Zhubo Deng 605186231

0. You've trained a decision tree for classifying emails as being spam or ham and resulting tree, is getting very bad performance on both the train/test data splits. Your implementation has no bugs so what is causing the problems?

B

Solution: B, Decision tree is too shallow. Since increasing the learning rate of your decision tree will cause overfitting, but this one is getting very bad performance on both the train/test data, rather than only on the test data.

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 $Y = X_1 \wedge X_2 \wedge X_3$. That is, $Y = 1$ if $X_1 = 1$ and $X_2 = 1$ and $X_3 = 1$, 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	0
0	1	0	0	0
1	1	0	0	0
0	0	1	0	0
1	0	1	0	0
0	1	1	0	0
1	1	1	0	1

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

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 $n \geq 4$.)

1/1. Total: 2^n Mistakes = $\frac{2^n}{2^3} = 2^{n-3}$ iff $X_1 = X_2 = X_3 = 1$ then $Y = 1$; X_4 is the remaining.

A In general, remaing: $n-3$. so mistakes = 2^{n-3}. , error rate = $\frac{2^{n-3}}{2^n} = \frac{1}{8}$.

1/2. 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?

B

Solution: No, no splits will reduce the error rate. Since the single leaf tree will always predict 0 (the proportion of 0 is always 7/8), and it will makes the same number of errors.

1/3. What is the entropy of the output label Y for the 1-leaf decision tree (no splits at all)?

B. $-\left(\frac{7}{8}\log\frac{7}{8} + \frac{1}{8}\log\frac{1}{8}\right) = -(-0.1926 \times 0.875 - 3 \times 0.125) = 0.543$

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

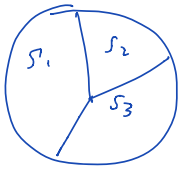
$$B(q) = -q \log q - (1 - q) \log(1 - q).$$

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.

Bernoulli Ran. Var. $p(X=1) = q$

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



Defn: $H[S] = -\sum_{v=1}^{k} P(S=a_v) \log_2 P(S=a_v)$

Defn: $\text{Gain}(S,A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$.

Given that $H(S) = B(\frac{p}{p+n})$, then:

$$\text{Gain} = B\left(\frac{p}{p+n}\right) - \left[\frac{p_1+n_1}{p+n} B\left(\frac{p_1}{p_1+n_1}\right) + \frac{p_2+n_2}{p+n} B\left(\frac{p_2}{p_2+n_2}\right) + \cdots\right]$$

$$\text{Gain} = B\left(\frac{p}{p+n}\right) - \sum_k \frac{p_k+n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right)$$

Since $\frac{p_k}{p_k+n_k}$ is the same for all k, then $\frac{p_k}{p_k+n_k} = \frac{p}{p+n}$, $p+n = \sum_k p_k+n_k$

plug in, $\text{Gain} = B\left(\frac{p}{p+n}\right) - \frac{\sum_k p_k+n_k}{p+n} B\left(\frac{p}{p+n}\right)$

$$= B\left(\frac{p}{p+n}\right) - \frac{p+n}{p+n} B\left(\frac{p}{p+n}\right)$$

$$= B\left(\frac{p}{p+n}\right) - B\left(\frac{p}{p+n}\right)$$

$$= 0$$

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.

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

(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?

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

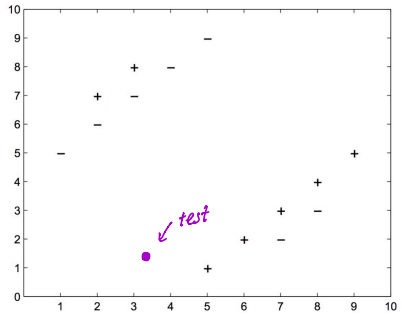


Figure 1: Dataset for KNN binary classification task.

(a) Since a point can be its own neighbor, so k = 0 should minimize the training error, which is 0. Each data has its own region in the space.

(b) When k is too big (k = 13), it will not do classification successfully. When k is too small (k = 0), it will overfit the data.

(c) In KNN, k must be odd number for a 2-class, and also, k cannot be a multiple of class number. So in these case, k could be 1, 3, 5, 7, 9, 11, 13.
Therefore, we can take k = 7, since we've already known that k cannot be too small or too large from part a and part b. If there's a test data as shown, when k = 7, it will be classified as "+" due to 7 nearest neighbor (5 of + and 2 of -). I think k = 5 should also work for this problem since this test point will also be classified as "+" due to 5 nearest neighbor.

