CM146, Winter 2018
Problem Set 1: Decision trees and k-Nearest Neighbors
Due Jan 30, 2018 at 11:59 pm

Jingyue Shen

704797256

Parts of this assignment are adapted from course material by Andrea Danyluk (Williams), Tom Mitchell and Maria-Florina Balcan (CMU), Stuart Russell (UC Berkeley), Carlos Guestrin (UW) 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. 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 $n \geq 4$.)

There are 2^n examples in total. The number of examples whose Y = 0 should be 2^{n-3}. And the number of examples whose Y = 1 should be $2^n - 2^{n-3} = 7 * 2^{n-3}$. So if we use 1-leaf decision tree, since the number of examples of Y = 1 is larger than Y = 0, the leaf node should be Y = 1. And thus there will have 2^{n-3} mistakes.

(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?

No. Since no matter X1, X2, X3 or other X_k feature we choose to split the data, in the resulting two leaf nodes, the number of examples whose Y = 1 is always larger than those whose Y =0. So the sum of the number of mistaken examples in the two leaf nodes does not change and equals to the number of examples whose Y = 0.

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

The entropy $H[S] = -\frac{2^{n-3}}{2^n} \times \log \frac{2^{n-3}}{2^n} - \frac{7\times 2^{n-3}}{2^n} \times \log \frac{7\times 2^{n-3}}{2^n} = 0.54356$

(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?

If we split the dataset using X_1, then we can reduce the entropy of Y by a non-zero amount.

$$H[S'] = \frac{2^{n-1}}{2^n} \times 0 + \frac{2^{n-1}}{2^n} \times (-\frac{2^{n-3}}{2^n} \times \log^{\frac{2^{n-3}}{2^n}} - \frac{2^{n-1} - 2^{n-3}}{2^n} \times \log^{\frac{2^{n-1}-2^{n-3}}{2^n}}) = 0.45282$$

And $IG = H[S] - H[S'] = 0.54356 - 0.45282 = 0.09074 > 0$

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.

Denote the ratio $\frac{p_k}{p_k+n_k}$ as α, then after the splitting,

$$H[S_k] = -\alpha \times \log^{\alpha} - (1-\alpha) \times \log^{1-\alpha},$$

All $H[S_k]$ are the same. So the entropy after splitting should be

$$H[S'] = \sum_k \frac{S_k}{S} \times H[S_k] = 1 \times H[S_k] = B(\alpha)$$

The entropy before splitting is $H[S] = B(\frac{p}{p+n})$.

What we need to prove is that $\alpha = \frac{p_k}{p_k+n_k} = \frac{p}{p+n}$

Proof: Let $\frac{p_k}{p_k+n_k} = \frac{x}{y} \times C_k$, where x and y are relatively prime numbers and C_k is a constant

so that $x \times C_k = p_k$ and $y \times C_k = n_k + p_k$.

Then $\frac{p}{p+n} = \frac{p_1+p_2+...+p_n}{(p_1+n_1)+(p_2+n_2)+...+(p_n+n_n)} = \frac{x\times(C_1+C_2+...+C_n)}{y\times(C_1+C_2+...+C_n)} = \frac{x}{y} = \frac{p_k}{p_k+n_k}$

So $H[S] = B(\frac{p}{p+n}) = H[S']$ and $IG = H[S] - H[S'] = 0$

3 k-Nearest Neighbors and Cross-validation [15 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.

