CSM164 Homework 1

Zipeng Fu

January 29, 2018

1 Splitting Heuristic for Decision Trees

(a) The best 1-leaf decision tree making minimum error should output 1 for all instance examples. Hence, the mistakes made should be total number times the possibility of examples with label $Y = 0$.

$$2^n * \left(\frac{1}{2}\right)^4 = 2^{n-4}$$

(b) No, there is no such split. If the decision boundary of the only internal node, which the root, is based on the value of a single X_i, then a split will not reduces the number of mistakes. Without loss of generality, I pick X_1 as the attribute to be tested at the internal node. If X_1 is 1, the leaf should be 1 because Y is always 1. If X_1 is 0, the leaf should be 0, without losing the meaning of split based on the value of attribute. Then, the probability of $Y = 1$ given that $X_1 = 0$ is

$$1 - \frac{\left(\frac{1}{2}\right)^4}{\frac{1}{2}} = 1 - \left(\frac{1}{2}\right)^3 = \frac{7}{8}.$$

As the result, the mistakes will be made over 2^n training examples will be

$$\frac{7}{8} * 2^n = 7 * 2^{n-3},$$

which is much larger than the mistakes in part (a).

(c)

$$Entropy = -\frac{1}{16} * \log_2\left(\frac{1}{16}\right) - \frac{15}{16} * \log_2\left(\frac{15}{16}\right) \approx 0.3373$$

(d) Yes, the method in part (b) reduces the entropy of the output Y. The conditional entropy is

$$Entropy = \frac{1}{2}\left(-\frac{7}{8}\log_2\left(\frac{7}{8}\right) - \frac{1}{8}\log_2\left(\frac{1}{8}\right)\right) + \frac{1}{2} * 0 \approx 0.2718.$$

2 Entropy and INformation

(a) If the ratio $\frac{p_k}{p_k+n_k}$ is the same for all k, then the fraction of positive for the superset X_i is also the same. Let the ratio be r.

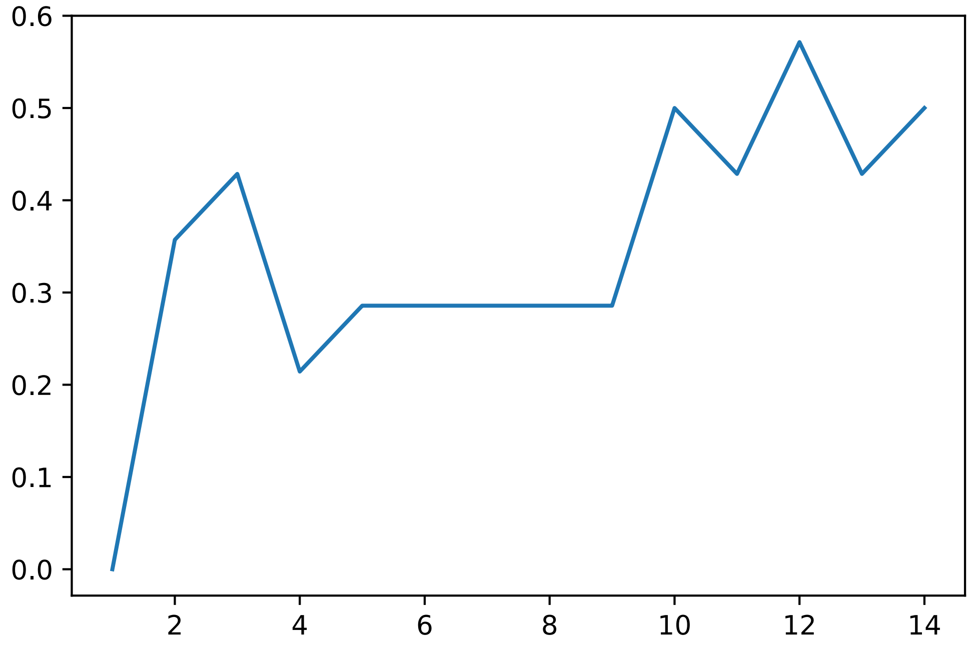
$$H\left(\text{S\_before\_split}\right) = B\left(r\right)$$

$$H\left(\text{S\_after\_split}\right) = \sum_{i=0}^{k} \text{fraction}\left(S_i\right) H\left(S_i\right) = \sum_{i=0}^{k} \text{fraction}\left(S_i\right) B\left(r\right) = \left[\sum_{i=0}^{k} \text{fraction}\left(S_i\right)\right] * B\left(r\right) = B\left(r\right)$$

Hence, the entropies before the split and after the split are equal, which means the information gain of this attribute is 0.

