k-NN Clustering and Decision Tree Analysis

Ian R. Stewart

November 5^{th}, 2018

Fall 2018 – COSC 528

Project 3

Abstract

This project implements two approaches to predict cancerous results from five features. The first method employed is *k*-Nearest-Neighbor (*k*-NN) clustering using a variety nearest-neighbors values. A confusion metric and five metrics are reported to analyze the performance of different *k*-values for the *k*-NN, which include: model accuracy, true positive rate, positive predictive value, true negative rate, and F1-score. Analysis of the *k*-NN clustering proved that eight nearest-neighbors provided the lowest error rate on the test data with an averaged accuracy of 97.7% over ten runs. Second, a decision tree class is created with the option to employ three measures for decision tree branching: entropy, Gini index, and misclassification error. Both techniques utilized in this project are implemented in a supervised fashion, which uses a portion of the original dataset (i.e. training data) to build a predictive model and validates the model on the remaining data (i.e. test data). A maximum depth of three was found to be ideal with 97.1% accuracy, as the other values ranging from two to ten provided higher predictive error for the test dataset. To lower the dimensionality of the data, SVD was implemented to calculate the appropriate number of principal components to use while still maintaining a relatively high level of total variance explained. This approached resulted in 90% of the variance explained in the first six principal components, thus reduced the data matrix to six columns. The created *k*-NN and decision tree python functions were implemented on the lower dimensional dataset which resulted in a predictive accuracy of 94.5% and 95.9% for *k*-NN and decision tree analysis, respectively.

List of Tables

List of Figures

1 Introduction

Two common analysis to perform in the field of biostatistics and epidemiology is causation analysis and predictive model creation. These analyses attempt to find a relationship between data features in order to better control or treat medical issues. Classification is an ideal candidate for accomplishing this task, where a classifier assigns objects to predefined categories or outcomes and builds a set of rules to categorize future, unknown data. The general approach to classification follows the schematic in Figure 1, where the classification algorithm utilized can take multiple forms, such as clustering or decision tree.

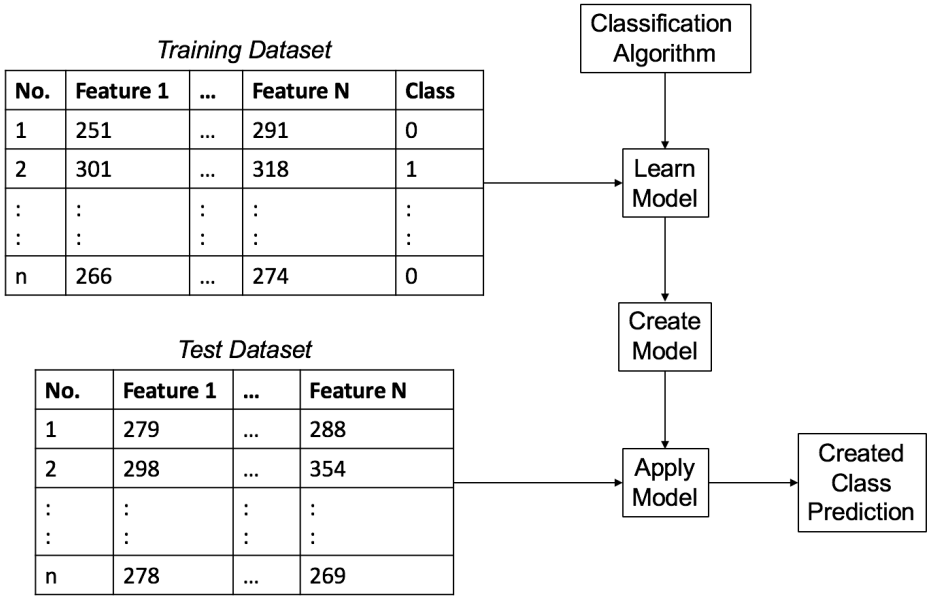

Figure 1: General overview for creating classification predictions.

This project uses a dataset that contains cancer classification (i.e., malignant or benign) along with ten attributes. The idea is to use the data attributes to predict the cancer classification for a given patient. The dataset is mean-centered and z-standardized prior to employing two classification methods: clustering and decision tree analysis.