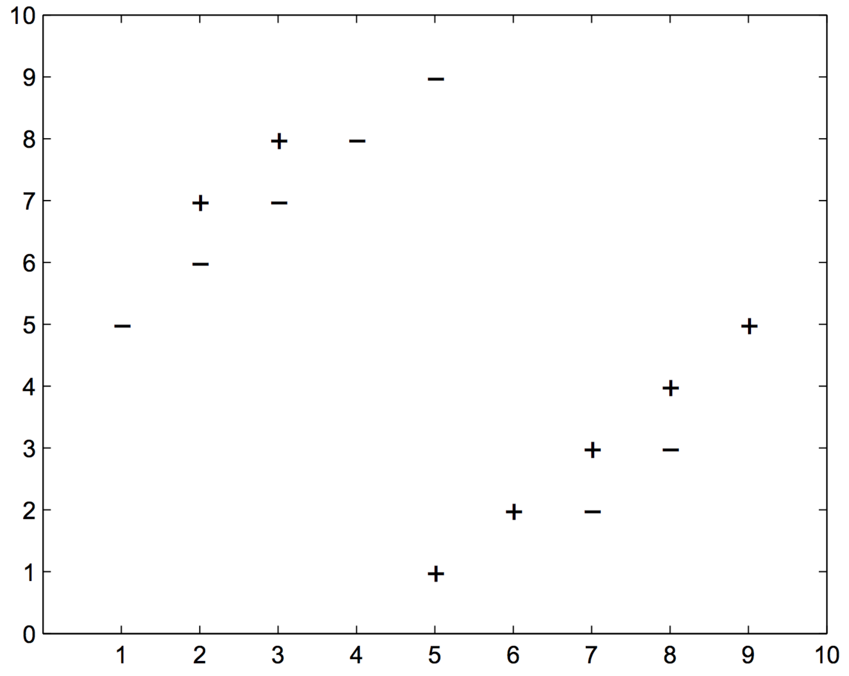


Figure 1: Dataset for KNN binary classification task.

(a) **(5 pts)** What value of k minimizes the training set error for this dataset? What is the resulting training error?

k = 1 minimizes the training set error. The training set error is 0.

(b) **(5 pts)** Why might using too large values k be bad in this dataset? Why might too small values of k also be bad?

When k is too small, KNN will have overfitting problem. If there are outliers in our dataset, then the prediction result will be inaccurate. If k is too large, then KNN will always predict the most common label within the training dataset.

(c) **(5 pts)** What value of k minimizes leave-one-out cross-validation error for this dataset? What is the resulting error?
K = 5 minimizes the leav-one-out cross-validation error for the dataset. The resulting error is 0.2857.

4 Programming exercise : Applying decision trees and k-nearest neighbors [60 pts]

Submission instructions

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

Introduction[1]

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

In this problem, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy.

Starter Files

---

code and data

- code : `titanic.py`
- data : `titanic_train.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:
  `http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.train_test_split.html`
- Metrics:
  `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html`

Download the code and data sets from the course website. For more information on the data set, see the Kaggle description: `https://www.kaggle.com/c/titanic/data`. (The provided data sets

[1]This assignment is adapted from the Kaggle Titanic competition, available at `https://www.kaggle.com/c/titanic`. Some parts of the problem are copied verbatim from Kaggle.

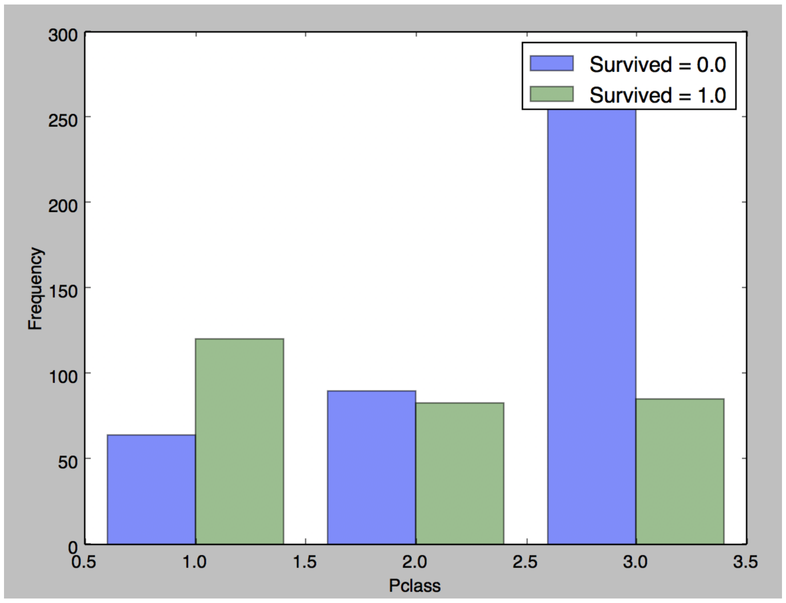
are modified versions of the data available from Kaggle.[2])

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.