3 k-Nearest Neighbors and Cross-validation

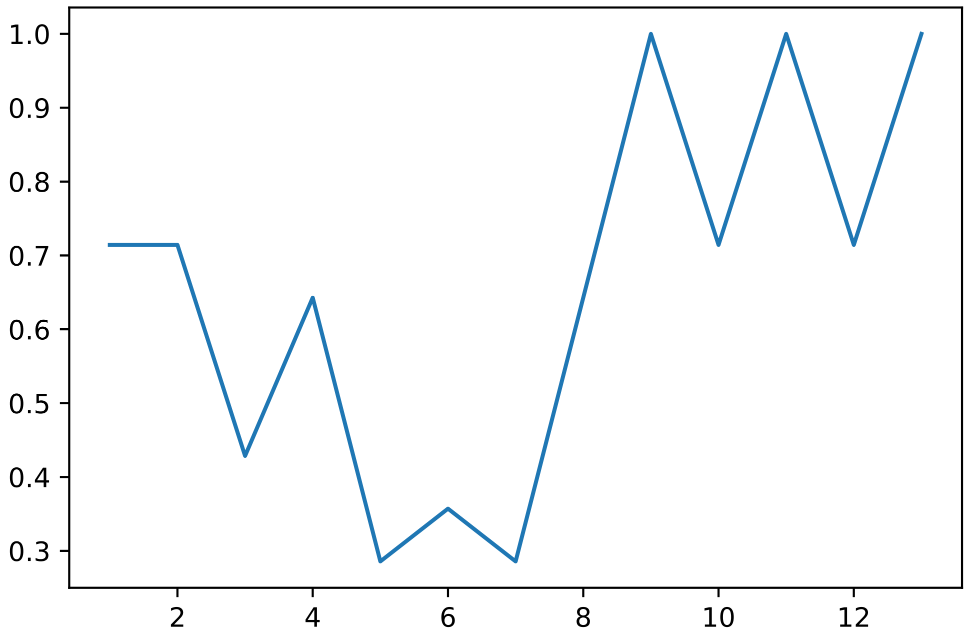
(a) When k is 1, the nearest neighbor is the testing point itself, selecting from the training set. Thus, no error will occur and the minimum resulting trainning error is 0.



```
from sklearn.neighbors import KNeighborsClassifier as KNN
import matplotlib.pyplot as plt
X = [[1,5],[2,6],[2,7],[3,7],[3,8],[4,8],[5,1],[5,9],\
    [6,2],[7,2],[7,3],[8,3],[8,4],[9,5]]
y = [0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1]
klist = []
errlist = []
for k in range(1, 15):
    knn = KNN(n_neighbors=k, p=2)
    knn.fit(X, y)
    errCount = 0
    predict = knn.predict(X)
    for i in range(14):
        if predict[i] != y[i]:
            errCount += 1
    klist.append(k)
    errlist.append(errCount/14)
plt.plot(klist, errlist)
plt.savefig("3aGraph.svg")
```

(b) For KNN problems, the choice of the hyper-parameter, k, is crucial. Using too large values k will make distinctions between difference classes blurred. The label predicted largely relies on the global majority labels in the training set, instead of the local majority. On the other hand, too small values of k will let the predicted labels of test set susceptible to noise in the training set. The results are prone to outliners near the test input attribute values (Detailed graph plot is shown in part a).

(c) When k equals to 5 or 7, the leave-one-out cross-validation error is minimized. The resulting error is 4 out of 14 test examples (0.285714285).