1.1. *k*-NN Clustering Technique

A widely popular classification technique called k-Nearest-Neighbor (k-NN) clustering employed in this project attempts to cluster the data features to assist in the classification prediction process. This technique is a common choice due the output interpretability, and relatively high predictive power, but can suffer from higher calculation times for large datasets as this technique iterates over the entirety of the dataset. The k-value in k-NN is a hyperparameter that is chosen by an analyst that calculates the classification of a given data point based on the classification of the closest k number of data points. A conventional approach to choose an appropriate k-value is to assess the error rates for various parameter values. This task can be performed via a number of performance metrics, such as the training error rate and validation error rate when using a training-test split of the data. The performance metrics implemented in this project will be discussed in the analysis portion of this report. For the sake

of clarity as well as adding context to the *k*-NN technique implementation, the following steps provide a general overview of the implementation procedure:

1. Split the original data into a training and test set,
2. Iterate through the test dataset;
    a. Calculate distance between each row in the dataset to the training dataset,
    b. Sort calculated distances in ascending order,
    c. Choose most frequent classification for test point based on top *k* rows of sorted array,
3. Return classification prediction.

There exist multiple methods for calculating the distance between data points, such as the Chebyshev distance, Manhattan distance or the conventional Euclidean (L^2) distance. In attempt to create a more robust calculation, the distance function implemented in this technique calculated the absolute difference between the test and training data. By not squaring the distance, which occurs in the popular squared difference metric, large difference values do not outweigh small difference values. This method is coupled with mean-centered, *z*-standardized dataset, which is employed prior to the clustering technique.

A python function (*getNeighbors*) was created to implement this technique using five inputs: training data features, training data classifications, test data features, and test data classification, and integer value for *k* parameter. Identical to the previously discussed procedure, the function calculates the distance via absolute difference for each row in the test set to the training set and returns the classification for each row based on the *k* parameter provided. The details for the implementation, performance metrics and results are discussed later in this report.

1.2 Decision Tree Analysis

For classification or regression purposes, a hierarchical data structure can be created to essentially *divide-and-conquer* a dataset into its constituents, providing a nonparametric methodology that is a directly interpretable and competitively efficient given data to train, or build, the decision tree. This process is considered a nonparametric method as the structure is not fixed a priori nor any assumes regarding the class densities are made.
Decision trees recursively splits a sequence into a smaller number of steps based on a decision function, commonly referred to as a decision node. The function utilized for the decision node returns discrete classifications labels for the tree's branches; that is, given an input data value, a decision function is applied, and a specific branch is chosen based on the function's output. This process continues recursively until no more splits are possible, called a leaf node.
In this project, the task is to build a classification tree where each branch split in the decision tree is quantified by an impurity measure. That is, a split is considered pure if all instances when choosing the branch belong to the same classification and no more splits are necessary. The decision function employed to split the data can take several shapes, some with binary output (i.e., 0 or 1) and others discrete output, and the instances should be split to decrease the impurity if the node is impure. This project uses three impurity measures to make the split decisions: (1) entropy, (2) Gini index, and (3) Misclassification Error. These measures of class impurity are given by:

$$
\begin{aligned}
&Entropy: && E(N) = \sum_i -p_i \log_2(p_i) \; ; \\
&Gini\ index: && G(N) = 1 - \sum_i p_i^2 \; ; \text{and,} \\
&Misclassification\ Error: && H(N) = 1 - \max p_i \; ,
\end{aligned}
$$

where, p_i is the probability of class i. The measure of entropy attempts to maximize the mutual information in the decision tree. The Gini index is maximal if the classes are perfectly mixed, such as 0.5 for a binary, two-class problem.

These measures of impurity for a two-class (binary) problem can be expressed in the following equations, where $\phi(p, 1-p)$ is a non-negative function that measures the split impurity:

$$
\begin{aligned}
&Entropy: && \phi(p, 1-p) = -p\log_2(p) - (1-p)\log_2(1-p) ; \\
&Gini\ index: && \phi(p, 1-p) = 2p(1-p) \; ; \text{and,} \\
&Misclassification\ Error: && \phi(p, 1-p) = 1 - \max(p, 1-p) \; ,
\end{aligned}
$$

where, $p^1 = p$ and $p^2 = (1-p)$ represent the probability for the first and second class, respectively. From this definition, a value of 0.5 for p^1 results an identical value for p^2, thus a poor decision branch. Of note, the Entropy and Gini index impurity measures have a negative parabola shape with peaks at 0.5 (criterion on *y*-axis and p on *x*-axis), while the Misclassification error measure takes a linear or triangular shape with the peak at 0.5, thus is not a smooth function and can be difficult to optimize numerically. Essentially, the decision tree algorithm is implemented in two steps (i.e., training and prediction) with multiple sub-steps:

- **Training Step:**
  - Split data into two groups based on impurity metric on all features;
  - Repeat split on the created two datasets based on chosen impurity metric, resulting in four datasets;
  - Repeat process creating branches and leaf nodes until maximum decision tree depth is met;
  - Return branches and split values;
- **Prediction Step:**
  - Read and evaluate test data based on initial (root) split criteria;
  - Evaluate two datasets splits on following level's node split criteria;
  - Repeat process until a leaf node is reached; and,
  - Return predicted classification value for each row in test data.

