

# Project 3: k-Nearest-Neighbors and Decision Tree's

CS425

Samuel Steinberg

October 29<sup>th</sup>, 2019

## Introduction:

This paper analyzes the implementation of the k-Nearest-Neighbors (kNN) algorithm and a univariate decision tree to classify sample instances of potential breast cancer as benign or malignant. The kNN algorithm is a non-parametric method used for classification where an object is classified by the plurality vote of its neighbors. Since it is non-parametric, it does not take in any assumptions about the data. Decision trees are hierarchal data structures that implement the divide-and-conquer strategy and are built with labeled training data. They are non-parametric and the goodness of the split is quantified by an impurity measure. The resulting classifications output by the algorithms are evaluated using a confusion matrix, with performance metrics such as accuracy, true positive rate (TPR), positive predictive value (PPV), true negative rate (TNR) and F1 Score being collected and analyzed to find the optimal parameters for accurate classification. The database being tested in this paper is called *breast-cancer-wisconsin.data*, which contains 699 entries of patient data and attributes.

## Data Pre-Processing:

Data was given for potential cases of breast cancer in the form of 11 attributes:

1. Sample code number: id number
2. Clump Thickness: 1-10
3. Uniformity of Cell Size: 1-10
4. Uniformity of Cell Shape: 1-10
5. Marginal Adhesion: 1-10
6. Single Epithelial Cell Size: 1-10
7. Bare Nuclei: 1-10
8. Bland Chromatin: 1-10
9. Normal Nucleoli: 1-10
10. Mitoses: 1-10
11. Class: (2 for benign, 4 for malignant)

The first step taken was to drop useless attributes, which in this data set was just the sample code number, reducing the dimension of the data to 10. After this, mean imputation was imposed on missing data to get a realistic value in place of the missing one. Finally, the Class attribute was removed from the matrix of attributes. This is what the remaining nine attributes are attempting to classify. The data was then split into testing, training, and validation sets. This allows for the training of the model, to then perform testing on the trained model, and to validate.

### Solution Description:

After data pre-processing is complete, it is possible to begin computation. The remaining nine numerical attributes that are to be used to predict and classify cancer as benign or malignant are input into a matrix. This is necessary to run the kNN algorithm, which uses Euclidean distance for n-dimensional space. See Figure 1 below:


$$D(a, b) = \sqrt{\sum_{i=1}^n (b_i - a_i)^2}$$


Figure 1: Euclidean Distance Formula on left, visual display to the right

Here, the distance between points a and b is the length of the line segment connecting them. This metric is essential to the kNN algorithm because this will be the distance metric for determining the closest neighbors between points. The classifications determined by the algorithm are then put into a confusion matrix to allow visualization of the performance. The performance of the algorithm depended on the predicted class vs. true class, as shown in Figure 2 below:

True Class	Predicted Class	
	benign	malignant
benign	TN	FP
malignant	FN	TP

Figure 2: Confusion matrix

In this matrix, TN represents a true negative, FP a false positive, FN a false negative, and TP a true positive. See Figures 3-7 for the performance metrics extracted from the matrix.

$$\text{Accuracy} = (TN + TP) / (TN + TP + FN + FP)$$

Figure 3: Accuracy Formula

The Accuracy metric represents the overall accuracy of the algorithm in the form of a percentage of how many correct classifications there were out of the total. Hence, the calculation is the number of correct (true) classifications over the total number of classifications.

$$\text{TPR (true positive rate, recall, or sensitivity)} = TP / (TP + FN)$$

Figure 4: TPR Formula

As shown in Figure 4 above, TPR is the percentage of the positive classifications determined by the algorithm over the total number of real positives. Since a FN (false negative) actually represents a truly positive case of cancer, the addition of it and the number of TP (true positives) make up the denominator for the total number of positive cases.

$$\text{PPV (positive predictive value or precision)} = \text{TP} / (\text{TP} + \text{FP})$$

