CPSC 340 Assignment 1 (due 2018-01-17 at 9:00pm)

1 Data Exploration

1.1 Summary Statistics

Rubric: {reasoning:2}

Report the following statistics: The minimum, maximum, mean, median, and mode of all values across the dataset.

the minimum is 0.352
the max is 4.862
the mean is 1.324625
the median is 1.159
the mode is 0.77

1. The 5%, 25%, 50%, 75%, and 95% quantiles of all values across the dataset.
   the quantiles are [0.46495 0.718 1.159 1.81325 2.62405]

2. The names of the regions with the highest and lowest means, and the highest and lowest variances.
   highest variance is from variable Mtn
   lowest variance is from variable Pac
   highest mean is from variable WtdILI
   lowest mean is from variable Pac

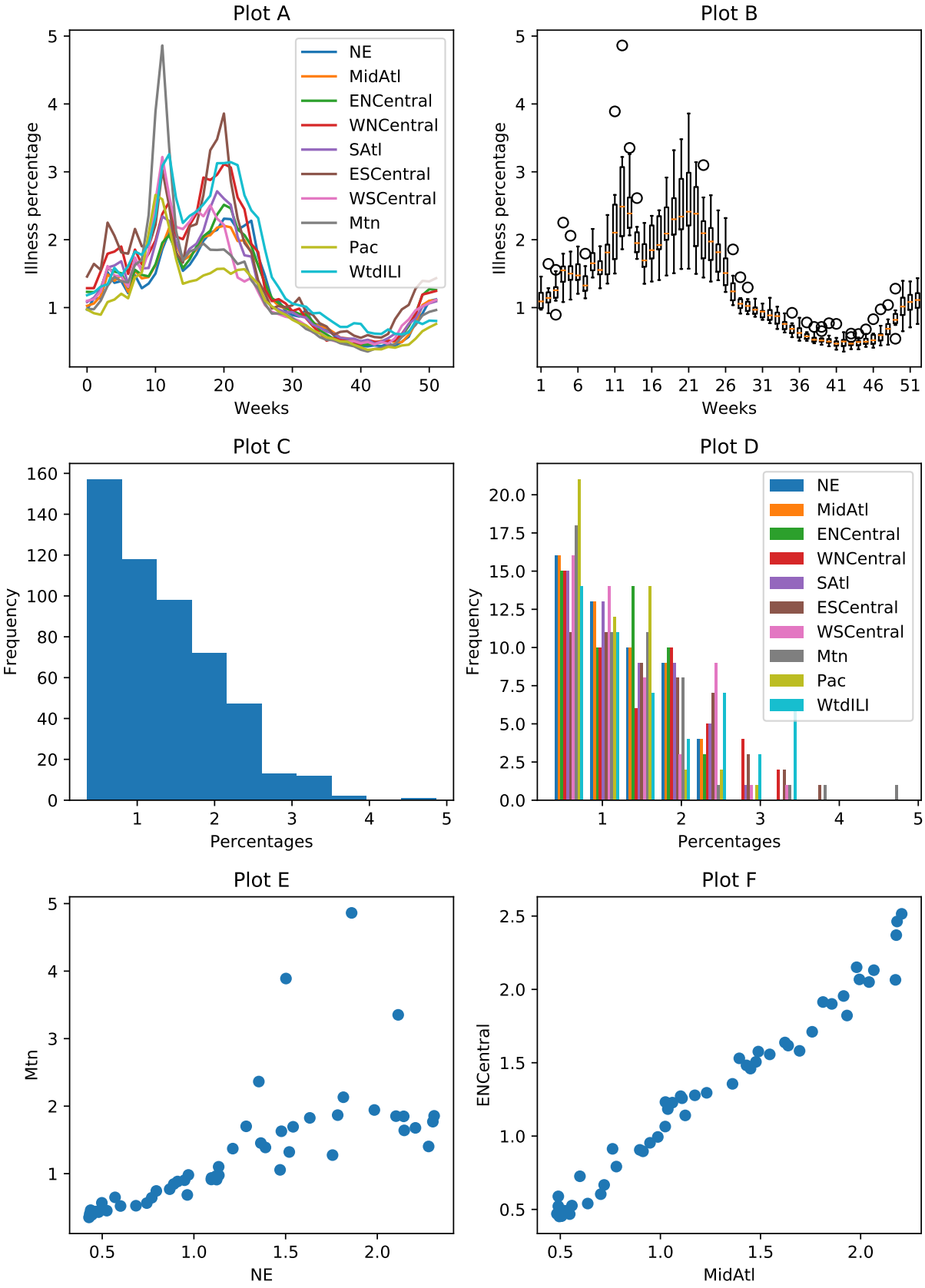
In light of the above, is the mode a reliable estimate of the most "common" value? Describe another way we could give a meaningful "mode" measurement for this (continuous) data. Note: the function `utils.mode()` will compute the mode value of an array for you.