In this project, a python class *decision_tree* is created that creates a decision function using the *decision_tree.train* function after initializing the object with the impurity measure (e.g., entropy, Gini index, or misclassification error) and the maximum decision tree depth. Once the decision tree is created via the **.train* function, the **.predict* function uses the decision tree created (python dictionary) from the **.train* function to split the test data using the created tree and returns the resulting classifications.

1.3 Dimensionality Reduction

After the initial implementation of the clustering and decision tree, a dimensionality reduction method is implemented to reduce the number of features required to perform the analysis. In an ideal scenario, feature selection and/or extraction should not be required as separate process, where the classification or regression should be employed in a manner that use and removes features, respectively, as necessary. This case can be true for low dimensionality data, but by removing the irrelevant data features from the analysis can potentially strengthen the subsequent analysis by:

1. reduce computation time due to decrease in size of matrices;
2. simpler models provide more robust results on smaller datasets;
3. increase understanding of the dataset by keeping the features that impact the data; and,
4. potential increase in graphical representation of the original data as the analysis can be visualized in a lower dimension.

To analyze the usefulness of each feature to decrease the dimensionality of the data, principal component analysis (PCA) is utilized. As an unsupervised method, PCA attempts to maximize the variance and uses the variance as the criteria to acquire the appropriate level of dimensions to use for subsequent data analysis. Essentially, PCA is an eigenvalue problem, where the eigenvectors, referred to as the principal components (PCs), provide, after the data is projected on to the eigenvalue, a more spread out or separated data. There are several methods to the PCs, but a conventional method solves for the eigenvectors of the covariance matrix from the original data. The resulting PCs are then sorted in a decreasing order from greatest to least in the amount of variance explained by each PC. That is, the eigenvector with the highest eigenvalue is the direction that the data is most spread out, which is ideal for clustering.
Rather than directly calculating the eigenvalues and eigenvectors for PCA, another method called singular value decomposition (SVD), returns the principal components and allows for the data is to be decomposed into eigenvectors and eigenvalues. To add context, let's say the data $\boldsymbol{X}$ is a $(N \text{ x } d)$rectangular matrix and can be represented as the product of three matrices: $\boldsymbol{V}, \boldsymbol{A},$ and $\boldsymbol{W}^{\mathrm{T}}$, as shown in the following equation.

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^{T}$$

The matrix $\boldsymbol{U}$ $(N \text{ x } N)$ contains the eigenvectors of $\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}$ in its columns, the matrix $\boldsymbol{S}$ $(N \text{ x } d)$ contains the singular values, or eigenvalues, along the diagonal, and the matrix $\boldsymbol{V}^{\mathrm{T}}$ $(d \text{ x } d)$ contains the eigenvectors of $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ in its columns. To transforms the data into the resulting PC space, the product of $\boldsymbol{U}$ and $\boldsymbol{S}$ is used. To prove this, point that PCA and SVD will result in equal solutions, a random $(5 \text{ x } 4)$ matrix was created, and two approaches were used: (1) Python's SVD function in SciPy linear algebra library (*scipy.linalg.svd)*, and (2) Python's SciKit-Learn's decomposition library for PCA (*sklearn.decomposition.PCA*).

$$\boldsymbol{X} = \begin{bmatrix} 59 & 64 & 71 & 83 \\ 75 & 80 & 55 & 97 \\ 94 & 64 & 1 & 111 \\ 44 & 36 & 74 & 98 \\ 12 & 85 & 40 & 156 \end{bmatrix}$$

Using the SVD function to decompose the above matrix $\boldsymbol{X}$ into the three matrices in the SVD equation and subsequently multiplying the $\boldsymbol{U}$ and $\boldsymbol{S}$ matrixes, the following matrix is obtained.