4.1 Visualization [5 pts]

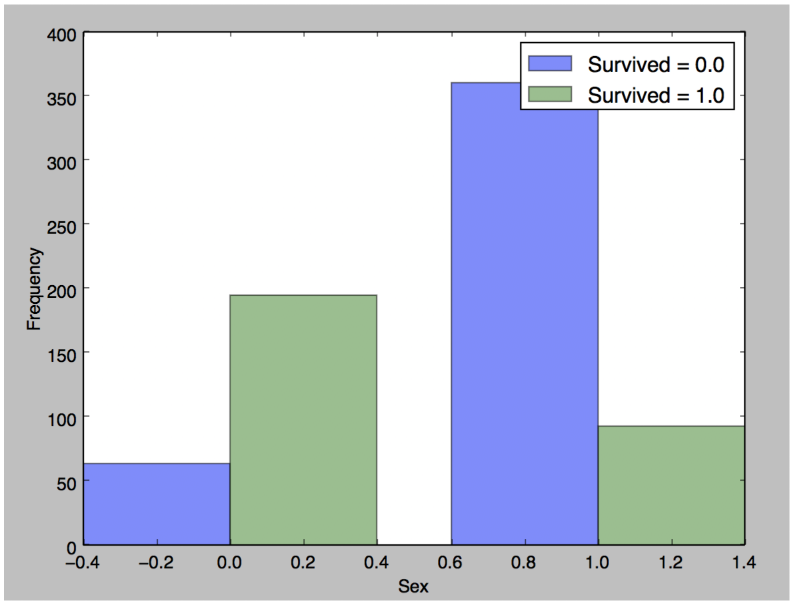
One of the first things to do before trying any formal machine learning technique is to dive into the data. This can include looking for funny values in the data, looking for outliers, looking at the range of feature values, what features seem important, etc.

(a) **(5 pts)** Run the code (`titanic.py`) to make histograms for each feature, separating the examples by class (e.g. survival). This should produce seven 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?

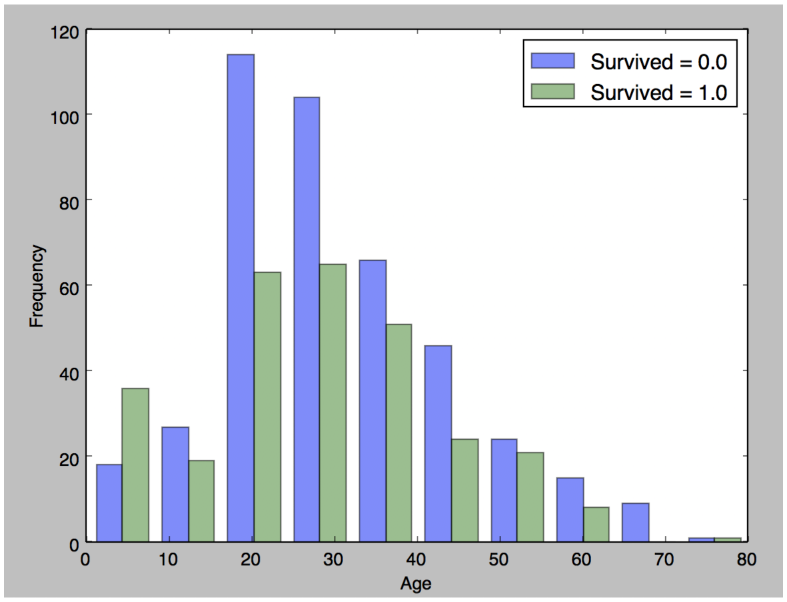


For the feature "ticket class", most passengers are in the third class. If we calculate the percetage of survivors with respect to all passengers in a given ticket class, we can find that the higher the ticket class is, the more likely a passenger can survive.