As the all values are continuous, the mode is meaningless. Instead of that, we can round of (discretize)the values and count the mode. Or we can make a histogram and check the mode area.

1.2 Data Visualization

Rubric: {reasoning:3}

Consider the figure below.


Plot A
Illness percentage
Weeks
NE
MidAtl
ENCentral
WNCentral
SAtl
ESCentral
WSCentral
Mtn
Pac
WtdILI
Plot B
Illness percentage
Weeks
Plot C
Frequency
Percentages
Plot D
Frequency
Percentages
NE
MidAtl
ENCentral
WNCentral
SAtl
ESCentral
WSCentral
Mtn
Pac
WtdILI
Plot E
Mtn
NE
Plot F
ENCentral
MidAtl

The figure contains the following plots, in a shuffled order:

1. A single histogram showing the distribution of *each* column in X.
   D: Different color for each colum
2. A histogram showing the distribution of each the values in the matrix X.
   C: Single histogram for a matrix
3. A boxplot grouping data by weeks, showing the distribution across regions for each week.
   B: Boxplot, x axis is week.
4. A plot showing the illness percentages over time.
   A: Overtime equals to weeks, Y axis is ilness
5. A scatterplot between the two regions with highest correlation.
   F: correlation straightforward
6. A scatterplot between the two regions with lowest correlation.
   E: aweful correlation.

2 Decision Trees

2.1 Splitting rule

Rubric: {reasoning:1}

Is there a particular type of features for which it makes sense to use an equality-based splitting rule rather than the threshold-based splits we discussed in class?

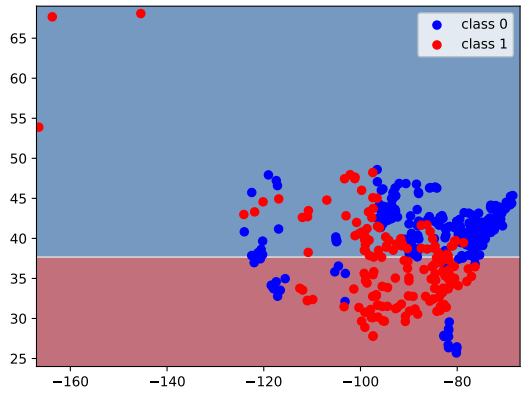
Yes. Categorical features!

2.2 Decision Stump Implementation

Rubric: {code:3}

The file *decision_stump.py* contains the class `DecisionStumpEquality` which finds the best decision stump using the equality rule and then makes predictions using that rule. Instead of discretizing the data and using a rule based on testing an equality for a single feature, we want to check whether a feature is above or below a threshold and split the data accordingly (this is the more sane approach, which we discussed in class). Create a `DecisionStump` class to do this, and report the updated error you obtain by using inequalities instead of discretizing and testing equality.

Hint: you may want to start by copy/pasting the contents `DecisionStumpEquality` and then make modifications from there.



