Ross Brancati
CS589 Machine Learning
Homework 1: Classification
Due: 9/17/21 (I was granted an extension to Monday 9/27 for a family emergency)

Part 1: K Nearest Neighbors

1.

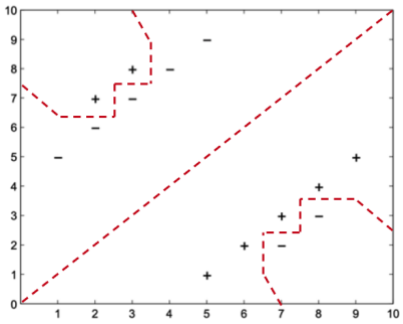


Figure 1: 1-nearest neighbor decision boundary

Too large of a value for k could be bad because it could misclassify a sample point in which someone is trying to predict the class of, or underfit the data. For example, if we had a point in the top left corner, within the decision boundary for the + label, but we had k set to some value between 5 and 7, it would classify this point as a – label instead. Alternatively, if k was set to too small of a value, we could overfit the data which would not classify test points accurately.

2.

Table 1: KNN F1 scores for values of k.
The F1 score increases up to k = 10, then slightly falls decreases.

KNN F1 Scores	
K	F1_Score
3	0.5660
5	0.5677
10	0.6040
20	0.6027
25	0.5816

3. F1 score accounts for an uneven number of positive and negative labels, or an unbalanced dataset. F1 score is a harmonic mean of *precision*: a measure of correctly identified positive cases and *recall*: a measure of the correctly identified positive cases from all true positive cases. In short, F1 score accounts for false positives and false negative predictions. For example, let's say there are 90 positive values and 10 negative values in a dataset and the KNN algorithm predicted all 90 positives correctly, but predicted 0 negative values correctly. This would make your accuracy for this prediction 0.90 (90%). However, the accuracy metric fails to account for those negative classes that were predicted incorrectly. In this Pima Indian Diabetes Dataset, specifically the x_test data, there are only 81 samples that have the class of positive for diabetes, compared to 150 negative classes. My algorithm predicted about 70-75% accuracy for the values of *k* tested, but clearly the F1 scores are vastly different from the accuracy.

Part 2: Decision Trees

1. The criteria used for selecting the variable for a node when training a decision tree is the *Gini Index*, which is a metric that describes the probability that a class of a variable will be incorrectly classified if it is randomly chosen. To calculate the Gini index, we perform a mock split of the data for each value of the feature and calculate the Gini index based on that split. Each value of the feature is tested, and the value with the lowest Gini index is chosen as the value which we split the data on. In a decision tree, the variable or feature of the root node is selected as the one that has the lower Gini index. As we move down the decision tree, the Gini index is calculated at each child node, and the variable with the lowest Gini index is again selected as the variable to split the data on. This method is not optimal because we must use brute force at each node to calculate the best Gini index. In modern day machine learning, the speed at which an algorithm can run is almost as important as producing accurate predictions. Other metrics such as mean or median could be used with features that contain continuous data to help improve the speed of classification, but these are not as accurate as the brute force method. To summarize, brute force Gini index calculation in decision trees is greedy, which improves accuracy, but sacrifices time.

2.

Table 2: Decision Tree F1 scores for increasing tree depths.
The F1 score increases up to a depth of 5, then begins to slightly decrease.

Decision Tree F1 Scores	
Depth	**F1_Score**
2	0.5362
3	0.6486
5	0.6595
7	0.6279
9	0.6071
11	0.6071

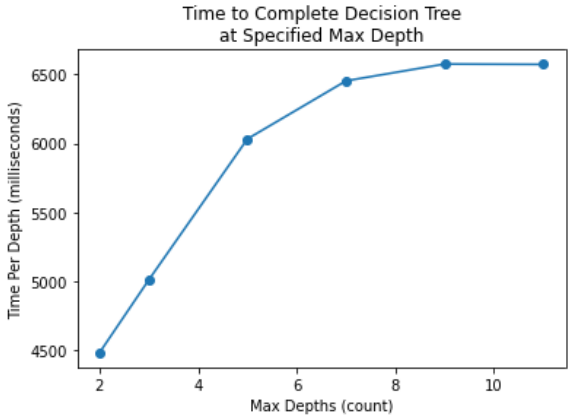


Figure 2: Time to complete classification task at specified depths. As the depth of the tree increases, the time it takes to perform classification and prediction also increases.