$$\boldsymbol{X_{SVD}} = \begin{bmatrix} -1.085 & -0.420 & 0.644 & 0.169 \\ -0.201 & 0.500 & 1.046 & -0.147 \\ 0.134 & 2.108 & -0.701 & 0.033 \\ -1.406 & -1.251 & -0.911 & -0.073 \\ 2.559 & -0.937 & -0.099 & 0.018 \end{bmatrix}$$

Then using the *sklearn* decomposition library for the PCA function on the original data matrix and subsequently transforming the results into the PC-space, the following matrix is obtained.

$$\boldsymbol{X_{PCA}} = \begin{bmatrix} -1.085 & -0.420 & 0.644 & 0.169 \\ -0.201 & 0.500 & 1.046 & -0.147 \\ 0.134 & 2.108 & -0.701 & 0.033 \\ -1.406 & -1.251 & -0.911 & -0.073 \\ 2.559 & -0.937 & -0.099 & 0.018 \end{bmatrix}$$

Comparing the two resulting matrices shows the matrices are identical. This provides confidence in the usefulness and correctness of using the singular values from the SVD method.
Upon calculating these values, the amount of variance explained by each PC can be used to calculate the appropriate of number PCs or dimensions to use for the k-Means portion. A common approach is to choose an amount of variance to be explained by the lower-dimensioned matrix, say 90%, or choose a parameter where the variance slope changes drastically, called the *Elbow Method*. Both approaches use a *Scree graph,* which plots the variance explained per each PC, or eigenvector, kept for analysis. The graph looks similar to an inverse function (i.e. $1/x$), where the first PC contains the highest variance explained and decreases as the number of PCs employed increases. This graph is created later in this report and used to choose the appropriate number of the PCs to utilize.

2 Data Exploration

This project uses a (699 *row* × 11 *column*) dataset from the state of Wisconsin with ten parameters and a binary classification feature for breast cancer. The data features with the reported value range for the Wisconsin breast cancer dataset are as follows:

	Data Features	**Values**
1.	*Sample Code Number*	ID no.
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

The data was rescaled to a mean-centered, *z*-standardized range for each feature by subtracting each feature by the feature mean value and subsequently dividing by the feature standard deviation.

Initial exploration data showed an issue in the *Bare Nuclei*, where unexpected non-integer values were observed. Further investigation showed string values containing a question mark symbol (i.e. str('?')). These 16 rows were dropped from the analysis, as 16 data rows accounts for approximately 2% of the total data and was thus assumed to have minute impact if dropped from the analysis. Additionally, the first data feature *Sample Code Number* was dropped from the analysis as the data values were unique for each row and provides no benefit for the analysis portion. To provide additional context to reader, Figure 2 shows the relationship between the remaining data features and breast cancer classification. Of note, the color on the Figure corresponds to the breast cancer classification (i.e. blue for benign and orange for malignant) in order to visually quantify trends and relationships within the parameters.