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?

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

[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.

[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(...)`.

4/a.
Pclass: First class had a higher survival rate than other two, and female had a higher rate than male.
Sex: Female had a higher survival rate in general.
Age: 0~10 had a higher survival rate; 20~30 had a lower survival rate.
SibSp: People who has 1 sib or sp had a higher survival rate; people who has 0 sib or sp had a lowest survival rate.
Parch: People has 1~2 parch had a higher survival rate; people has 0 parch had a lower survival rate.
Fare: People has 0 fare had a lower survival rate; people has more than 0 fare had a higher survival rate.
Embarked: People embarked on Chengbourg had a higher survival rate than others.

4/b.
After coding, I got an error of 0.485.

Output:
Classifying using Majority Vote...
    -- training error: 0.404
Classifying using Random...
    -- training error: 0.485

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?

(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.

(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 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?

(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?

(g) **(10 pts)** One problem with decision trees is that they can *overfit* to training data, yielding

4/c. Output:

```
Classifying using Decision Tree...
    -- training error: 0.014
```

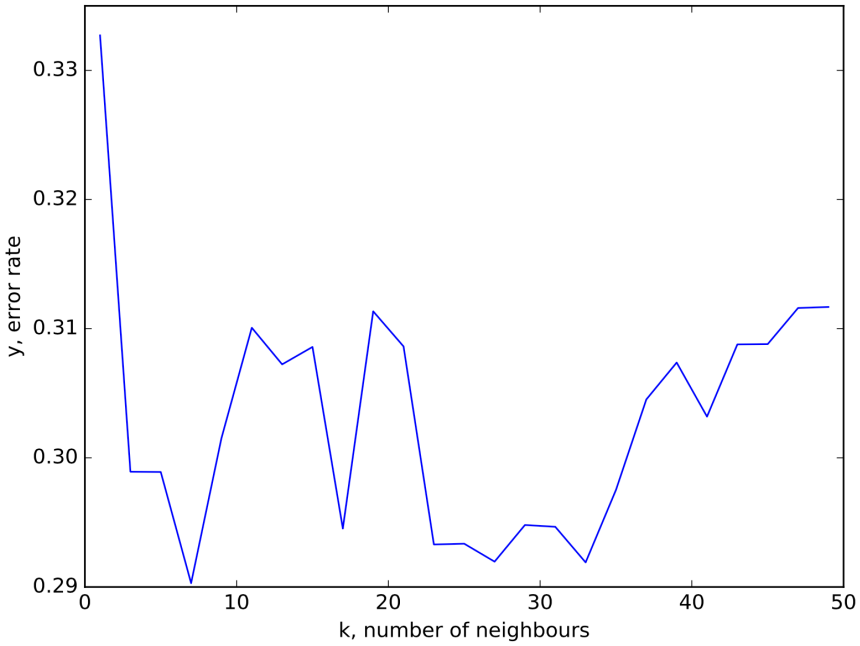
4/d. Output:

```
Classifying using k-Nearest Neighbors...
    -- training error for k = 3: 0.167
    -- training error for k = 5: 0.201
    -- training error for k = 7: 0.240
```

4/e. Output:

```
Investigating various classifiers...
Average results of MajorityVoteClassifier:
    -- training error: 0.404, test error: 0.407
Average results of RandomClassifier:
    -- training error: 0.489, test error: 0.487
Average results of DecisionTreeClassifier:
    -- training error: 0.012, test error: 0.241
Average results of KNeighborsClassifier:
    -- training error: 0.212, test error: 0.315
```

