOOCV error; for both values of k, we get 4 misclassified points and 10 correctly classified points, resulting in an error of $\frac{4}{14} = 0.286$.

Cross-validation is a better measure of test set performance because it measures performance on data other than the data used to train the model. Essentially, we are cycling through which data is used as test data (and the complement being used as training data), thereby removing any bias that could result from training or testing on a particular set of 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?

Solution: If we use $k = 0$, the error is 0. If we use $k = 1$, we get an error of $\frac{10}{14} = 0.714$. If we use $k = 13$, we get an error of $\frac{14}{14} = 1$. Asserting values for k that are too small or too large reduces test set accuracy because the former leads to overfitting and the latter to underfitting, or simply, the degeneration of the problem into a simple majority vote, potentially causing all points to be misclassified (as demonstrated in the example).

4 Programming exercise : Applying decision trees and k-nearest neighbors [50 pts]

Introduction[1]

This data was extracted from the 1994 Census bureau database by Ronny Kohavi and Barry Becker. For computational reasons, we have already extracted a relatively clean subset of the data for this HW. The prediction task is to determine whether a person makes over $50K a year.

In this problem, we ask you to complete the analysis of what sorts of people were likely to earn more than $50K a year. In particular, we ask you to apply the tools of machine learning to predict which individuals are more likely to have high income.

Starter Files

---

code and data

- Code: CS146-Winter2023-PS1.ipynb
- Data: nutil.py and adult_subsample.csv

documentation

- Decision Tree Classifier:
  http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- K-Nearest Neighbor Classifier:
  http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- Cross-Validation:
  https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html
- Metrics:
  http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html,
  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html?highlight=f1%20score#sklearn.metrics.f1_score
- Data Preprocessing:
  https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html?highlight=standardscaler#sklearn.preprocessing.StandardScaler

---

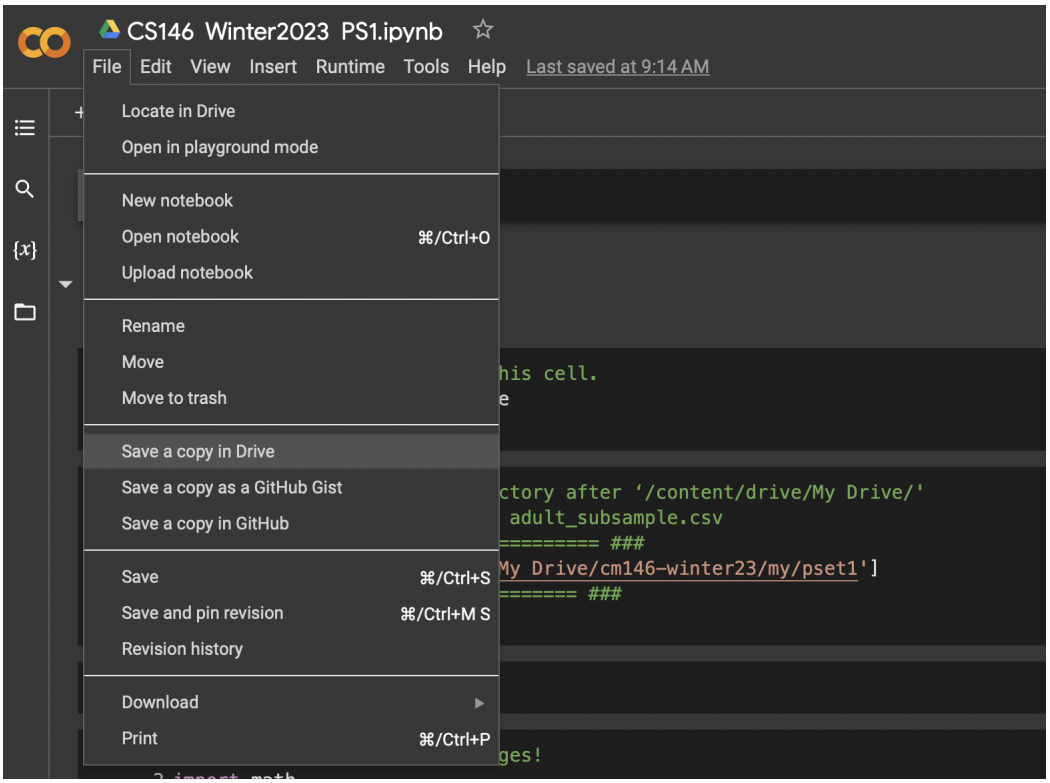
Note that any portions of the code that you must modify have been indicated with `TODO`. Do not change any code outside of these blocks.