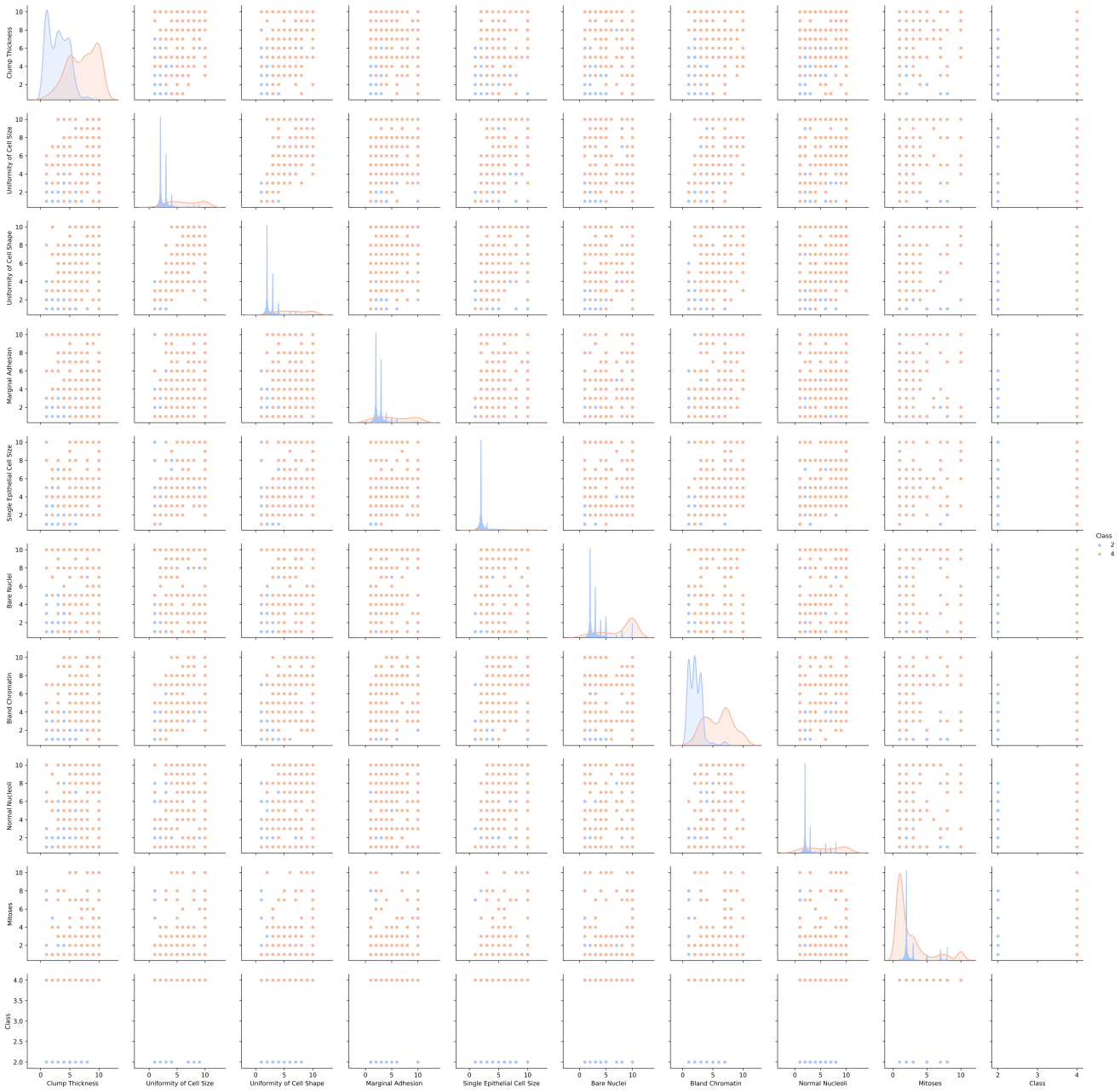

Figure 2: Full comparison plot of final data matrix utilized during the analysis with class coloring.

With the data mean-centered, *z*-standardized, and removed poor portions of the dataset, the remaining (683 *row* × 11 *column*) dataset is ready for the two discussed analysis techniques: *k*-NN clustering and Decision Tree analysis.

3 Data Analysis

This section highlights the two methods employed in this project, results, and performance metrics for the methods.

3.1 *k*-NN Clustering

The dataset was randomly split into four arrays (i.e. training data, training data classification, test data, and test data classification) via created python function (*test_split*) with 60% of the data in training set and the remaining 40% in the test set, resulting in datasets with 410 and 273 rows, respectively. The created *getNeighbors* function in python defaults to a *k*-value of 3 and returns the predicted classification parameters based on the provided *k*-value. The predicted classifications are compared to the correct, known classifications in the test data classification array via five performance metrics:

1. Accuracy,
2. True Positive Rate,
3. Positive Predictive Value,
4. True Negative Rate, and
5. F_1 Score.

All of these performance metrics use the comparison between the predicted classification values to the test set classification array by calculating the true positive (TP) and negative (TN) ratios, as well as the false positive (FP) and negative (FN) ratios. Using these four parameters, the previous five performance metrics are calculated via the following equations:

- Accuracy = $(TN + TP) / (TN + TP + FN + FP)$
- TPR = $TP / (TP + FN)$
- PPV = $TP / (TP + FP)$
- TNR = $TN / (TN + FP)$
- F_1 Score = $2 * PPV * TPR / (PPV + TPR)$

Lastly, another performance metric is created to compare the four-prediction rate calculated, referred to as a confusion matrix. For a two-class problem, such as the dataset for this project, the confusion matrix shows the relationship between the four-prediction rates as shown in the Table 1.

Table 1: Example confusion matrix for two-class problem.

	Predicted Values	
True Values	Class 1	Class 2
Class 1	TN	FP
Class 2	FN	TP

The five-performance metrics and the confusion matrix are created via two created python functions, *getMetrics* and *confusion_matrix*. Using a 60%-40% training-test split and the

default *k*=3 value with the *getNeighbors* function, the two performance metric functions discussed return the following values.