Decision Stump with inequality rule error: 0.253

2.3 Constructing Decision Trees

Rubric: {code:2}

Once your decision stump class is finished, running `python main.py -q 2.3` will fit a decision tree of depth 2 to the same dataset (which results in a lower training error). Look at how the decision tree is stored and how the (recursive) *predict* function works. Using the same splits as the fitted depth-2 decision tree, write an alternative version of the predict function for classifying one training example as a simple program using if/else statements (as in slide 6 of the Decision Trees lecture). Save your code in a new file called `simple_decision.py` (in the *code* directory) and make sure you link to this file from your README.

Note: this code should implement the specific, fixed decision tree which was learned after calling `fit` on this particular data set. It does not need to be a learnable model.

Answer : *https* : *//github.ugrad.cs.ubc.ca/CPSC*340−2017*W*−*T*2/*k*6*y*1*b*$_a$1/*blob/master/code/decision_tree.py*

2.4 Decision Tree Training Error

Rubric: {reasoning:2}

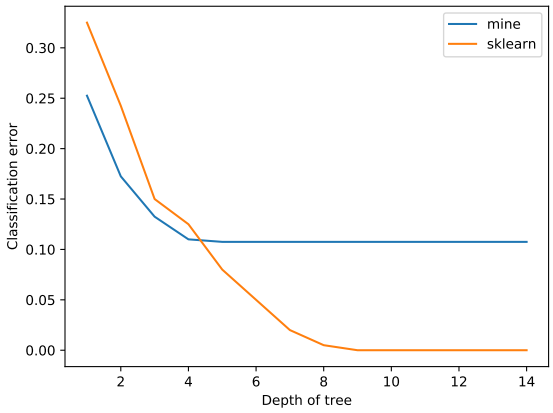
Running `python main.py -q 2.4` fits decision trees of different depths using two different implementations: first, our own implementation using your DecisionStump, and second, the decision tree implementation from the popular Python ML library *scikit-learn*. The decision tree from sklearn uses a more sophisticated splitting criterion called the information gain, instead of just the classification accuracy. Run the code and look at the figure. Describe what you observe. Can you explain the results?

Note: we set the `random_state` because sklearn's DecisionTreeClassifier is non-deterministic. I'm guessing this is because it breaks ties randomly.

Note: the code also prints out the amount of time spent. You'll notice that sklearn's implementation is substantially faster. This is because our implementation is based on the $O(n^2d)$ decision stump learning algorithm and sklearn's implementation presumably uses the faster $O(nd \log n)$ decision stump learning algorithm that we discussed in lecture.

Answer: The classification error are close for the two methods when the dept is less than 4. Mine method performs better in these cases. As the depth increses, sklearn performs better and the classification error becomes close to 0 when the depth reaches 8. Mine method has classification error fixed at 0.10. The

classification error will get harder to decrease when the depth of tree gets larger because it probably go through all the features and split values, but under information gain, it might have additional information to improve the goodness of prediction.



2.5 Cost of Fitting Decision Trees

Rubric: {reasoning:3}

In class, we discussed how in general the decision stump minimizing the classification error can be found in $O(nd \log n)$ time. Using the greedy recursive splitting procedure, what is the total cost of fitting a decision tree of depth m in terms of n, d, and m?

Hint: even thought there could be $(2^m - 1)$ decision stumps, keep in mind not every stump will need to go through every example. Note also that we stop growing the decision tree if a node has no examples, so we may not even need to do anything for many of the $(2^m - 1)$ decision stumps.

Answer: The cost in O(mnd logn) as there are m layers and each layer has n examples.

3 Training and Testing

If you run `python main.py -q 3`, it will load *citiesSmall.pkl*. Note that this file contains not only training data, but also test data, `X_test` and `y_test`. After training a depth-2 decision tree it will evaluate the performance of the classifier on the test data.[1] With a depth-2 decision tree, the training and test error are fairly close, so the model hasn't overfit much.