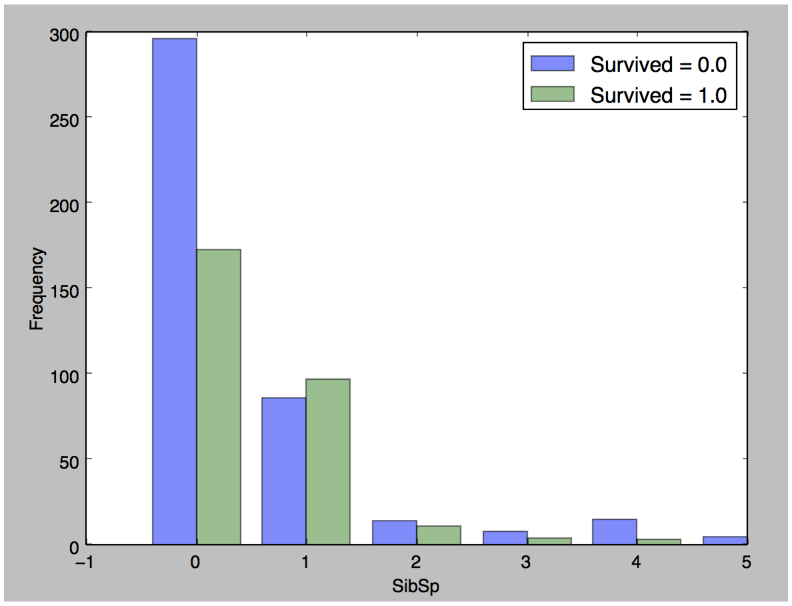
[2]Passengers with missing values for any feature have been removed. Also, the categorical feature `Sex` has been mapped to `{'female': 0, 'male': 1}` and `Embarked` to `{'C': 0, 'Q': 1, 'S': 2}`. If you are interested more in this process of *data munging*, Kaggle has an excellent tutorial available at https://www.kaggle.com/c/titanic/details/getting-started-with-python-ii.



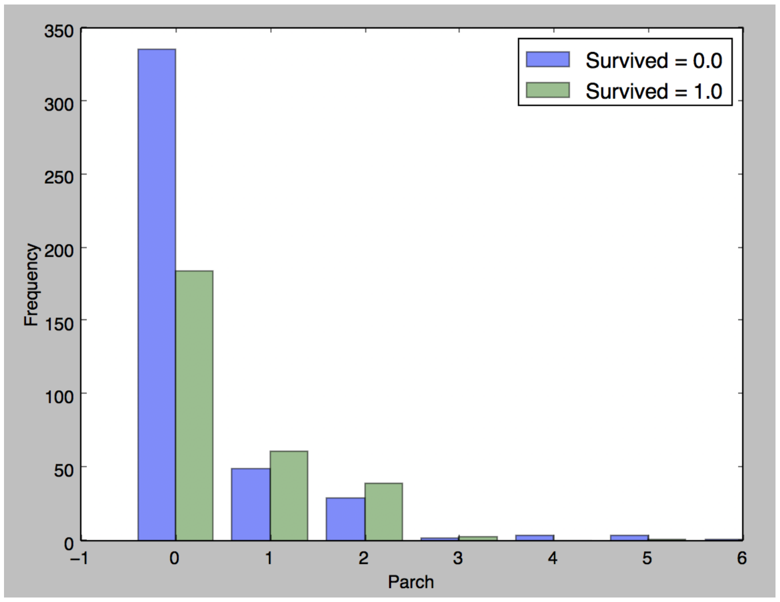
For the feature "sex", since female = 0, male = 1, we can see that women have much higher possibility to survive than men.