Figure 5: PPV Formula

The PPV is the probability that patients with a positive classification actually have a malignant diagnosis. It takes the percentage of the number of correct positive classifications over the total number of actual positive classifications. See Figure 5 above.

$$\text{TNR (true negative rate or specificity)} = \text{TN} / (\text{TN} + \text{FP})$$

Figure 6: TNR Formula

The TNR metric is a very similar measure of sensitivity to TPR in Figure 4, but for negative classifications. It takes the number of correctly classified negative diagnoses over the total number of actual negative diagnoses.

$$\text{F}_1 \text{ Score} = 2 \times \text{PPV} \times \text{TPR} / (\text{PPV} + \text{TPR})$$

Figure 7: F<sub>1</sub> Score Formula

The F<sub>1</sub> Score is a measure of a test's accuracy using the harmonic mean of precision and recall. This helps to balance precision and recall, which at times can offer a better idea of model performance than accuracy. The balance can be seen in Figure 7 above, with PPV (precision) and TPR (recall) being balances against each other.

$$\hat{P}(C_i | \mathbf{x}, m) \equiv p_m^i = \frac{N_m^i}{N_m}$$

Figure 8: Purity Formula

Purity is the measure of the extent that clusters contain a single class. As shown in Figure 8, for each cluster  $p$  the number of data points from the most common class are taken ( $N_m^i$ ). Since this was used for decision tree classification in this project,  $m$  represents a node. Finally,  $N_m^i$  is divided by its sum over all  $i$ , represented by  $N_m$  with  $\sum_i N_m^i = N_m$ .

$$I_m = - \sum_{i=1}^K p_m^i \log_2 p_m^i$$

Figure 9: Entropy Formula

Figure 9 represents the calculation for entropy, one of the three impurity measures used in this project.  $p_m^i$  represents the purity for each point  $i$  at node  $m$ , while  $K$  represents the number of classes. Entropy is a good way to measure the extent by which cluster labels match class labels.

$$\phi(p, 1 - p) = 2p(1 - p)$$

Figure 10: Gini Index Formula

The second impurity measure explored was the Gini Index. Found with the formula in Figure 10 where  $p$  denoted purity, it measures the probability of a variable being wrongly classified when chosen.

$$\phi(p, 1 - p) = 1 - \max(p, 1 - p)$$

Figure 11: Misclassification Error Formula

Figure 11 represents the third and final impurity measure: misclassification error (where  $p$  denotes purity). This measurement can be generalized to minimum risk given a loss function. In other words, this is simply a measurement of how often the classifier is incorrect.

After impurity measures are found, it is possible to begin tree construction. Figure 12 illustrates how a tree is constructed. The GenerateTree function will create a leaf labelled by the majority class in the observation if its impurity is less than the threshold given by  $\theta_1$ . If this is not the case, then the attribute is split and a value denoting the best fit is found in the SplitAttribute function. The tree construction continues recursively and in parallel until complete purity. In this project, a maximum depth could also be specified by the user to stop construction.

```

GenerateTree( $\mathcal{X}$ )
  If NodeEntropy( $\mathcal{X}$ ) <  $\theta_1$  /* equation 9.3 */
    Create leaf labelled by majority class in  $\mathcal{X}$ 
    Return
   $i \leftarrow \text{SplitAttribute}(\mathcal{X})$ 
  For each branch of  $x_i$ 
    Find  $\mathcal{X}_i$  falling in branch
    GenerateTree( $\mathcal{X}_i$ )

SplitAttribute( $\mathcal{X}$ )
  MinEnt ← MAX
  For all attributes  $i = 1, \dots, d$ 
    If  $x_i$  is discrete with  $n$  values
      Split  $\mathcal{X}$  into  $\mathcal{X}_1, \dots, \mathcal{X}_n$  by  $x_i$ 
       $e \leftarrow \text{SplitEntropy}(\mathcal{X}_1, \dots, \mathcal{X}_n)$  /* equation 9.8 */
      If  $e < \text{MinEnt}$  MinEnt ←  $e$ ; bestf ←  $i$ 
    Else /*  $x_i$  is numeric */
      For all possible splits
        Split  $\mathcal{X}$  into  $\mathcal{X}_1, \mathcal{X}_2$  on  $x_i$ 
         $e \leftarrow \text{SplitEntropy}(\mathcal{X}_1, \mathcal{X}_2)$ 
        If  $e < \text{MinEnt}$  MinEnt ←  $e$ ; bestf ←  $i$ 
  Return bestf