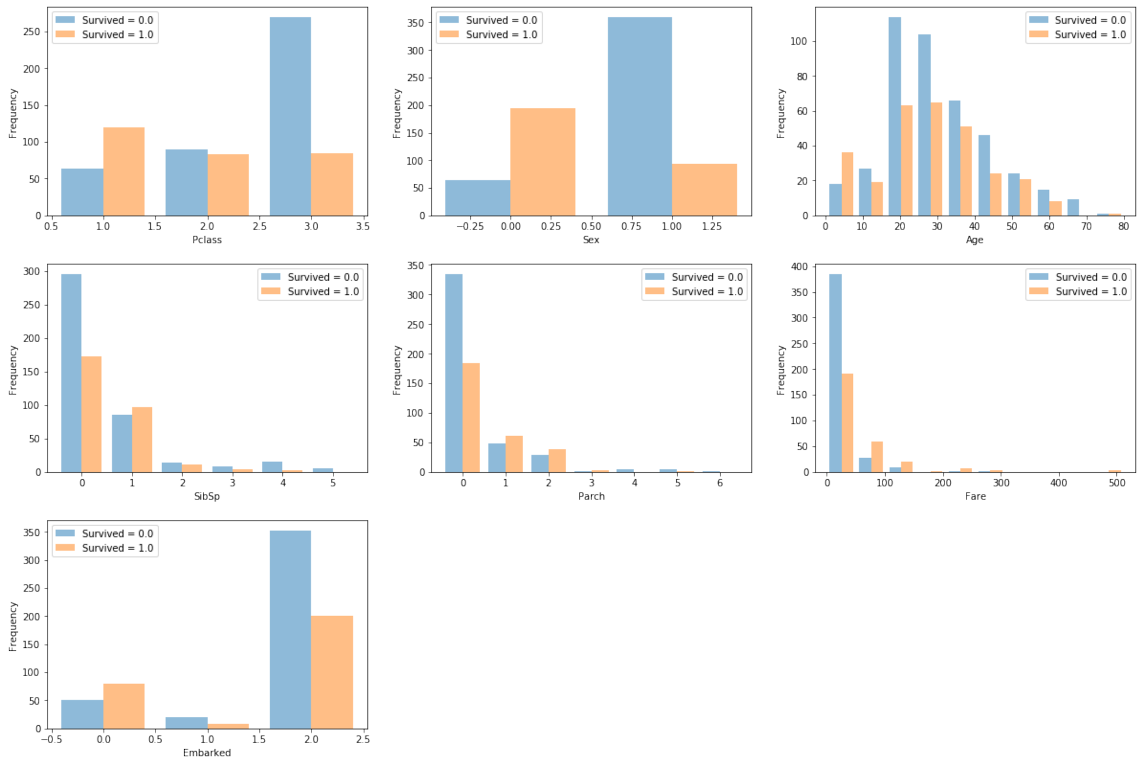


```
from sklearn.neighbors import KNeighborsClassifier as KNN
import matplotlib.pyplot as plt
X = [[1,5],[2,6],[2,7],[3,7],[3,8],[4,8],[5,1],\
    [5,9],[6,2],[7,2],[7,3],[8,3],[8,4],[9,5]]
y = [0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1]
klist = []
errlist = []
for k in range(1, 14):
    errCount = 0
    for i in range(14):
        knn = KNN(n_neighbors=k, p=2)
        knn.fit(X[:i]+X[i+1:], y[:i]+y[i+1:])
        predict = knn.predict([X[i]])
        if predict[0] != y[i]:
            errCount += 1
    klist.append(k)
    errlist.append(errCount/14)
plt.plot(klist, errlist)
```

```
    plt.savefig("3cGraph.pdf")
```

4 Programming exercise: Applying decision trees and k-nearest neighbors

(a)



Ticket Class	Upper class passengers tend to have a higher survival rate than lower classes.
Sex	Women tend to have a higher survival rate than men.
Age	Young kids below 10 years old and the eldly over 70 years old have a higher survival rate than teenagers, young adults and middle-aged adults.
Siblings/Spouses	People with 1 sibling or spouse have a higher survival rate than any other groups.
Parents/Children	People with no parents or children have a significant lower survival rate than other groups.
Fare	People paying less than \$50 have the lowest survival rate. Other groups survived with approximately same percentage.
Port Embark	(0 = Cherbourg, 1 = Queenstown, 2 = Southampton) People embarked in Chengbourg have the lowest survival rate. The other 2 groups have approximately same survival rate.

(b) RandomClassifier have a property of probability. I set it to be the probability of 0 as the predicted label. Since the seed is fixed, the pseudo-random number generator will generate the same sequence of 0 and 1 with corresponding probabilities. Hence, the resulting error is fixed, which is 0.485.

```
Classifying using Random...
-- training error: 0.485
```

(c) Set the decision tree model to be based on information gain by passing in the parameter, *criterion*, to be *entropy*. Training error of this Dicision Tree Classifier based on information gain is 0.014.

