Assignment1

1 Data Exploration

1.1 Summary Statistics

$\underline{SummaryStatistics:}$

minimum = 0.352

maximum = 4.862

mean = 1.324625

median = 1.159

mode = 0.77

1) 5% = 0.46495

25% = 0.718

50% = 1.159

75% = 1.81325

95% = 2.62405

2) Highest Mean = WtdILI

Lowest Mean = Pac

Highest Variance =Mtn

Lowest Variance = Pac

1.2 Data Visualization

1.D:histogram with label for each column

2.C:histogram for percentage but not categorized by region

3.B: matches because is the only boxplot

4.A: only graph labels illness percentage

5.F: comparing 2 regions, correlation is stronger than E

6.E: comparing 2 regions, correlation is weaker than F

Mode not necessarily reliable estimate of most “common value” because the distribution is continuous so the value we got doesn’t necessarily occur the most amount of times, it could not be a number in the distribution at all. A more reliable manner of attaining the most common value would be to establish a range of values like a histogram, and attain a mode from that.

2 Decision Trees

2.1 Splitting rule

Is there a particular feature where equality based splitting makes more sense?

- Yes, there is.
  - You would want to do equality based splitting for "categorical" features where the features have been split in multiple columns
  - The only values are 0 and 1, so it doesn't make much sense to do threshold based splitting

2.2 Decision Stump Implementation

Mode predictor error: 0.415
Decision stump with inequality rule error: 0.380
Decision stump with threshold rule error: 0.265

2.3 Constructing Decision Trees

2.4 Decision Tree Training Error

- For our naive implementation:
  - As we continue to train trees with higher depth, we experience overfitting
  - Our model becomes too specific and begins to predict based on particular pecuiliarities of our training dataset
  - Turns out this isn't very accurate when applied to other data that we haven't seen before
    - Hence our success rate stays constant (in some cases, the accuracy would get even worse)
- For sckit-learn implementation:
  - sckit-learn doesn't use accuracy score to build their decision trees.
  - sckit-learn uses information gain as their approach.

Information gain cares about the number of examples that get split into each branch for a given rule. For example, if we have a rule that splits our dataset 90-10, and a rule that splits our dataset 60-40, a decision tree using information gain would pick the latter rule, even if it has a lower accuracy score.

This is because by having a rule that more evenly splits the dataset, we learn more about how to classify an example. If we have a rule that doesn't split our dataset very evenly, then essentially we are only classifying a few specific examples which doesn't teach us very much. This allows the model that uses information gain to be much more accurate when generalized to new data as opposed to the accuracy score.

2.5 Cost of Fitting Decision Trees

We know that creating a decision stump (with the best rule) costs O(ndlogn) time.

- However, each object appears only once at each depth. We always have n objects at each depth.
  - This is because we split the dataset.
    - As an example, for depth = 2, let's say we get two datasets of n/2
      - The runtime would be O(d(n/2)log(n/2)) + O(d(n/2)log(n/2)), which ultimately just sums up to O(ndlogn)
        - Hence, the runtime is O(mndlogn)

3 Training and Testing

3.1 Training and Testing Error Cruves

Training vs. testing plot provided. They both start pretty high and then training error gets really low while testing error plateaus above it.

3.2 Validation Set

Validation set test error vs. depth provided 6 would be a good depth for the decision tree because at this point there is a dip in the error (it goes down to 0.215). Yes the answer does change if we switch the two, in this case we would pick a depth of 4 instead because it decreases faster in the beginning. Using more data would make the model more reliable because it would have seen a greater variety of cases so the depth would more accurately match the data.

4 K-Nearest Neighbours

4.1 KNN Prediction

- k = 1
    - 0.0645
- k = 3
    - 0.066
- k = 10
    - 0.097

Much lower than the decision tree.

```
4. The training error is 0 for k = 1 because the absolute most nearest neigh
bor is the training point itself.

5. Use a validation set to see which k gets the smallest validation error, t
hen pick that k. Also use cross-validation, as this will reduce the chances
that we get a low validation error due to by chance being able to predict a
specific validation set well.
```

4.2 Condensed Nearest Neighbours

1. Took around 30 seconds to a minute. Using KNN took much longer, so we simply just ctrl + c'd out.
2. Training error for CNN (k = 1) is 0.0075. Testing error for CNN (k = 1) is 0.0175. There were 457 objects in the subset.
3. Subset may not contain the training point being looked at, hence the most nearest neighbor is no longer the training point itself, leading to possible mislabelling.
4. The runtime is O(tsd). For every t, we have to compare the distances between all the examples in s, and choose the closest ones.
5. A lot of the states in the training dataset are just completely missing, leading to a lot of alternating between "red" and "blue" clusters of data points. This leads to high training error because the more alternating clusters you have, the more points you are going to get wrong as you begin to predict a cluster (because your subset at the time will only contain points of the other label). We also have a high testing error because the missing states in the training subset mislead the model to predict the opposite label. For example, the middle portion of the training dataset is missing a huge chunk of the red states - the model incorrectly believes that the nearest neighbors are blue (and thus incorrectly classifies the points as blue), when they should be red.
6. Yes, it works, and is quite fast. In fact, the speed is comparable to CNN (or perhaps even faster). It is also more accurate, so I'd prefer using decision trees for this. Perhaps our training dataset model just so happened to be good at predicting our testing dataset model, which seems slightly odd.

5 Very-Short Answer Questions

1. Would want to analyse the complexity and the general realtionship of the variables in the problem first
2. They might not be IID because its hard to keep all the factors independent, many factors could influence the results and relationships between the features - ex. the days you collected the data are not independent of each other. IID is generally not true.
3. Validation set is a subset of the training data, while the test set is something the program hasn't seen before. The computer should never interact with the test data.
4. The main advantage is that we don't have to stick to a fixed number of parameters to the model can be as complex as we'd like
5. It wont effect decision trees, but it will effect KNN because if the distances are too great, its a problem for KNN because it may not find its neighbours, because they're too far away
6. As n grows to infinity- assuming that k is a constant rather than some function of n, it doesn't effect the runtime since the big O of the function is (n^2*d) - independent of the k variable.
7. Increasing k would increase the training error and decrease the approximation error because by increasing k we simplify the required similarities - making the training broad but making real-life testing less similar to the test requirements
8. Increasing the number of training examples would increase the training error, since the program is seeing more varied data, and it would decrease the approx. error because the training error would be a better approximation of the testing error.

```
In [ ]:
```