CM146, Winter 2024
Problem Set 1: Decision trees, Nearest neighbors
Due Jan 31, 2024 at 11:59 pm PST

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) **(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. Justify and answer for the general case when $n \geq 4$ for full credit.)

Solution: The best 1-left decision tree makes 1 mistake for every $8 = 2^3$ data points. Each time we add a new feature, the number of mistakes doubles. Thus, the best decision tree makes 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? (Note that, as in lecture, you should restrict your attention to splits that consider a single attribute.)

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).

Solution: No. If we create an internal node with a single feature, there are 4 possibilities for the external leaves:

- 0/0, which is wrong 7/8 of the time,
- 1/0 and 0/1, which are at best wrong 3/8 of the time,
- 1/1, which is wrong 1/8 of the time.

At best, adding an internal node yields a tree with the same accuracy as our single-leaf decision tree.

(c) **(5 pts)** What is the entropy of the label Y?

Solution: $H[Y] = -\sum_{k=0}^{1} P(Y = a_k) \log P(Y = a_k) = -(\frac{7}{8} \log \frac{7}{8} + \frac{1}{8} \log \frac{1}{8}) \approx 0.544$

(d) **(5 pts)** 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, splitting across any feature will reduce the entropy of Y. All 2-leaf splits will result in one leaf with 4 $Y = 1$ data points and the other leaf with a single $Y = 0$ point and 3 $Y = 1$ points. Thus, the conditional entropy for all such splits is

$$\begin{aligned} H[Y|X] &= \Sigma_{k=0}^{1} P(Y = a_k) H[Y|X = a_k] \\ &= \frac{1}{2}(\frac{1}{4} \log \frac{1}{4} + \frac{3}{4} \log \frac{3}{4}) + \frac{1}{2}(\frac{4}{4} \log \frac{4}{4}) \\ &\approx 0.406 \end{aligned}$$

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) **(2 pts)** Show that $0 \leq H(S) \leq 1$ and that $H(S) = 1$ when $p = n$.

Solution: Because $H(S) = B(q(p))$, we can prove that $0 \leq H(S) \leq 1$ by showing that the maximum of $B(q)$ is 1 and the minimum is 0. Taking the derivative of $B(q)$, we get

$$\begin{aligned} B'(q) &= \frac{d}{dq}\Big[-q\log_2 q - (1-q)\log_2(1-q)\Big] \\ &= -\Big(\log_2 q + q\frac{1}{q}\frac{1}{\ln 2}\Big) - \Big[(-1)\log_2(1-q) + (1-q)\Big(\frac{1}{(1-q)}\frac{1}{\ln 2}\Big)(-1)\Big] \\ &= \log_2(1-q) - \log_2(q) \\ &= \log_2\Big(\frac{1-q}{q}\Big). \end{aligned}$$

$B'(q) = 0$ when $q = \frac{1}{2}$ and $B'(q)$ is undefined for $q = 0$, so our critical points are $B(\frac{1}{2}) = 1$ and $B(0) = 0$. Thus, $B(q)$ has a minimum value of 0 and a maximum of 1, so we must have that $0 \leq H(S) \leq 1$.

(b) **(3 pts)** 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.

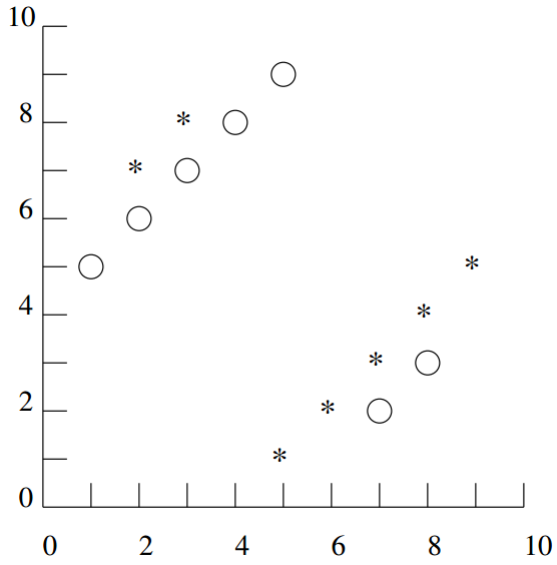
Solution: Let $q = \frac{p_k}{p_k+n_k}$, which is the same for all subsets and for the total sample. Then by definition, both the total entropy $H[Y]$ and the conditional entropy of all subsets $H[Y|X = a_k]$ is $B(q)$. Notice that since $H[Y|X = a_k]$ is constant across all subsets, so we can factor it out of a sum. Thus, we get that the conditional entropy is