3. With the original *max_depths* list provided in the shell code, I found that the F1 score did not appear to improve if I the depth was greater than 5. So, I decided to test a new list of depths around the best performing maximum depth (max_depths = [4, 5, 6, 7, 8]) while keeping the *min_size* hyperparameter at 5, which yielded the following F1 scores:

Table 3: Decision Tree F1 after tuning maximum depth. The depth that appears to yield the best F1 score is still 5.

Decision Tree F1 Scores	
Depth	**F1_Score**
4	0.6188
5	0.6595
6	0.6061
7	0.6279
8	0.5952

Next, I wanted to try testing different minimum terminal node sizes to see if that improved the performance of my model. I kept the depth at 5, which has produced the best results thus far. Not surprisingly, as we reduced the number of observations in the terminal node, the time to complete the classification task declined sharply from 1 observation at the terminal node to 2 observations at the terminal node, then plateaued.

Table 3: Decision Tree F1 Score after tuning minimum number of observations at terminal node. A smaller number of observations at the terminal node does not seem to significantly improve the F1 score.

Decision Tree F1 Scores	
Depth	**F1_Score**
1	0.6667
2	0.6667
3	0.6667
4	0.6595
5	0.6595
6	0.6595
7	0.6595

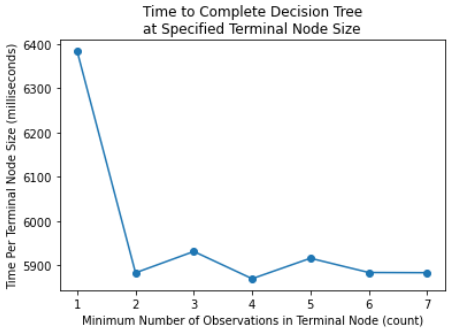


Figure 3: Time to complete classification task at different number of observations in the terminal node. F1 score does not seem to improve with a smaller number of observations, but the time to complete the classification task sharply declines from one terminal observation to 2 terminal observations.

From the above grid searching, the optimal depth of my decision tree is 5, and the optimal number of observations at the terminal node can be kept at 5, because changing the number of observations to smaller values does not significantly improve performance. It should be noted that when the number of observations at the terminal node is between 2 and 7, the time to complete the classification is approximately the same.

Part 3: Probability and Estimation

1.

Which class has higher posterior probability?

① We have $P(w|y=1)$ and $P(w|y=0)$

- $P(y=1|w) = \frac{P(w|y=1)P(y)}{P(w)}$

$P(y=1|w_1) = \frac{P(w_1|y=1)P(y=1)}{P(w_1)} = \frac{(0.2)(0.3)}{0.6} = 0.1$

$P(y=1|w_2) = \frac{P(w_2|y=1)P(y=1)}{P(w_2)} = \frac{(0.3)(0.3)}{0.8} = 0.1125$

$P(y=1|w_3) = \frac{P(w_3|y=1)P(y=1)}{P(w_3)} = \frac{(0.5)(0.3)}{0.6} = 0.25$

↳ $P(y=1|w) = (0.1)(0.1125)(0.25)$

$\boxed{P(y=1|w) = 0.0028125}$

- $P(y=0|w) = \frac{P(w|y=0)P(y)}{P(w)}$

$P(y=0|w_1) = \frac{P(w_1|y=0)P(y=0)}{P(w_1)} = \frac{(0.4)(0.7)}{0.6} = 0.467$

$P(y=0|w_2) = \frac{P(w_2|y=0)P(y=0)}{P(w_2)} = \frac{(0.5)(0.7)}{0.8} = 0.4375$

$P(y=0|w_3) = \frac{P(w_3|y=0)P(y=0)}{P(w_3)} = \frac{(0.1)(0.7)}{0.6} = 0.117$

↳ $P(y=0|w) = (0.467)(0.4375)(0.117)$

$\boxed{P(y=0|w) = 0.0239}$

The class *y=0* has higher posterior probability according to the above calculations.

What is the posterior probability that the document is relevant/useful?