Table 2: Five performance metrics for 60%-40% random split using three nearest neighbors.

Performance Metrics	
Accuracy	0.9677
TPR	0.9569
PPV	0.9487
TNR	0.9733
F1-Score	0.9528

Table 3: Confusion matrix for 60%-40% random split using three nearest neighbors.

	Predicted Values	
True Values	*Benign*	*Malignant*
Benign	219	6
Malignant	5	111

Using a *k*-value of 3 with the 60%-40% provided a relatively high 96.77% accuracy. The number for the *k*-value is a hyperparameter that should reflect the best possible value that provides an adequate level of accuracy but does not overfit the data (bias-variance tradeoff). A method to choose an appropriate *k*-value is to plot the error rate as a function of the hyperparameter *k*. Using the created python functions, the error rate is plotted for using *k*-values ranging from 2 to 8 was calculated with the results provided in .

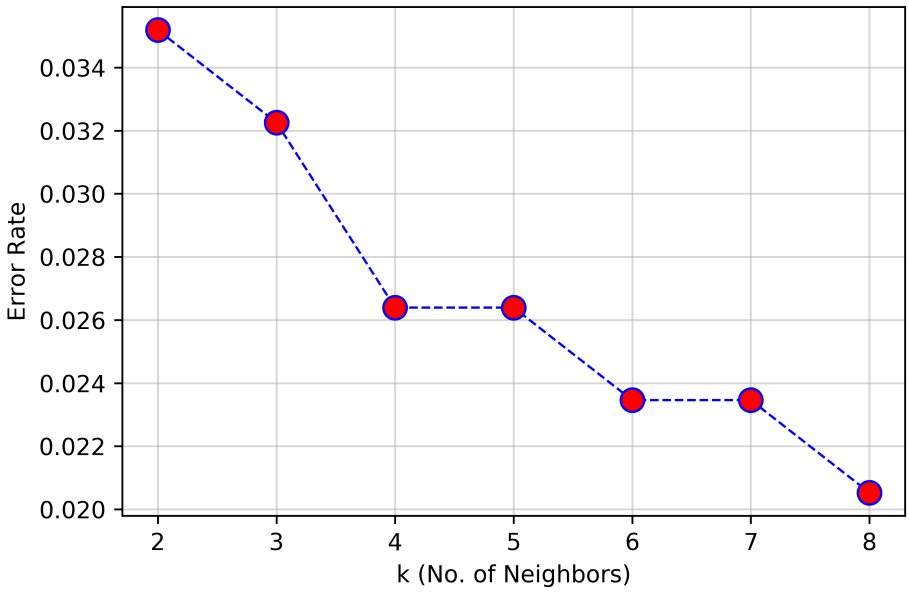

Figure 3: Error rate as a function of the hyperparameter *k*-value.

Using the *k*-value range from 2 to 8 as well as 17 and 33 nearest neighbors, the five-performance metrics were calculated, provided in Table 4, to assist in the hyperparameter selection. This table provides the average value after ten iterations. The average was taken with new training-test splits calculate every iteration to provide increased confidence in the *k*-NN cluster and performance metric calculations, where outliers may be present if only one iteration were utilized.

Table 4: *k*-NN average performance metrics for multiple *k*-values.

k-NN No.	Accuracy	TPR	PPV	TNR	F1 Score
2	0.965	0.949	0.949	0.973	0.949
3	0.959	0.923	0.956	0.978	0.939
4	0.971	0.957	0.957	0.978	0.957
5	0.959	0.923	0.956	0.978	0.939
6	0.974	0.974	0.950	0.973	0.962
7	0.968	0.957	0.949	0.973	0.953
8	0.977	0.983	0.950	0.974	0.966
17	0.965	0.949	0.949	0.973	0.949
33	0.965	0.949	0.949	0.973	0.949

From these values calculated and the previous Figure, a value of 8 is chosen as ideal for the *k*-value error rate is lowest, which introduces the smallest false positive and negative values, which results in the confusion matrix provided in the following Table.

Table 5: Confusion matrix for 60%-40% random split using eight nearest neighbors.

	Predicted Values	
True Values	*Benign*	*Malignant*
Benign	221	6
Malignant	2	112

3.2 Decision Tree Analysis

Using the same 60%-40% training-test split previously constructed for the *k*-NN analysis, the *decision_tree* class was used to create the decision tree and predict the classification from the test data. The decision tree returned the following confusion matrix and performance metrics for a tree of eight for the depth.