$$\begin{aligned} H[Y|X] &= \sum_{i=1}^{k} P(X = A_k)H[Y|X = a_k] \\ &= H[Y|X = a_k]\sum_{i=1}^{k} P(X = A_k) \\ &= H[Y|X](1) \\ &= B(q). \end{aligned}$$

This means that the total gain is $GAIN = H[Y] - H[Y|X] = B(q) - B(q) = 0$.

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) **(3 pts)** 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: Training error is minimized at 0 for $k = 1$ because each point's nearest neighbor is itself. This does not generalize well to test error because we are labeling each point by comparing it to itself instead of its neighbors, so we have no idea how accurately new, unseen data points will be classified.

(b) **(3 pts)** 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: LOOCV test error is minimized at $\frac{4}{14} \approx 0.286$ with $k = 5$ and $k = 7$. LOOCV error is a better proxy for test error because the point itself is not included in the training set and can only be labeled through comparison to its neighbors.

(c) **(4 pts)** 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?

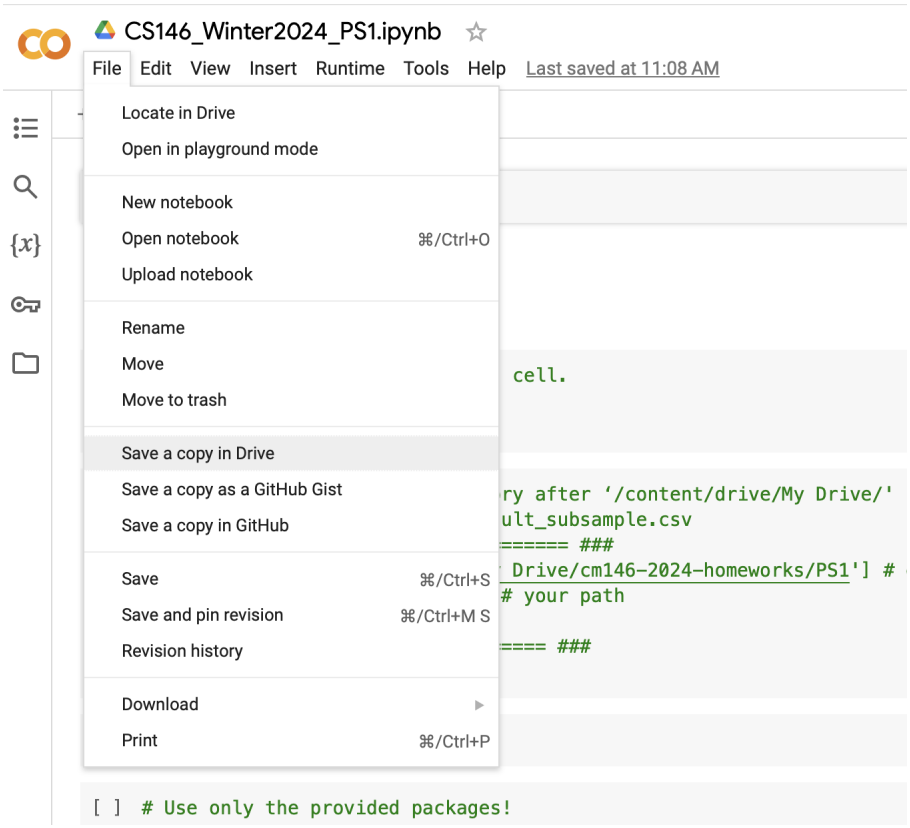
Solution: The LOOCV error for the lowest k ($k = 1$) is $\frac{10}{14} \approx 0.714$ and the highest k ($k = 13$) is $\frac{14}{14} = 1.0$. With ks that are too small, random noise has a significant effect on classification, resulting in overfitting. With ks that are too large, the we allow points too far away to affect the label, resulting in underfitting. Additionally, with very large ks, we are effectively left with plurality voting, as the decision is significantly affected by which label has more points in the data set.