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.

[1] This assignment is adapted from the UCI Machine learning repository, available at https://archive.ics.uci.edu/ml/datasets/adult.

Next, for all the coding, please refer to the following colab notebook CS146-Winter2023-PS1.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 have just uploaded.

The notebook has marked blocks where you need to code.

$$\#\#\# \text{ ========= } TODO : START \text{ ========= } \#\#\#$$

$$\#\#\# \text{ ========= } TODO : END \text{ ========== } \#\#\#$$

Submission instructions

- Only provide answers and plots through gradescope. Do not submit code.

For the questions please read below.

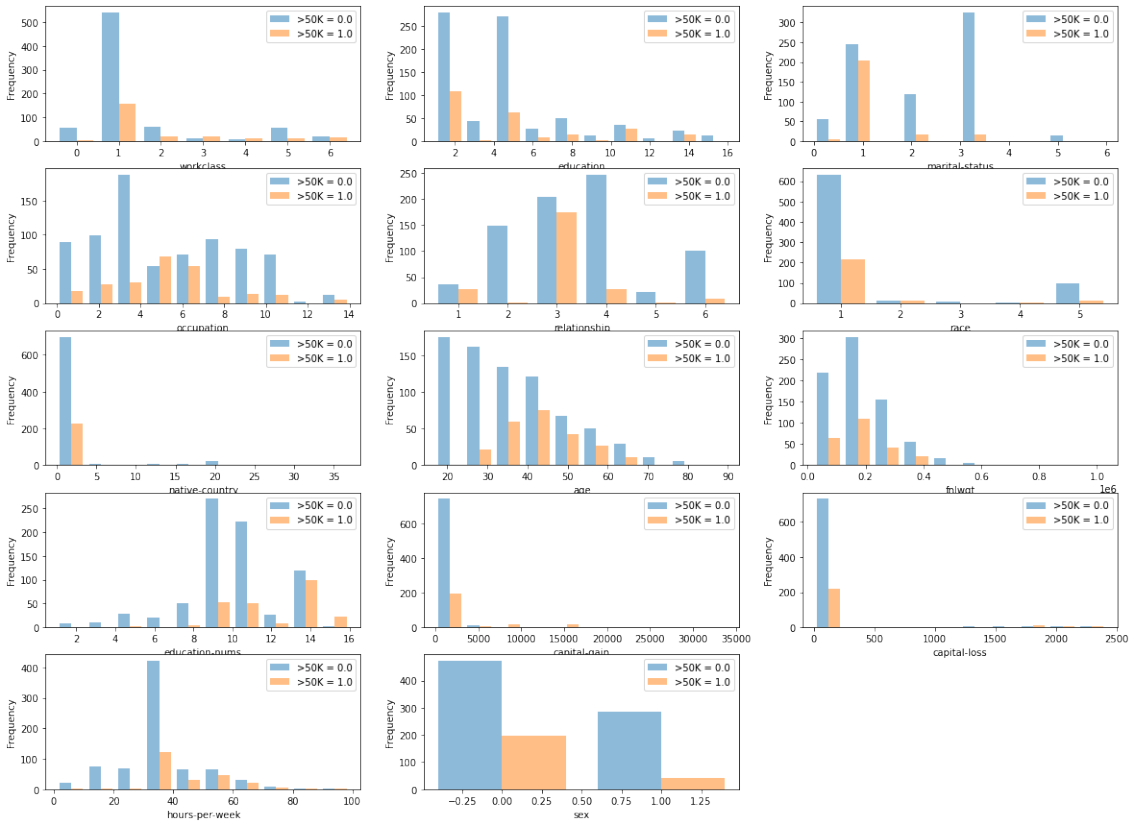
4.1 Visualization [5 pts]

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 by running the function plot_histograms in the notebook. This should produce fourteen plots, one for each feature, and each plot should have two overlapping histograms, with the color of the histogram indicating the class. For each feature, what trends do you observe in the data if any? (Please only describe the general trend. No need for more than two sentences per feature)

Solution:



The following analysis is conducted keeping in the mind the data markers from the linked resource:

- Workclass: Most people are self-employed. Government workers are most likely to make >50k.
- Education: The more educated a person is, the more likely they are to make >50k.
- Marital Status: Divorced people are the most likely to make >50k.
- Occupation: Sales and exec-managerial are most likely to make >50k.
- Relationship: Husbands are more likely to make >50k.
- Race: Most people in this dataset are white, but Asians have a high fraction of people making >50k.
- Native Country: Most people in this study are from the USA. Most of these people make <50k.
- Age: People in the 40-50 age group are most likely to make >50k.
- Fnlwgt: The fraction of people making >50k is relatively constant. There are more people who make <50k.
- Education Nums: Most people have 8-10 years of education. People with ¿14 years of education have higher chances of making >50k.
- Capital Gain: Most people have a low capital gain. People with high capital gain tend to make >50k.
- Capital Loss: Most people have a low capital loss. People with high capital loss are evenly spread above the below the 50k mark.
- Hours per week: Most people work around 40 hours weekly. People who work more tend to earn >50k.
- Sex: There are more women in this study. Women have a higher chance of making >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

[2]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(...)`.

example, if 85% of the examples in the training set have `>50k = 0` and 15% have `>50k = 1`, then, when applied to a test set, `RandomClassifier` should randomly predict 85% of the examples as `>50k = 0` and 15% as `>50k = 1`.

Implement the missing portions of `RandomClassifier` according to the provided specifications. Then train your `RandomClassifier` on the entire training data set, and evaluate its training error. If you implemented everything correctly, you should have an error of 0.374 or 0.385.

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?

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.

Solution:

Classifying using k-Nearest Neighbors, training error for k = 3: 0.153; k = 5: 0.195; and 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 average cross-validation training error, test error and test micro averaged F1 Score (If you dont́ know what is F1, please click here) of each of your four models (for the `KNeighborsClassifier`, use k=5). To do this, generate a random 80/20 split of the training data, train each model on the 80% fraction, evaluate the error on either the 80% or the 20% fraction, and repeat this 100 times to get an average result. What are the average training and test error of each of your classifiers on the adult_subsample data set?

Solution:

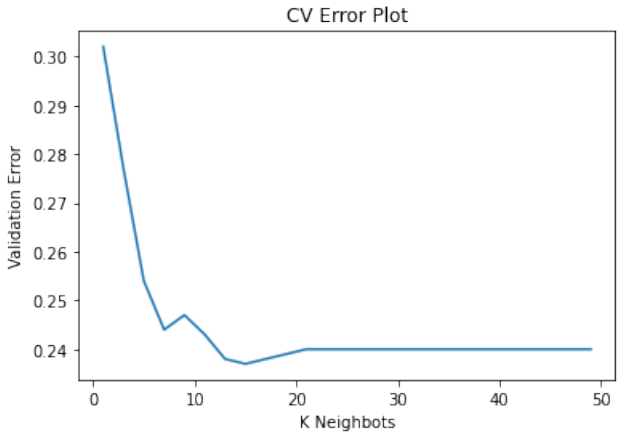
Investigating various classifiers:

Majority: Train error = 0.240, Test error = 0.240, F1 score: 0.760;
Random: Train error = 0.375, Test error = 0.382, F1 score: 0.618;
Decision Tree: Train error = 0.000, Test error = 0.205, F1 score: 0.795;
KNN: 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?

Solution:

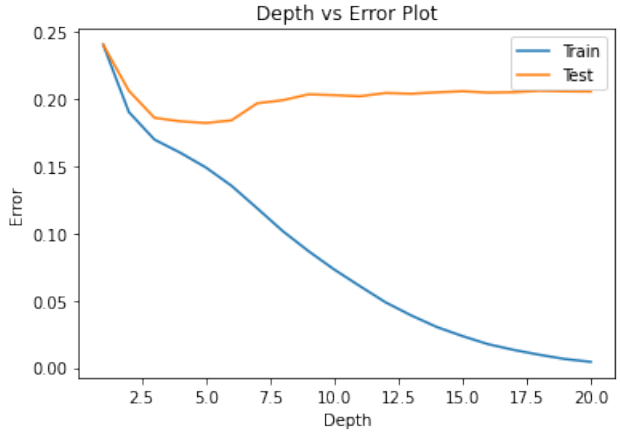


As k increases, the validation error decreases until k = 21, post which the error stagnates. The best value of k is 15, with an error value of 0.237.

(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, $1, 2, \ldots, 20$. You may find `cross_validate(...)` from `scikit-learn` helpful. Then plot the average training error and test error against the depth limit. Include this plot in your writeup, making sure to label all axes and include a legend for your classifiers. What is the best depth limit to use for this data? Do you see overfitting? Justify your answers using the plot.