Table 6: Training set confusion matrix for 8 level decision tree using Gini index for impurity measure.

	Predicted Values	
True Values	*Benign*	*Malignant*
Benign	178	42
Malignant	3	119

Table 7: Test set confusion matrix for 8 level decision tree using Gini index for impurity measure.

	Predicted Values	
True Values	*Benign*	*Malignant*
Benign	191	33
Malignant	1	116

After further analysis of the created *decision_tree* class, an observation was made that the accuracy and classification predictions did not change for different levels of maximum depth past four. To compare results and validate the changes in the decision tree analysis with various tree depths, python's scikit-learns library was used with the three impurity measures previously discussed.

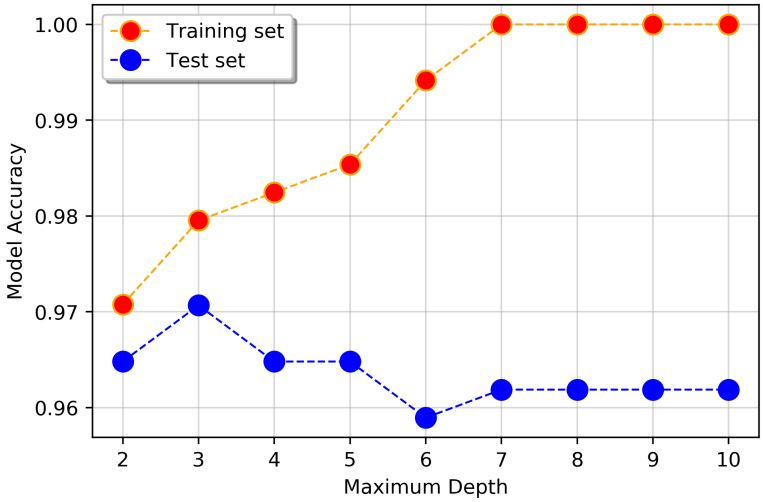

Figure 4: Decision Tree accuracy as a function of maximum tree depth using Gini index impurity measure.

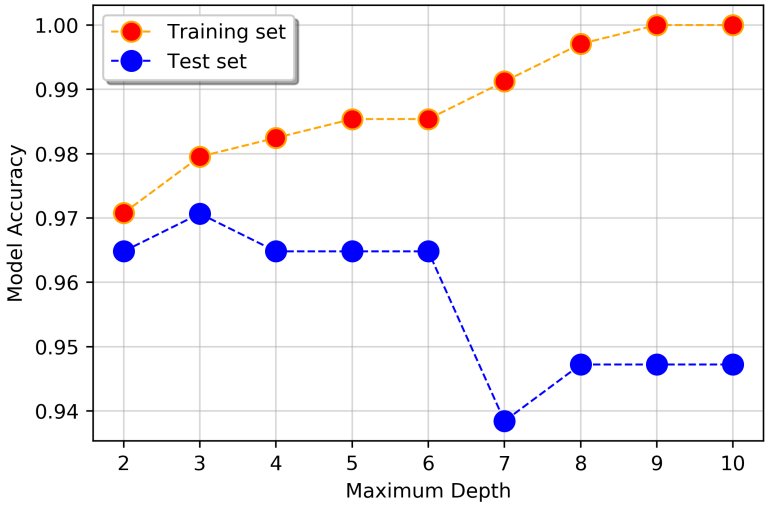

Figure 5: Decision Tree accuracy as a function of maximum tree depth using Entropy impurity measure.

From the model accuracy, it is relatively apparent that a maximum depth of greater than three overfits the data and decreases the model accuracy of the test set.

Table 8: Decision tree accuracy for a range of maximum depths using Gini index.

Dec. Tree Depth	Training Accuracy	Test Accuracy
2	0.971	0.965
3	0.980	0.971
4	0.982	0.965
5	0.985	0.965
6	0.994	0.959
7	1.0	0.962
8	1.0	0.962
9	1.0	0.962
10	1.0	0.962

Table 9: Decision tree accuracy for a range of maximum depths using Entropy.

Dec. Tree Depth	Training Accuracy	Test Accuracy
2	0.971	0.965
3	0.980	0.971
4	0.982	0.965
5	0.985	0.965
6	0.985	0.965
7	0.991	0.938
8	0.997	0.947
9	1.0	0.947
10	1.0	0.947

Using a maximum depth of three for the decision tree, the confusion matrix and the five-performance metrics can be calculated using the known and predicted classification values for the test data.