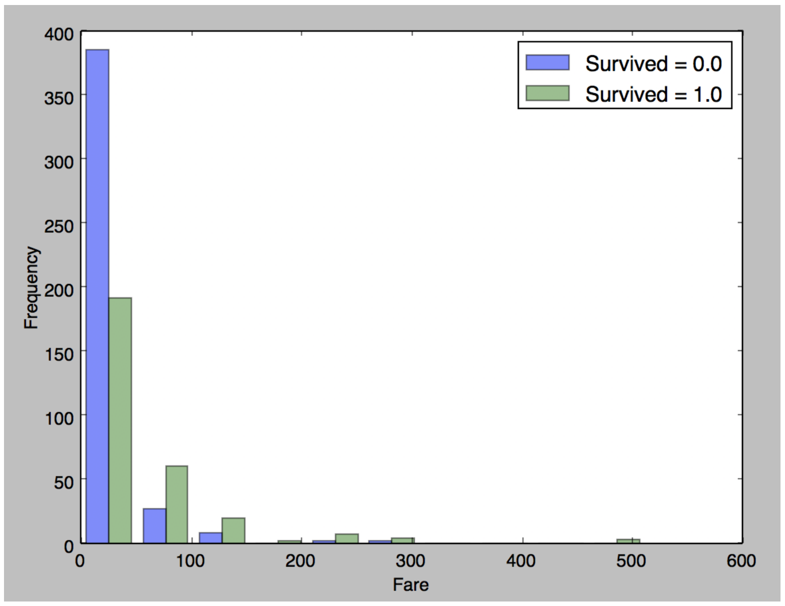
For the feature "age", we can see that most passengers are adults. And the trend is that kids and old people whose age is greater than 50 are more likely to survive.



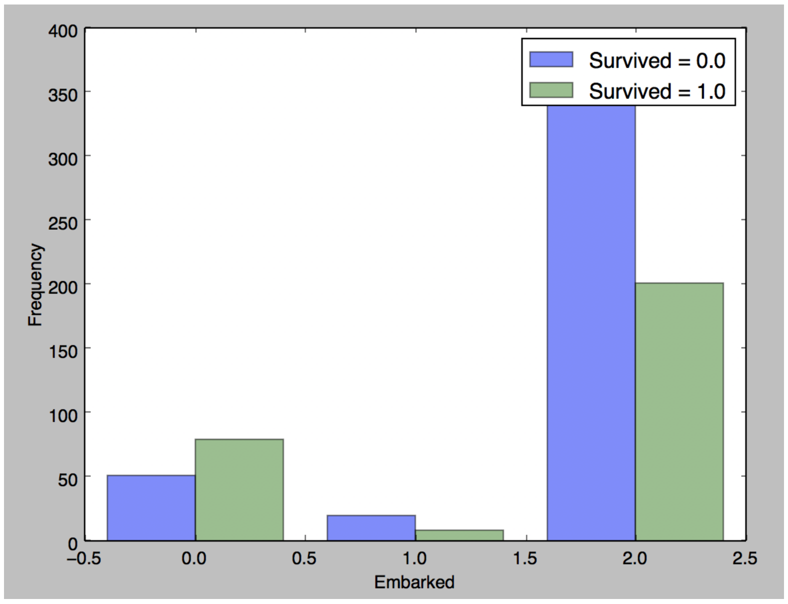
For the feature "the number of siblings/spouses aboard", we can see that most passengers came with no siblings/spouses and the number of passengers with siblings decrease drastically when the feature value grows. I think there is no particular relation between the number of siblings/spouses and the survival rate.



For the feature "the number of parents/children aboard", we can see that most passengers came without parents/children, and as the feature value grows, the number of people in that value decrease drastically. Besides, people with one or two children/parents have highest survival rate. This might be caused by parents trying their best to save their children first and thus give children higher survival rate when the number of kids is small.



For the feature "fare", I think it is highly related to "ticket class" feature. As shown in the graph, most passengers paid for minimum ticket price. And generally the survival rate increases as fare increases.



For the feature "embarked", most people embarked on Southampton. The survival rate is highest when people embark on Cherbourg, and lowest for people who embarked on Southampton(in other words, as the feature value increases, the survival rate decreases).

4.2 Evaluation [55 pts]

Now, let us 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.[3]

(b) **(5 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 60% of the examples in the training set have `Survived = 0` and 40% have `Survived = 1`, then, when applied to a test set, `RandomClassifier` should randomly predict 60% of the examples as `Survived = 0` and 40% as `Survived = 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 0.485.

(c) **(5 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?

The training error is 0.014

(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.

The training error when k = 3 is 0.167
The training error when k = 5 is 0.201
The training error when k = 7 is 0.240

(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 us use cross-validation instead.