The posterior probability that the document is relevant/useful is:

$\vec{x} = (0, 1, 1)$

↳ Probability that $\vec{x}$ belongs to class $y=1$

$P(y=1|\vec{x}) = \frac{P(\vec{x}|y=1)P(y=1)}{P(\vec{x})}$

$= \left(\frac{P(w_2|y=1)P(y=1)}{P(w_2)}\right)\left(\frac{P(w_3|y=1)P(y=1)}{P(w_2)}\right)$

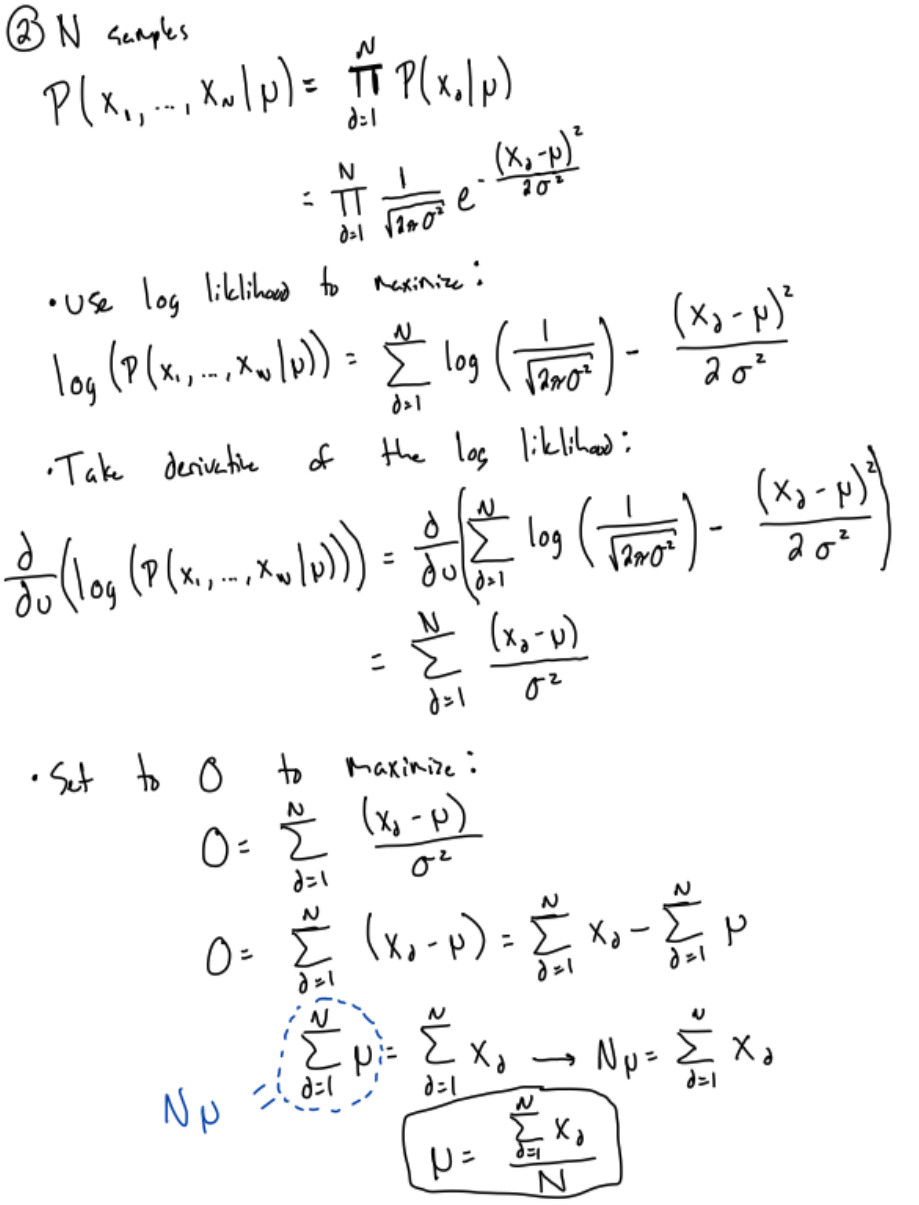
$P(y=1|\vec{x}) = (0.1125)(0.25)$

$\boxed{P(y=1|\vec{x}) = 0.028125}$

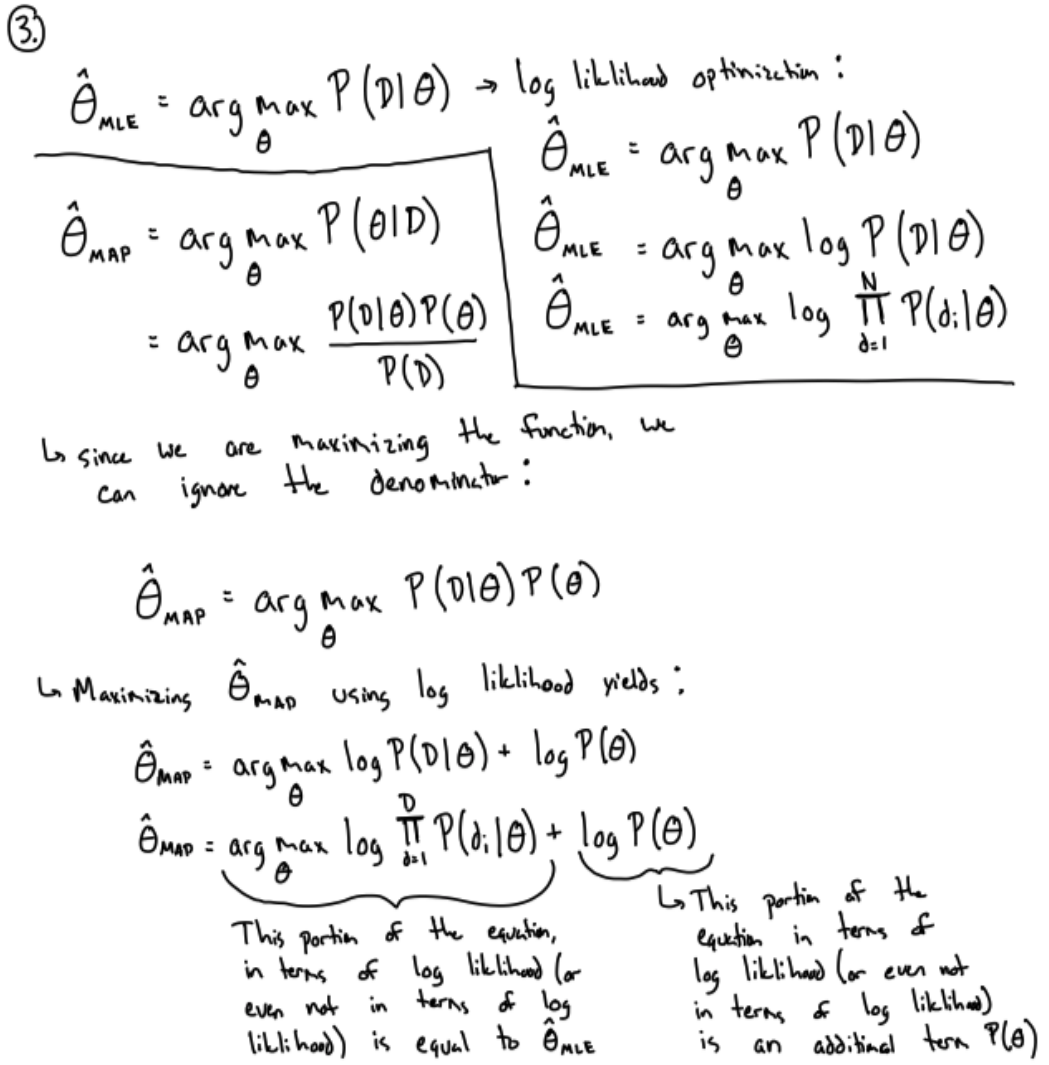
What is the naïve assumption in this problem?

The naïve assumption in this problem is that the words in the document are conditionally independent of each other (ie all features are conditionally independent)

2.



3.
Mathematically show how the MAP estimation equals the MLE estimation plus an additional term, log P(θ).



The prior in MAP represents a weighting term from prior data, so if this is in some way equivalent to 0, the MLE and MAP estimators will be identical. Since we are maximizing using log likelihood, constant values of the prior will be equal to 0, because we will eventually take the derivative of that term. That being said, if the prior estimate is equal for all features (ie all features are identically weighted), P(θ) will be constant, and the derivative of term upon maximization will be equal to 0. Therefore, MLE and MAP estimates will also be equivalent.