3.1 Training and Testing Error Curves

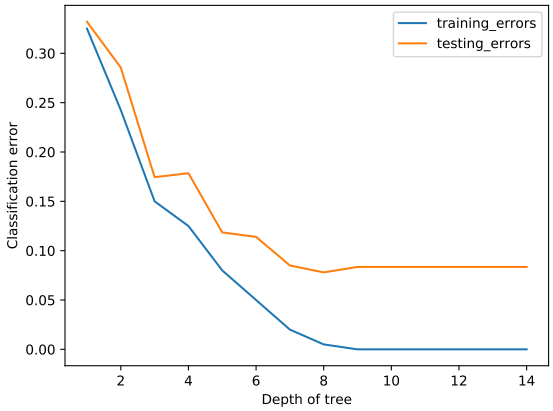
Rubric: {reasoning:2}

Make a plot that contains the training error and testing error as you vary the depth from 1 through 15. How do each of these errors change with the decision tree depth?

Note: it's OK to reuse the provided code from part 2.4 as a starting point.

[1]The code uses the "information gain" splitting criterion; see the Decision Trees bonus slides for more information.

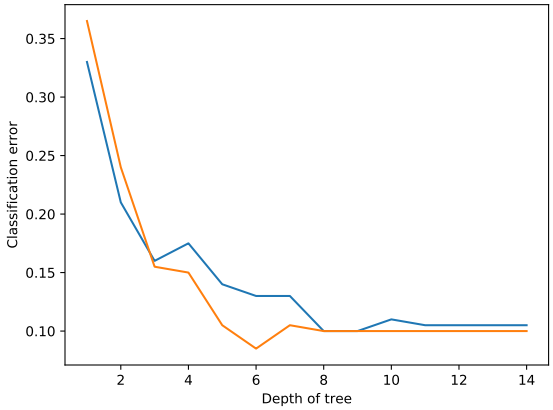
When the depth increase, both errors decrease till dept is around 8. Training error decreases till 0 while testin error gradually decreases to 0.10



3.2 Validation Set

Rubric: {reasoning:3}

Suppose that we didn't have an explicit test set available. In this case, we might instead use a *validation* set. Split the training set into two equal-sized parts: use the first $n/2$ examples as a training set and the second $n/2$ examples as a validation set (we're assuming that the examples are already in a random order). What depth of decision tree would we pick to minimize the validation set error? Does the answer change if you switch the training and validation set? How could use more of our data to estimate the depth more reliably?



The dept to minimize validation error is 8. After switching the depth of tree to minimize error is 6. We can use cross-validation to be more reliable.

4 K-Nearest Neighbours

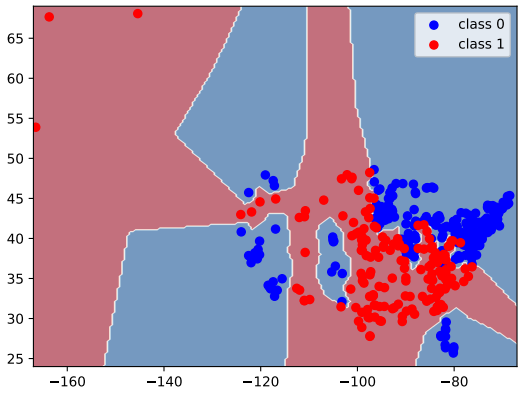
In the *citiesSmall* dataset, nearby points tend to receive the same class label because they are part of the same U.S. state. This indicates that a k-nearest neighbours classifier might be a better choice than a decision tree. The file *knn.py* has implemented the training function for a k-nearest neighbour classifier (which is to just memorize the data).