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

(d) I set the parameter of $KNeighborsClassifier$, *n_neighbors*, to be 3, 5, 7, respectively. Euclidean distance is used, $p = 2$. The results are

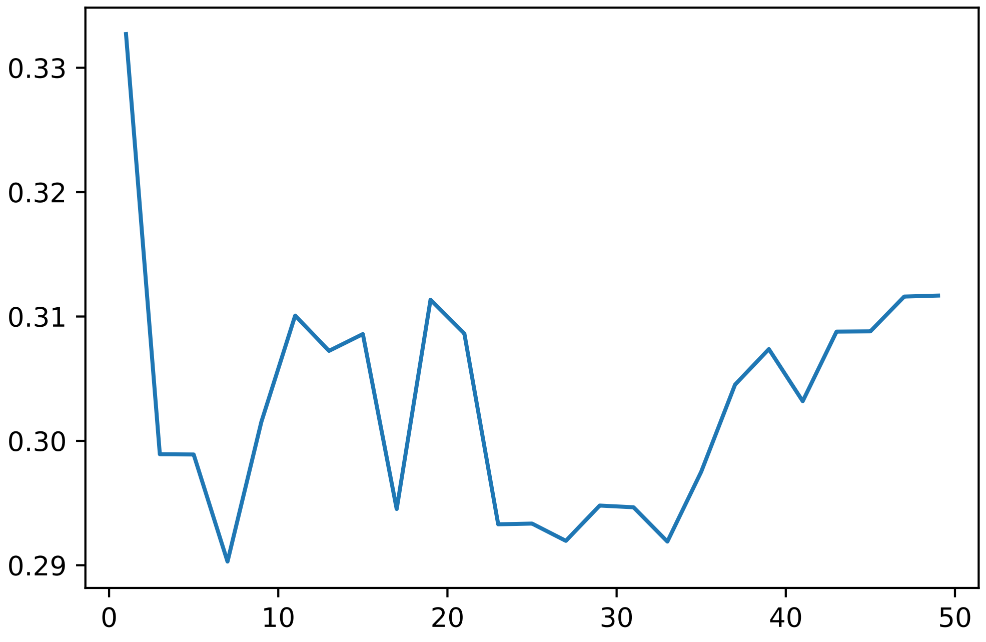
```
Classifying using k-Nearest Neighbors...
-- training error: 0.167 when k = 3
-- training error: 0.201 when k = 5
-- training error: 0.240 when k = 7.
```

(e) I set the *random_state* of *train_test_split* to be 1234. Then the pseudo number sequence is unchanged during each execution, so the output averages are fixed. The result is not normalized, as indicated on Piazza.

```
Investigating various classifiers...
Average results of MajorityVoteClassifier:
-- training error: 0.406, test error: 0.399
Average results of RandomClassifier:
-- training error: 0.518, test error: 0.559
Average results of DecisionTreeClassifier:
-- training error: 0.005, test error: 0.241
Average results of KNeighborsClassifier:
-- training error: 0.206, test error: 0.280
```

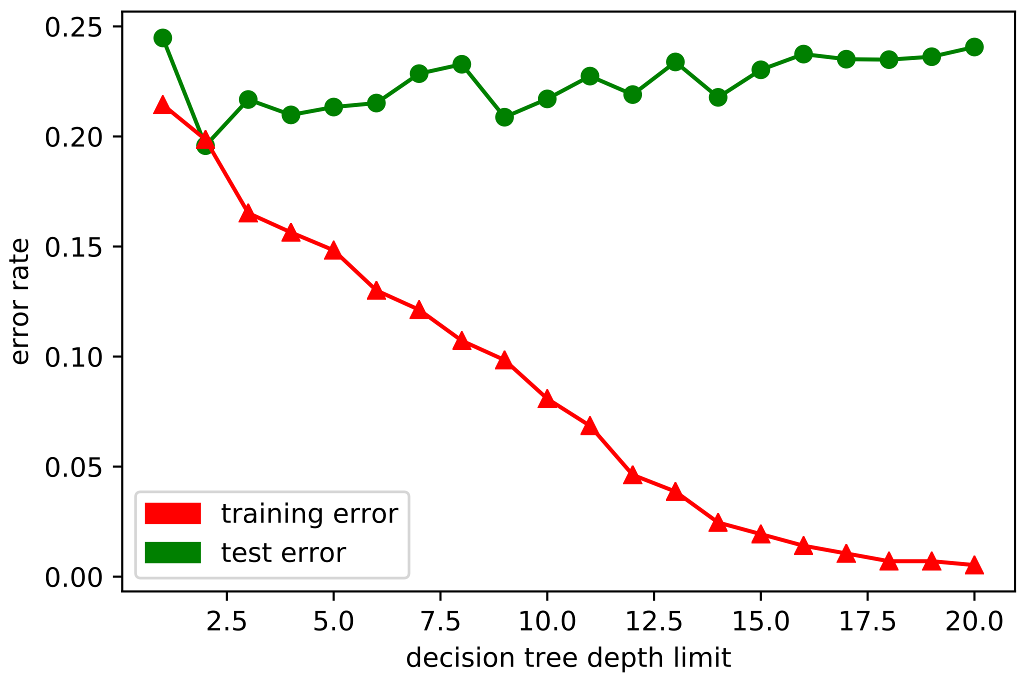
(f) (The features are not normalized.) In the figure below, the vertical axis is the validation error rate and the horizontal axis is the odd number, k. As we can see, the best value of k is 7, given the lowest error rate of 7 Nearest-Neighbors Model.