4 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. If you cannot open the notebook with Colab or don't see an option, click on "Open with" and select "Connect more apps", where you can add Colab.

Next, for all the coding, please refer to the following colab notebook CS146-Winter2024-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.

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

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

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) **(5 pts)** 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:

- **workclass**: both the $>$ 50K and $<$ 50K groups are most likely to belong to category 1.
- **education**: both the $>$ 50K and $<$ 50K are most likely to fall into either category 1 or 4, and there is a disproportionate amount of the $>$ 50K group in category 10.
- **marital status**: The vast majority of the $>$ 50K group fall into category 1, while the $<$ 50K group is split relatively evenly between categories 1, 2, and 3.
- **education**: A disproportionate about of the $<$ 50K group fall into category 3, while most of the $>$ 50K group fall into categories 4 and 6.
- **relationship**: The vast majority of the $>$ 50K group fall into category 3, while the $<$ 50K group is relatively evenly split between categories 2, 3, and 4.
- **race**: Almost all surveyed falls into category 1, although there are some in category 5.
- **native-country**: Almost all surveyed fall into category 0.
- **age**: the $<$ 50K group has the biggest peak at age 20 and decreases from there, while the $>$ 50K group has the biggest peak at 40.
- **fn1wgt**: there were more participants in the $<$ 50K category than in the $>$ 50K category.
- **education-nums**: most of the $<$ 50K participats were in categories 9 and 10, while the $> 50K$ group was split between categories 9, 10, and 14.
- **capital-gains**: almost everyone has no capital gains, but those who do are almost exclusively in the $>$ 50K category.
- **capital-loss**: almost everyone has no capital loss.
- **hours-per-week**: most people earning $<$ 50K work 40 or fewer hours per week, while most people earning $>$ 50K work 40 or more hours per week.
- **sex**: more participants were in category 0 for both groups, but the $>$ 50K group in particular was overwhelmingly in category 0.

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.[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 or 0.385.

(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: The training error is 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: The training error is 0.153 for $k = 3$, 0.195 for $k = 5$, and 0.213 for $k = 7$.

(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 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

[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(...)`.

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?

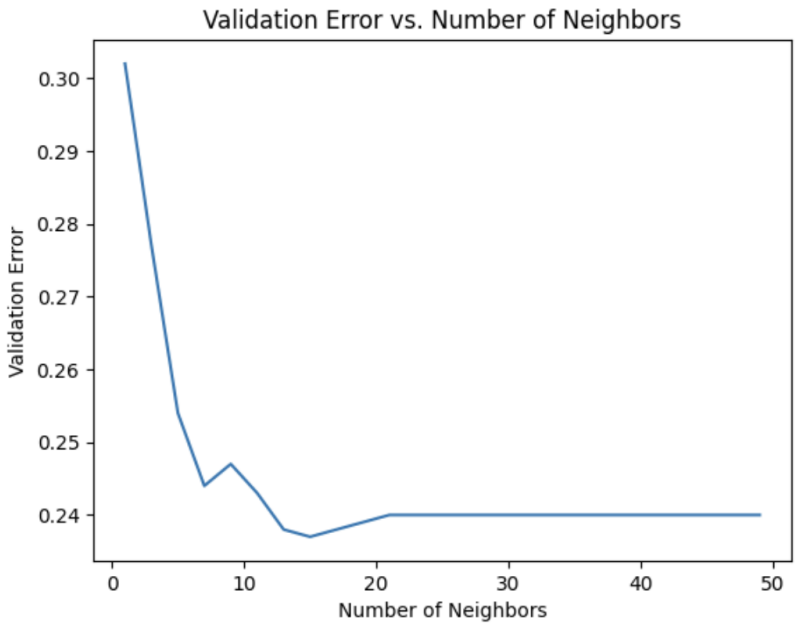
Solution:

Method	Training Error	Testing Error	F1 Score
Majority Vote	0.240	0.240	0.760
Random	0.372	0.373	0.627
Decision Tree	0.000	0.205	0.795
K Nearest Neighbors	0.202	0.259	0.741

Table 1: Performance Metrics of Different Methods

(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: As k increases from $k = 1$ to $k = 15$, the error rapidly decreases, and is minimized at $k = 15$. After that point, there is a small increase in error between $k = 15$ and $k = 20$, and the error is stable past $k = 20$. The optimal value is $k = 15$ neighbors.

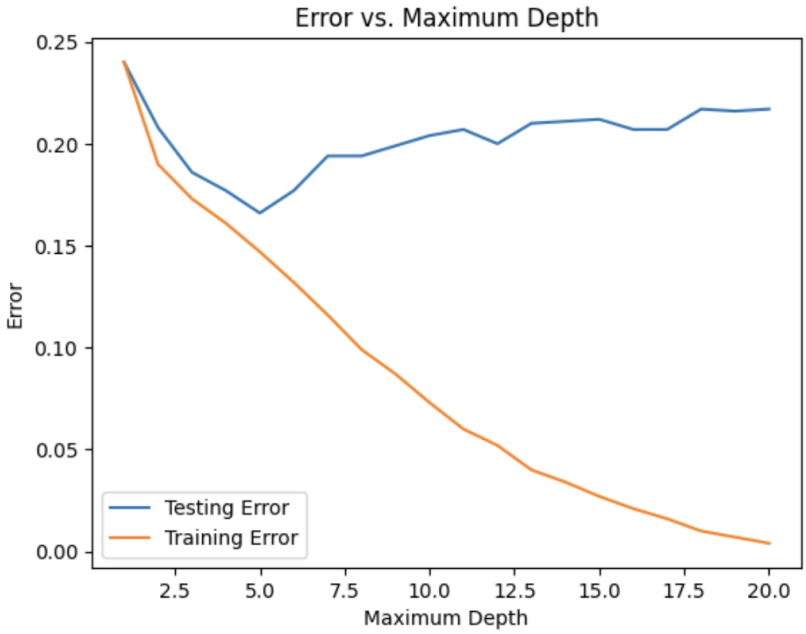


(g) **(5 pts)** One problem with decision trees is that they can *overfit* to training data, yielding complex classifiers that do not generalize well to new data. Let's see whether this is the case.

One way to prevent decision trees from overfitting is to limit their depth. Repeat your cross-validation experiments but for increasing depth limits, specifically, $1, 2, \ldots, 20$. 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 our data is 5. At a depth limit of 5, the testing error is minimized; when the depth is less than 5, the graph indicates our model underfits the data (because both training and test error are high), while for depth greater than 5, our model overfits the data (because training is low but testing error is high).



(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 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: For both the decision tree and KNN, training error increases as we include more data. Test error generally decreased for the decision tree and stayed relatively constant for KNN (see page below for graph).

(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: Performance is largely unchanged for the Majority, Random, and Decision Tree classifiers. The errors remained the same, the optimal depth was again 5 for the Decision Tree classifier, and the learning curves had little to no change. However, the performance for KNN significantly improved; the errors in part d decreased for all k values, the new optimal k was 27, and the learning curve shows a clear decrease for both training and test error.