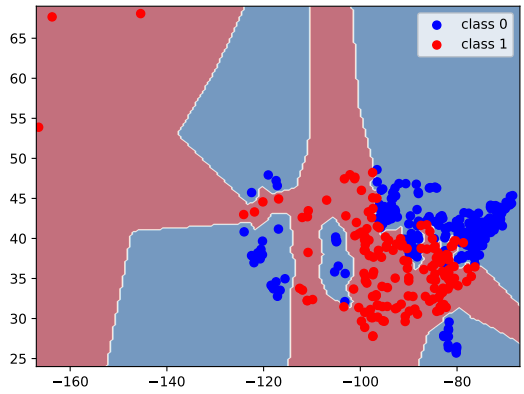
4.1 KNN Prediction

Rubric: {code:3, reasoning:4}

Fill in the *predict* function in *knn.py* so that the model file implements the k-nearest neighbour prediction rule. You should Euclidean distance, and may numpy's *sort* and/or *argsort* functions useful. You can also use *utils.euclidean_dist_squared*, which computes the squared Euclidean distances between all pairs of points in two matrices.

1. Write the *predict* function.
   https : *//github.ugrad.cs.ubc.ca/CPSC*340 − 2017*W* − *T*2/*k*6*y*1*b*$_a$1/*blob*/*master*/*code*/*knn.py*
2. Report the training and test error obtained on the *citiesSmall* dataset for $k = 1$, $k = 3$, and $k = 10$. How do these numbers compare to what you got with the decision tree?
   KNClassifier test error k=1: 0.065
   KNClassifier training error k=1: 0.000
   KNN test error k=3: 0.066
   KNN training error k=3: 0.028
   KNN test error k=10: 0.097
   KNN training error k=10: 0.072
   The errors are smaller than decision tree.
3. Hand in the plot generated by *utils.plotClassifier* on the *citiesSmall* dataset for $k = 1$, using both your implementation of KNN and the KNeighborsClassifier from scikit-learn.





   First one is KNClassifier, second one is KNN
4. Why is the training error 0 for $k = 1$?
   Training error is 0 for k=1 because KNN use the point itself to predict as the 1-nearnest neighborhood is just the point itself. Hence the training error should be 0 always.

5. If you didn't have an explicit test set, how would you choose k?
   We can use cross-validation or seperation the dataset into training and validation to find the optimal k.

4.2 Condensed Nearest Neighbours

Rubric: {code:3, reasoning:5}

The dataset *citiesBig1* contains a version of this dataset with more than 30 times as many cities. KNN can obtain a lower test error if it's trained on this dataset, but the prediction time will be very slow. A common strategy for applying KNN to huge datasets is called *condensed nearest neighbours*, and the main idea is to only store a *subset* of the training examples (and to only compare to these examples when making predictions). If the subset is small, then prediction will be faster.

The most common variation of this algorithm works as follows:

initialize subset with first training example;
for *each subsequent training example* **do**
  if *the example is incorrectly classified by the KNN classifier using the current subset* **then**
    add the current example to the subset;
  else
    do *not* add the current example to the subset (do nothing);
  end
end

Algorithm 1: Condensed Nearest Neighbours

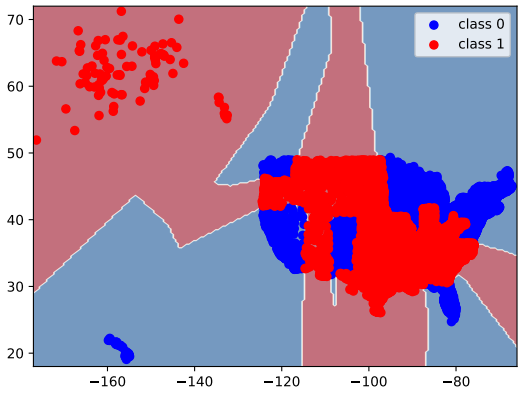
You are provided with an implementation of this *condensed nearest neighbours* algorithm in *knn.py*.