Implement the missing portions of `error(...)` according to the provided specifications. You may find it helpful to use `train_test_split(...)` from `scikit-learn`. To ensure that we

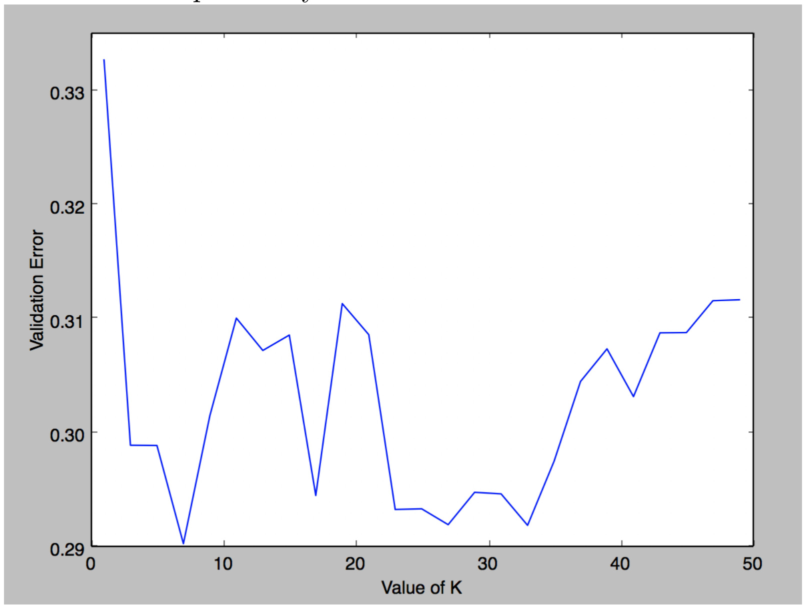
[3] Note that almost all of the model techniques in `scikit-learn` share a few common named functions, once they are initialized. You can always find out more about them in the documentation for each model. These are `some-model-name.fit(...)`, `some-model-name.predict(...)`, and `some-model-name.score(...)`.

always get the same splits across different runs (and thus can compare the classifier results), set the `random_state` parameter to be the trial number.

Next, use your `error(...)` function to evaluate the training error and (cross-validation) test error 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 Titanic data set?

	MajorityVoteClassifier	RandomClassifier	DecisionTreeClassifier	KNeighborsClassifier
train error	0.403779	0.489016	0.011529	0.212390
test error	0.407343	0.486573	0.240839	0.314685

(f) **(10 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?

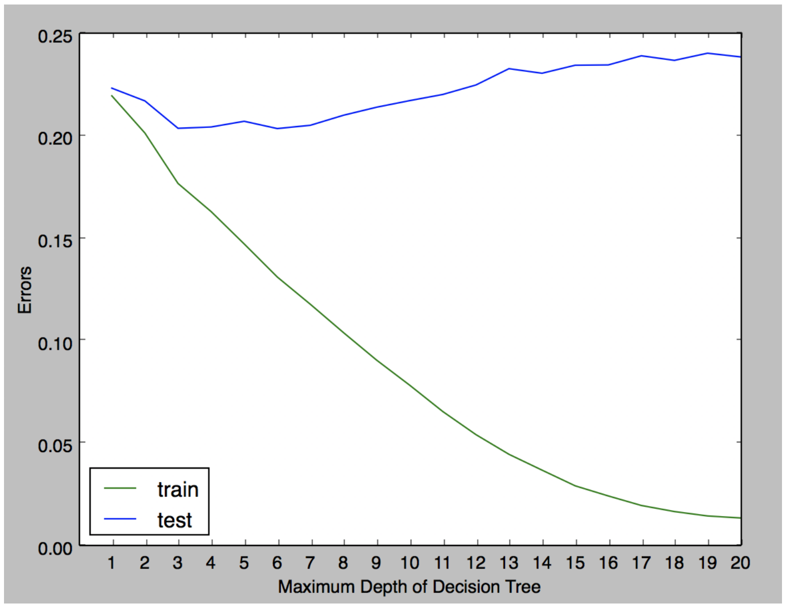


From the plot we can see that the validation error drops when k grows from 1 to 7, and then fluctuates when the value of k keeps increasing. In general the validation error does not vary too much, there is only about 0.04 difference between highest and lowest validation error. The initial high validation error is due to underfitting, while the increase of error when k keeps growing is due to overfitting. And when k = 7, the model has lowest validation error. So it should be the best k.