Table 10: Confusion matrix for decision tree using Gini index and maximum depth of three.

	Predicted Values	
True Values	*Benign*	*Malignant*
Benign	217	7
Malignant	3	114

Table 11: Performance metrics for decision tree using Gini index and maximum depth of three.

Max Depth	Accuracy	TPR	PPV	TNR	F1 Score
3	0.971	0.974	0.942	0.969	0.958

This represents 97.1% accuracy using a maximum depth of three for the decision tree using the Gini index. This is slightly lower than the previous *k*-NN method, which required eight nearest neighbors.

3.3 Dimensionality Reduction Analysis

To obtain a final data matrix with *n* number of PCs, a python function (*getValues*)was created that accepts a data matrix and the number of PCs requested by the use and returns a data set that is transformed into the PC space.

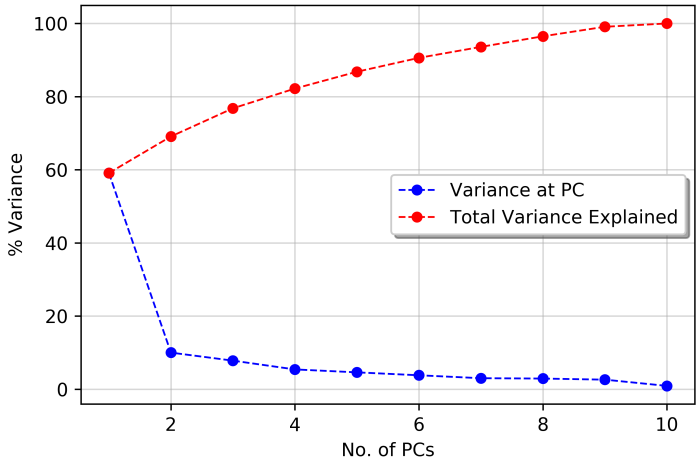

Figure 6: Illustration of variance explained by each principal component.

Table 12: Tabulated data for variance explained by each principal component.

P.C. No.	P.C. Variance	Total Variance Explained
1	59.1%	59.1%
2	10.0%	69.1%
3	7.8%	76.8%
4	5.4%	82.2%
5	4.6%	86.8%
6	3.8%	90.6%
7	3.0%	93.6%
8	2.9%	96.5%
9	2.6%	99.1%
10	0.9%	100.0%

Let's assume that a 90% total variance explained is an adequate representation of the original dataset and 10% variance unexplained is acceptable. Through this assumption, the data can now be reduced from ten features, or columns, to six features. Reducing the original data matrix to six features and performing a 60%-40% training-test split, the created *getNeighbors* and *decision_tree* python functions are used to classify the test data. The following tables correspond to the confusion matrices and performance metrics for each created function and the optimal hyperparameter chosen (e.g., *k*-value and maximum tree depth)

Table 13: Confusion matrix for *k*-NN using *k*-value of eight with new dataset.

	Predicted Values	
True Values	*Benign*	*Malignant*
Benign	173	5
Malignant	6	89

Table 14: Performance metrics for *k*-NN using new dataset.

***k*-Value**	**Accuracy**	**TPR**	**PPV**	**TNR**	**F1 Score**
8	0.945	0.989	0.867	0.923	0.924

Table 15: Confusion matrix for decision tree analysis using maximum depth of three with new dataset.

	Predicted Values	
True Values	*Benign*	*Malignant*
Benign	168	10
Malignant	0	95

Table 16: Performance metrics for decision tree using new dataset.

Max. Depth	**Accuracy**	**TPR**	**PPV**	**TNR**	**F1 Score**
3	0.959	0.934	0.947	0.972	0.942

To add context to these values, each model's accuracy can be compared to the accuracy for the lower dimensional data prediction. For the *k*-NN model, the original 8-nearest-neighbor model provided 97.7% accuracy using the full data set, which is 3.2% higher than the accuracy for lower dimensional data analysis of 94.5%. Regarding the decision tree analysis, the original 3-maximum-depth decision tree resulted in a predictive accuracy on the test set of 97.1%, where the accuracy using lower dimensional data found 95.9% accuracy. This show a 1.2% decrease in the accuracy of the decision tree predictive capability when using the lower dimensional dataset.

References

Alpaydin, Ethem. *Introduction to Machine Learning*. Third Edition. The MIT Press. Cambridge, Massachusetts. (2014). ISBN: 978-0-262-02818-9.