(Graph is shown on the next page.)



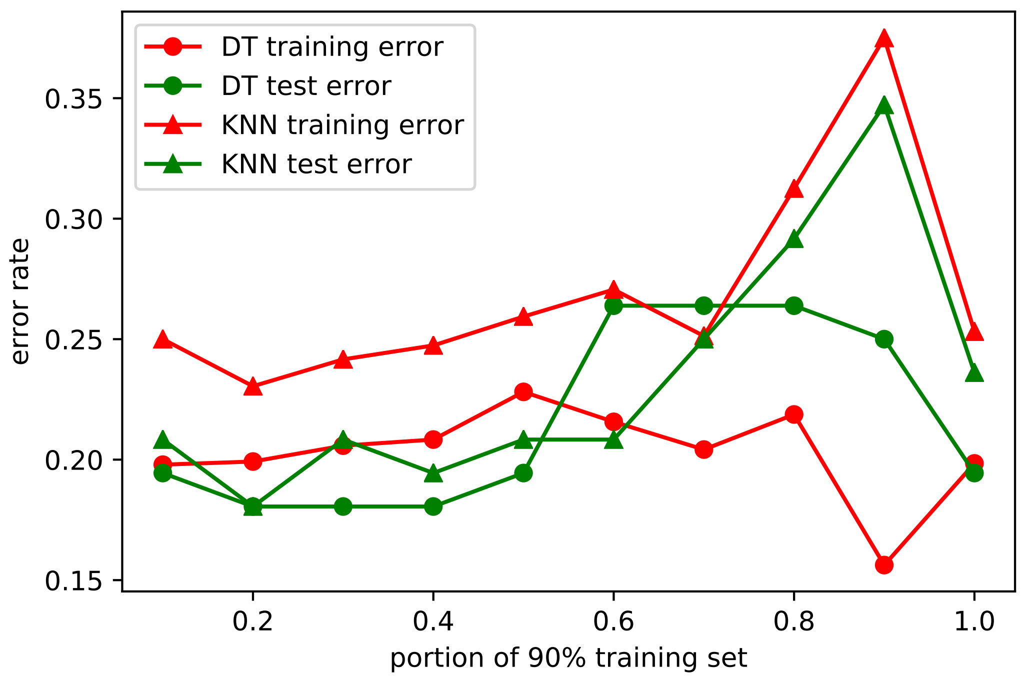
(g) The best depth limit to use for this dataset is 2, where the test error is the lowest, which is belwo 0.20. As we can observe from the graph, the overfiting occurs after the depth limit exceeds 2, because the test error starts to show an increasing trend when depth limit is more than 2, and at the same time, the training error keeps decreasing. This is the phenomenon of overfitting, which can be regularized by limiting the maximum depth based on validation set results.

(Graph is shown on the next page.)



(h) The graph shows mimimums for both test errors of KNN and Decision Tree at 0.2 portion of 90% training set. From 0.3 to 0.9, the noise of data, as well as lack of techniques, like feature selection, feature normalization and transforming categorial data to numerial data, is detrimental to the modelling process. Also, since the hyper-parameters are tuned based on 80/20 for Decision Tree and 90/10 for KNN. The error rates decrease for DT when the model is trained based on 0.9*90% of data set, and for KNN when it's trained based on 1.0*90% of data set.

(Graph is shown on the next page.)


DT training error
DT test error
KNN training error
KNN test error
error rate
portion of 90% training set
0.15
0.20
0.25
0.30
0.35
0.2
0.4
0.6
0.8
1.0