```

Figure 12: Decision Tree Generation

Figure 13 shows an example of a typical tree after construction, where ovals are decision nodes and squares are leaf nodes (these will be the classifying nodes). The univariate decision tree splits a node along one axis, with successive splits orthogonal to one another. Once a split is pure it is not split further.

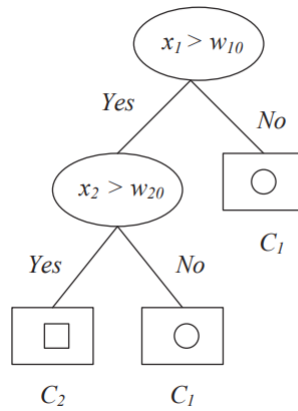


Figure 13: Typical Decision Tree Structure

## Analysis and Discussion:

After cleaning the data and building the attribute matrix, it was possible to begin work on the algorithms to predict whether a case of breast cancer was benign or malignant based on the nine remaining attributes. The first algorithm run was kNN, for which cross-validation was also performed to determine the best  $k$  (number of neighbors) and report performance. To start off the algorithm, distances are calculated for the closest distances to each training data point. After analyzing the neighborhood to evaluate each data point, they need to be classified as either benign or malignant. To do this in kNN, there needs to be a majority vote for the classification as determined by its neighbors. A confusion matrix was then created to validate the accuracy of the classification.

When training the model, performance dropped with more  $k$  neighbors. This could have been a result of how the data was split, since testing performance was fairly steady. See Figure's 14 and 15 below:

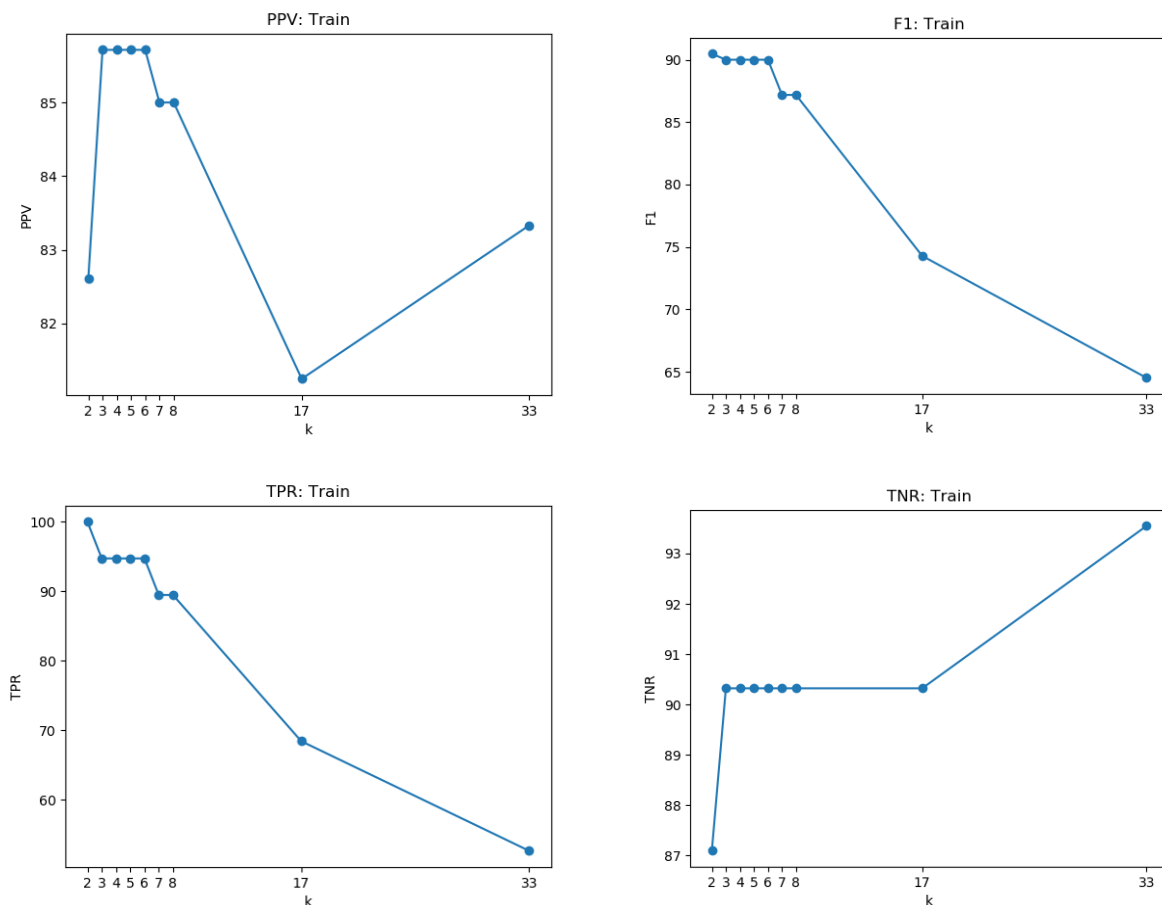


Figure 14: PPV, F1 Score, TPR, and TNR for each  $k$

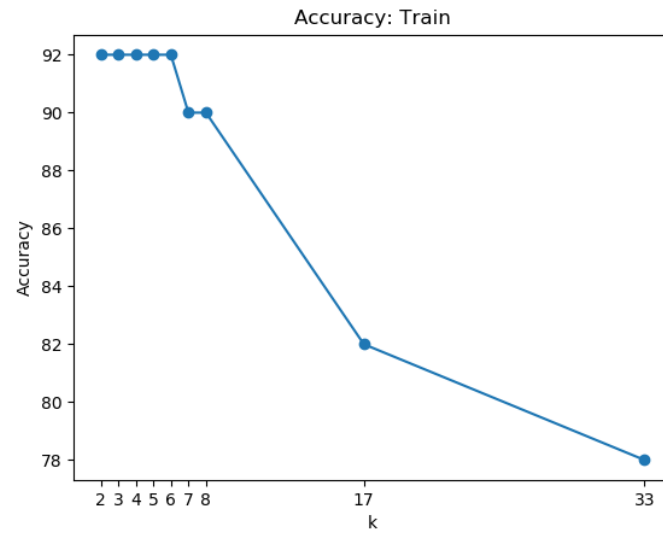


Figure 15: Accuracy for each  $k$

As seen above, performance was generally steady until  $k$  reached 17, after which there was a decline. The Accuracy and F1 Score were metrics where more importance was placed in this project, since they give the best representation of how accurate the model actually was. In the case of the training data, they had similar slopes and values which was a positive sign the model was performing at an acceptable rate. The training optimal  $k$  was confirmed when running the validation set, which also had the highest accuracy at a  $k$  of 4:

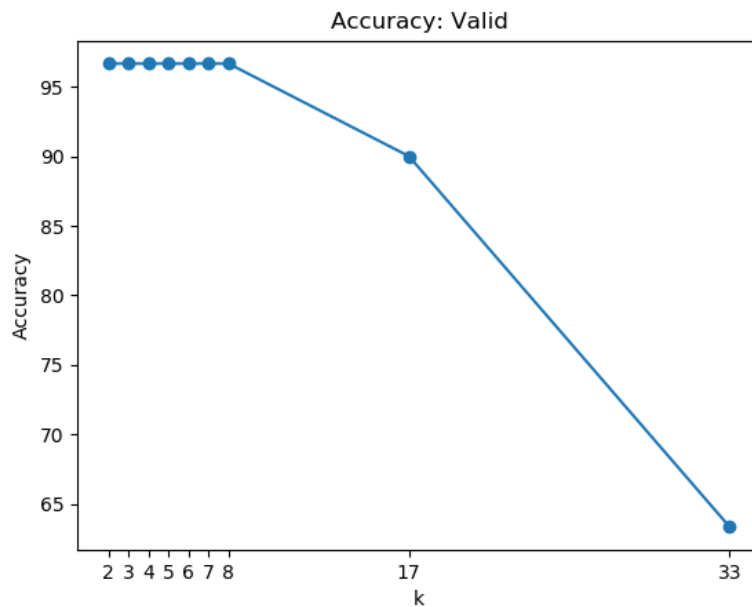


Figure 16: Validation Accuracy

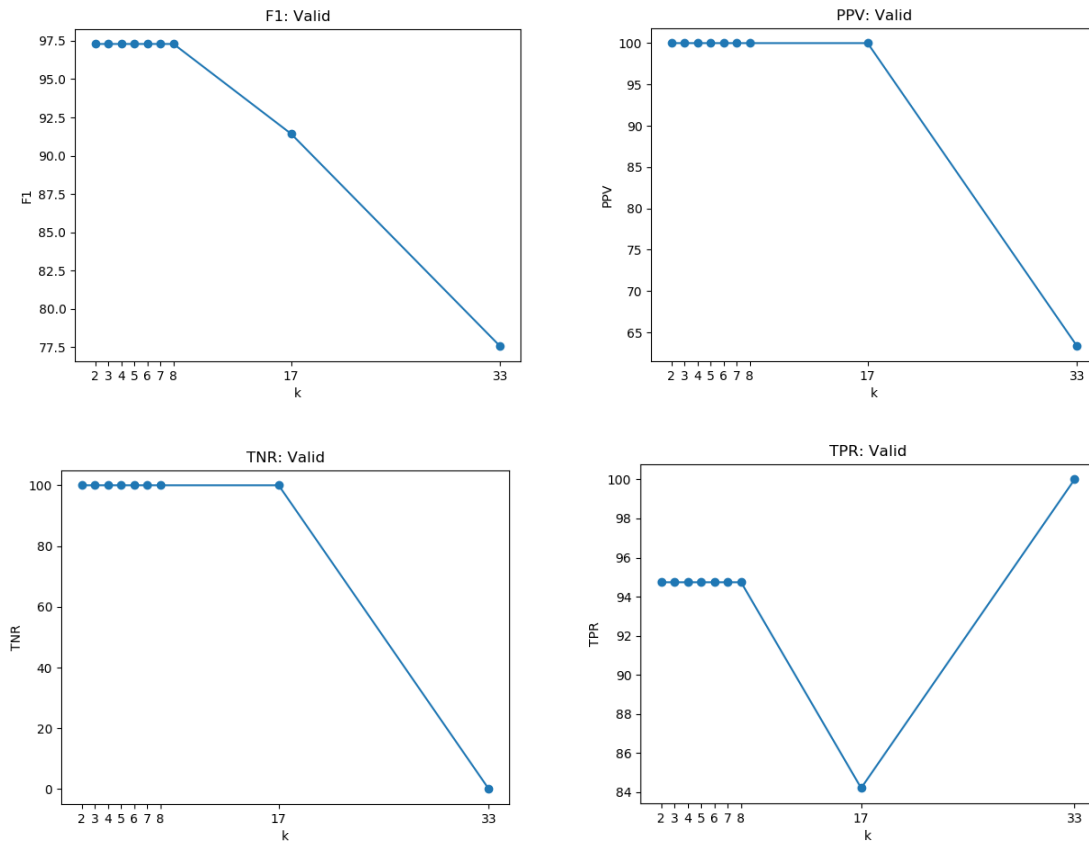


Figure 17: Validation Metrics

The best accuracy for the training data (Figure 15) was found to be around 92.01% with a  $k$  of 4, which was slightly greater than  $k$ 's of 2, 3, 5, and 6. Hence, a  $k$  of 4 was considered to be the optimal  $k$  when the validation data was also taken into account. Figure 18 shows the resulting metrics when run on the test data:

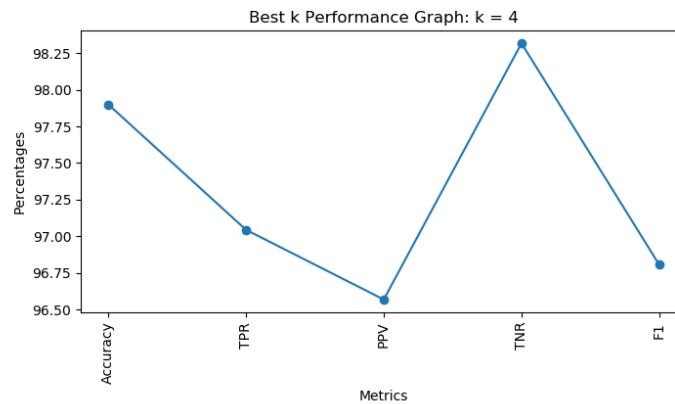


Figure 18: Testing data metrics with an optimal  $k$  of 4



As seen in Figure 18, all the metrics tested extremely well on the test data, which signaled that the model was being properly trained and validated. The metrics represented in Figure 18 are displayed below:

**Table 1: Confusion Matrix for Optimal k**

True Class	Predicted Class	
	benign	malignant
benign	409	7
malignant	6	197

**Accuracy** = 97.90%

**TPR** = 97.04%

**PPV** = 96.57%

**TNR** = 98.32%

**F1 Score** = 96.81%

These were excellent classification scores, and further confirmed that the training and validation models were accurate. Overall, the test data performed well on the model and was generally consistent. See Figure 19 for a chart of accuracies for all  $k$ :

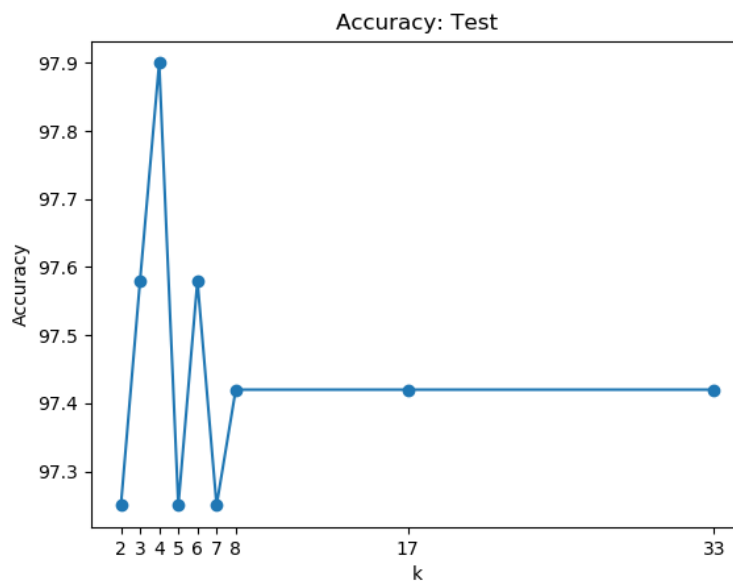


Figure 19: Accuracies for test data on all  $k$

After performing the kNN algorithm for classification, a univariate decision tree was implemented. The user was able to select an impurity measure to use (either entropy, gini, or misclassification), along with the choice to limit the tree by an impurity threshold or by specifying a maximum depth. The tree was generated using the algorithm from Figure 12, and predictions were made using the tree (see Figure 13 for an example constructed tree). For training purposes maximum depths  $k$  were tested between 2 and 10.