Your tasks:

1. The point of this algorithm is to be faster than KNN. Try running the condensed NN on the *citiesBig1* dataset and report how long it takes to make a prediction. What about if you try to use KNN for this dataset – how long did it take before you panicked and went for CTRL-C... or did it actually finish?
   CNN took 3.954691 seconds
   KNN took 34.608994 seconds
   CNN is better

2. Report the training and testing errors for condensed NN, as well as the number of variables in the subset, on the *citiesBig1* dataset with $k = 1$.
   CNN test error k=1 citiesBig1: 0.018
   CNN training error k=1 citiesBig1: 0.008
   There are 457.000 variables

3. Hand in the plot generated by *utils.plotClassifier* on the *citiesBig1* dataset for $k = 1$.

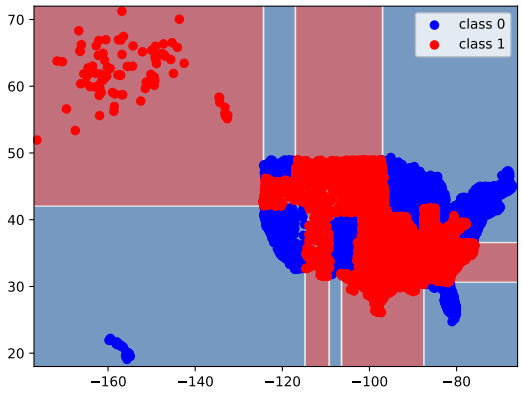


4. Why is the training error with $k = 1$ now greater than 0?
   This is because CNN does not just store the correctly classified examples but is able to add incorrectly classified examples into the training process.

5. If you have s examples in the subset, what is the cost of running the predict function on t test examples in terms of n, d, t, and s?
   O(sdt)

6. Try out your function on the dataset *citiesBig2*. Why are the test error *and* training error so high (even for $k = 1$) for this method on this dataset?
   CNN test error k=1 citiesBig2: 0.210
   CNN training error k=1 citiesBig2: 0.138
   Training error is large cuz the previously correctly classified examples would become incorrectly classified by CNN. The test error is high because there are a lot more data now and the training dataset might not be representative enough to correctly classify the test set.

7. Try running a decision tree on *citiesBig1* using scikit-learn's `DecisionTreeClassifier` with default hyperparameters. Does it work? Is it fast? Any other thoughts? Overall, do you prefer decicison trees of (condensed) KNN for this data set? Include the usual plot of the decision surface.
   Decision Tree Training error citiesBig1: 0.000
   Decision Tree Testing error citiesBig1: 0.011
   Decision Tree took 0.037795 seconds
   It is fast!
   I prefer decision trees for this data cuz it takes shorter time and the test error is not high.



5 Very-Short Answer Questions

Rubric: {reasoning:8}

Write a short one or two sentence answer to each of the questions below. Make sure your answer is clear and concise.

1. What is one reason we would want to look at scatterplots of the data before doing supervised learning?
   To detect outliers. Or because the problem may be really easy.
2. What is a reason that the examples in a training and test set might not be IID?
   The population distribution is changing.
3. What is the difference between a validation set and a test set?
   Validation set is for approximating test error, and it is a part from the training set which is not used for training.
   test set is a totally different data set which is not from training set, so it cannot affect the training process, nor sometimes have labels.
4. What is the main advantage of non-parametric models?
   More data, more complicated model.
5. A standard pre-processing step is "standardization" of the features: for each column of X we subtract its mean and divide by its variance. Would this pre-processing change the accuracy of a decision tree classifier? Would it change the accuracy of a KNN classifier?
   For decision tree, NO. This is because in decision tree, we only look at one feature for each step, so scaling doesn't have any point.
   For KNN dlassifier, Yes. This is because we calculate the distance given by the dimension from the number of features, and we need to scale it so as to make every feature have equal importance.
6. Does increasing k in KNN affect the training or prediction asymptotic runtimes?
   No. Training process contains storing datasets. Prediction process is mainly about calculation distances
7. How does increase the parameter k in k-nearest neighbours affect the two parts (training error and approximation error) of the fundamental trade-off (hint: think of the extreme values).
   Increasing K will lead to bigger training error, and less approximation error.
8. For any parametric model, how does increasing number of training examples n affect the two parts of the fundamental trade-off.

Increasing n will lead lower approximation error, but it will not necessary make the training error smaller.