Solution:

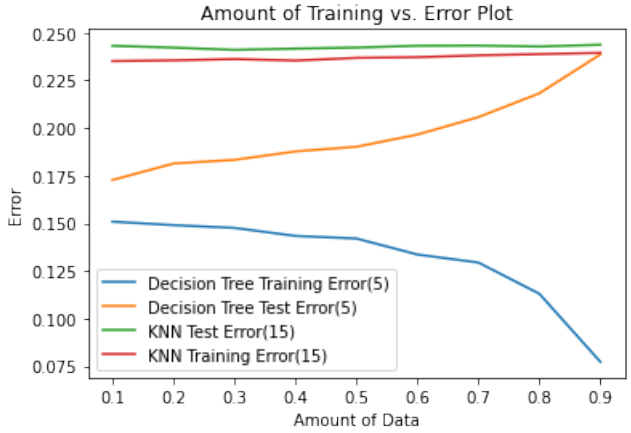


The best depth limit for this data is 5.0, with a testing error of 0.181. Overfitting is observable in the graph after a depth of 5.0. This is because the testing error increases in that range.

(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 using `train_test_split` from `scikit-learn` with `random_state` set to 0. Then, do the following experiments considering the 90% fraction as training and 10% for testing.

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

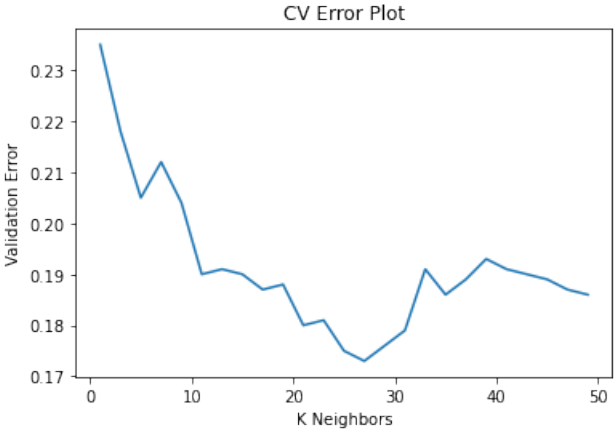
Solution:

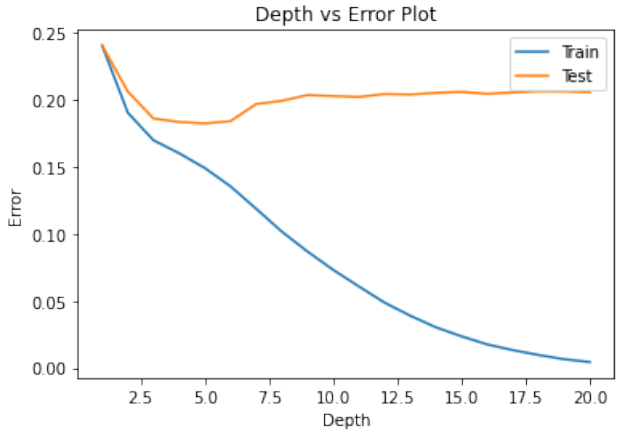


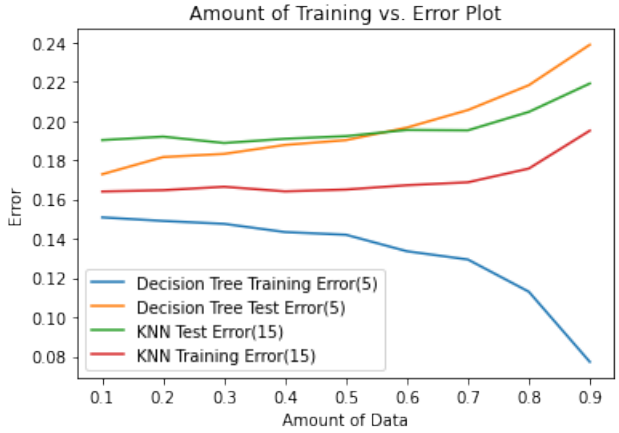
The Decision Tree error increases with the amount of training data. The training error is always less than the testing error.

(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.

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; k = 5: 0.129; and k = 7: 0.152.

Majority: Train error = 0.240, Test error = 0.240, F1 score: 0.760;
Random: Train error = 0.375, Test error = 0.382, F1 score: 0.618;
Decision Tree: Train error = 0.000, Test error = 0.205, F1 score: 0.795;
KNN: Train error = 0.133, Test error = 0.209, F1 score: 0.791.

Observations: For Random, Decision Tree, and Majority, the performance remains the same. KNN performance, however, significantly increases. Additionally, the best k value increases to 25-27.