After training the model and applying the test data set, the optimal maximum depth  $k$  was again found to be 4. The impurity measures did not significantly differ in their performance, all finding 4 as the optimal  $k$  and having similar accuracies. They mainly differed in their optimal impurity threshold ( $\theta_1$  in Figure 12):

- Entropy: 0.2
- Gini: 0.1
- Misclassification error: 0.05

Overall, the decision tree was fairly accurate, though not as much as the kNN algorithm. Though there was not a significant performance difference between impurity threshold, entropy generally performed the best. See Figure 20 below for the optimal  $k$  max depth of 4 with entropy acting as the impurity threshold:

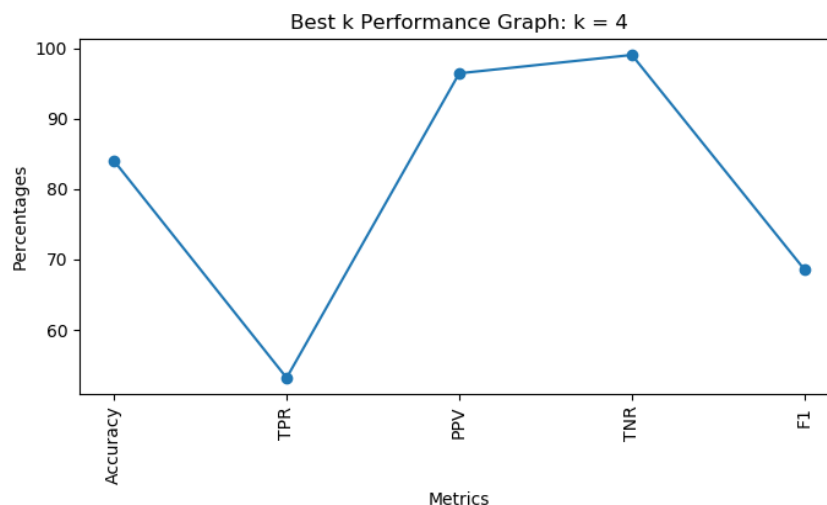


Figure 20: Optimal  $k$  with Univariate Decision Tree

As illustrated, performance was generally good. This statement is based off accuracy scoring 84.01% and an F1 score of 68.57%. The Accuracy and F1 Score were skewed by a low TPR, which could have been a result of how the sets were split. The following metrics displayed in Figure 20 are displayed below for the optimal  $k$  and optimal impurity threshold for entropy:

**Table 2: Confusion Matrix for Optimal k and Impurity Threshold Using Entropy**

True Class	Predicted Class	
	benign	malignant
benign	412	4
malignant	95	108

**Accuracy** = 84.01%

**TPR** = 53.20%

**PPV** = 96.43%

**TNR** = 99.04%

**F1 Score** = 68.57%

Table 2 clearly shows that benign classifications are being classified efficiently, though at the same time there are many more incorrect malignant classifications than there were in the kNN algorithm's optimal  $k$ . Despite this, the classification accuracy was still high enough to consider it a success.

Overall, while both the univariate decision tree classifier and kNN algorithm performed well, kNN had much better performance. Both algorithms found an ideal  $k$  to be 4, though kNN had 13.89% better accuracy and a much better F1 Score. The decision tree classifier misclassified too many malignant cases as benign, which brought the overall accuracy and F1 Score down, though it was highly proficient overall and in classifying benign cases. Entropy seemed to give the best results of the impurity thresholds in the algorithm (Figure 12), though the gini index and misclassification index did not yield entirely different results. Surprisingly, even though decision tree's have to constantly split until pure the algorithm performed much faster on the larger test sets. The kNN algorithm took notably longer when performing on the larger test set, though was quite fast on the smaller training and validation sets.