(g) **(10 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 us see whether this is the case for the Titanic data.

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.

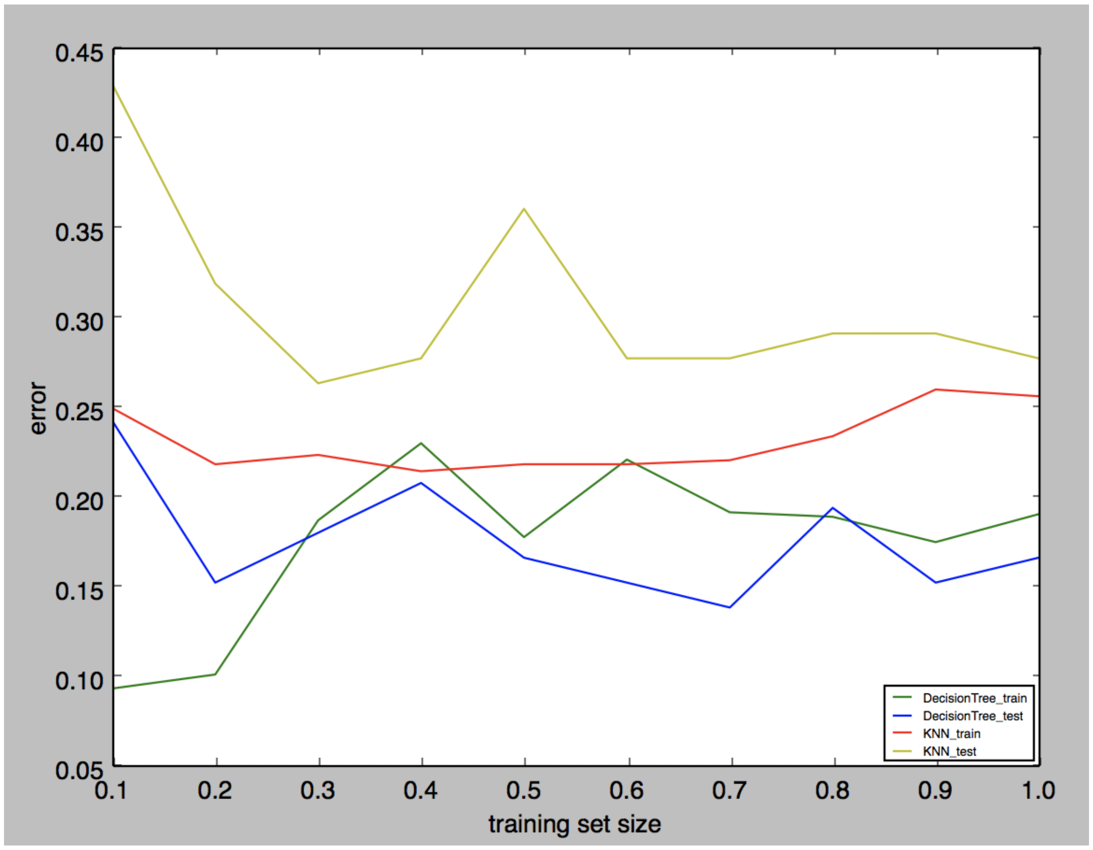
Include this plot in your writeup, making sure to label all axes and include a legend for your classifiers. What is the best depth limit to use for this data? Do you see overfitting? Justify your answers using the plot.



The best depth limit should be 3. The classifer with large depth limit apparently overfits the training data. As the depth limit increases, the training error keeps decreasing while the validation error decrease until depth limit = 3 and then keep increasing. There is a huge gap between training and validation error as depth limit grows, which is the sign of overfitting.

(h) **(10 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.



As shown in the graph, as the training set size increases, both KNN and Decision tree with optimum hyperparameters have similar training and validdation errors respectively. This indicates that both classifiers are not experiencing high bias or high variance problems. And after using full size of training set, we can see that decision tree has lower error on this dataset. So it indicates that decision tree might be a better choice for the classification problem of this dataset.