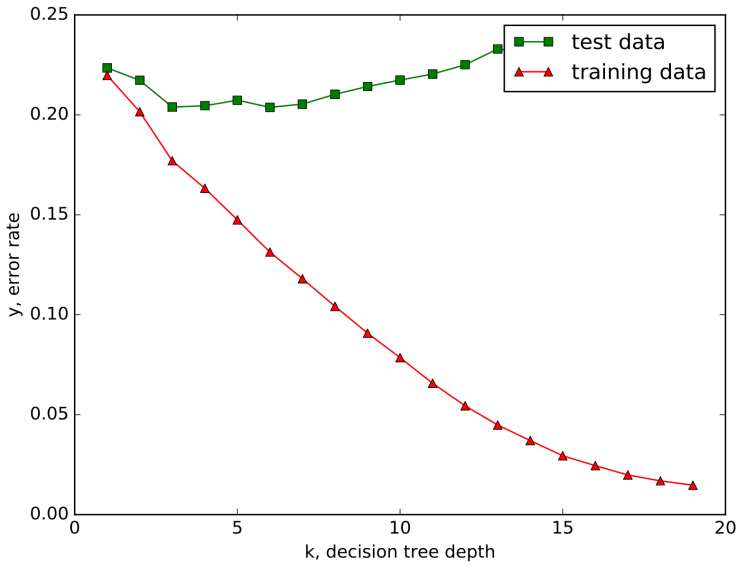
(f) **(10 pts)** One way to find out the best value of k for `KNeighborsClassifier` is n-fold cross validation. Find out the best value of k using 10-fold cross validation. You may find the `cross_val_score(...)` from `scikit-learn` helpful. Run 10-fold cross validation for all odd numbers ranging from 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?



4/f.
Solution:
The best value of k using 10-fold cross validation is 7. As shown in the graph, I run from k = 1 ~ 50, but it shows that at 7 , the error rate is the lowest. Therefore, 7 Nearest Neighbour has a higher accuracy.

(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.

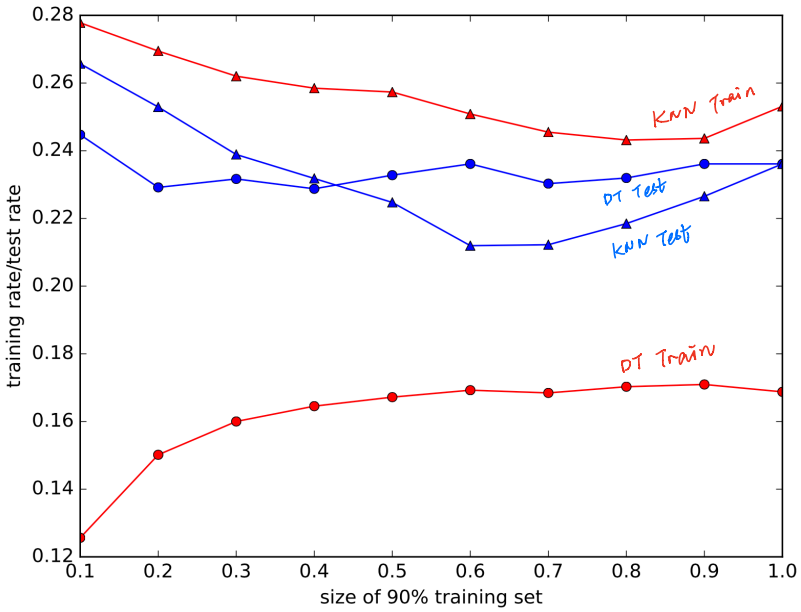
One way to prevent decision trees from overfitting is to limit their depth. Repeat your cross-validation experiments but for increasing depth limits, specifically, $1, 2, \ldots, 20$. Then plot the average training error and test error against the depth limit. Include this plot in your writeup, making 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.



4/g.
Solution:
Overfitting means the model fits well on the training data (red, triangle in the graph) but perform bad on the test data (green square in the graph). To prevent decision trees from overfitting the data, we should limit their depth. To analyze the test error rate, the error decreases from k = 1 to 3, and it stays around 0.20 from k = 3 to 6, and then it increases when k > 6. To analyze the training error, it keeps decreasing as k becomes larger. As shown in the figure, k = 6 is the “turning point”, that is, k = 6 has a low test error in the k range, and it has a normal traning error relatively in the range which indicates that then training data doesn't overfit the model. Therefore, k = 6 is the best depth limit to use for this data.

(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.



4/h.
Solution:
For both of the cases, training error has a increasing trend when training examples becomes larger, and also for test data; and for both of DT and KNN, they all have differences between the training error and test error. At beginning, both of them decrease slowly, but after proportion > 0.7, they start to increase because of data overfitting. Only decision tree training error always has a increasing trend . And also, no matter how many data we are training in this way, it'll still show this feature because the splitting proportion doesn't change.