COSC 528: Project 3

Ian Lumsden

November 6, 2018

1 Objective

The goal of this project was to classify breast tumors as malignant or benign using the provided data. In the first part of the project, a k-nearest-neighbors algorithm was implemented and used to classify the data. In the second part, a decision tree algorithm was implemented using *entropy*, *Gini Index*, and *misclassification error* as the possible impurity measures. The decision tree was also used to classify the data. Then, the hyperparameters of both models were optimized using cross-validation. Finally, the dimensionality of the data was reduced using Principal Component Analysis (PCA), and the decision tree was applied to this reduced data.

2 Preprocessing

The provided data was contained in the "breast-cancer-wisconsin.data" file. It consists of 691 rows of the following 11 unlabeled features:

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

The first feature (Sample Code Number) was dropped as it provided no meaningful data to the classification. Additionally, there were several NaN values in the Bare Nuclei column. These were removed by replacing them with the mean of the other elements in the feature column. Then, the class labels were adjusted so that 2 became 0 and 4 became 1. Finally, roughly 30 percent of the dataset was not unique. Initially, all this data was included in the dataset. However, after all models were implemented and tested, it was determined that the models were slightly more accurate when all but the first instance of non-unique data was removed. Therefore, all the results listed are based on the fully unique dataset.

3 K-Nearest-Neighbors

The first algorithm for this project was k-nearest-neighbors. It is a classification algorithm that, unlike other algorithms, primarily uses logic to perform classification. This algorithm's "training" stage consists only of storing the training instances and labels. During classification, distances between the data element to be classified and each training element are calculated. Then, the k closest training elements are used to poll for the class of the new data.

The specific implementation of k-nearest-neighbors used for this project was mostly the same as the generic description. The distance measure used in this implementation was Euclidean distance, which is defined below:

$$d_i = \sqrt{\sum_{f=0}^{8} (y_f - x_f)^2} = ||\mathbf{y} - \mathbf{x}||, \tag{1}$$

where $\mathbf{y}$ is the data to be classified, $\mathbf{x}$ is the current training element, and f represents the features. The one primary difference between this project's implementation and the generic implementation was that the polling for the classification was weighted using the inverse of the distance. The polling score for a class was defined by

$$s^i = \sum_{d \in \delta^i} \frac{1}{d}, \tag{2}$$

where s^i represents the score for class i and δ^i is the subset of the calculated distances in which the training element used was of class i. The algorithm used is described by the pseudo-code in algorithm 1.

Algorithm 1 k-Nearest-Neighbors Pseudo-Code

```
$train\_data \Leftarrow$ Passed Training Data
$train\_labels \Leftarrow$ Passed Training Labels
$cdata \Leftarrow$ Array of Data to be Classified
$clabels \Leftarrow []$
for elem in cdata do
    $distances \Leftarrow []$
    for train_elem in train_data do
        Append result of $euclidean\_distance(elem, train\_elem)$ to distances
    end for
    Sort distances
    $ks \Leftarrow$ first k elements of distances
    $s0 \Leftarrow 0$
    $s1 \Leftarrow 0$
    for dist in ks do
        if training element corresponding to dist is class 0 then
            $s0 = s0 + \frac{1}{dist}$
        else
            $s1 = s1 + \frac{1}{dist}$
        end if
    end for
    if $s0 > s1$ then
        Add 0 to $clabels$ as class for $elem$
    else
        Add 1 to $clabels$ as class for $elem$
    end if
end for
Return $clabels$
```

Once the k-nearest-neighbors algorithm was implemented, cross-validation was used to determine the best value of k for the dataset. The following result statistics were calculated to determine the best value:

1. **Number of True Negatives (TN)**
2. **Number of False Negatives (FN)**
3. **Number of True Positives (TP)**
4. **Number of False Positives (FP)**
5. **Confusion Matrix**: Set of TN, FN, TP, and FP
6. **Accuracy**: $\frac{TN+TP}{TN+TP+FN+FP}$
7. **True Positive Rate (TPR)**: $\frac{TP}{TP+FN}$
8. **Positive Predictive Value (PPV)**: $\frac{TP}{TP+FP}$
9. **True Negative Rate (TNR)**: $\frac{TN}{TN+FP}$
10. F_1 **Score**: $\frac{2*PPV*TPR}{PPV+TPR}$

The statistics can be found in Appendix A. Based on the statistics, the k-nearest-neighbors algorithm performed the best with $k \geq 3$. Since the algorithm is naturally faster at lower values of k, the best k value for this dataset is 3.

4 Decision Tree

For the second part of the project, a decision tree algorithm was used to classify the dataset. The training for the decision tree algorithm consists of creating a tree in which the nodes are features and the traversal is controlled by inequalities applied to the features. The root node of each subtree is determined by finding the feature with the lowest impurity measure.

The decision tree implementation for this project was implemented so that it could use one of three different impurity measures: *entropy*, *Gini Index*, and *misclassification error*. For binary classification, the equation for *entropy* is

$$\phi(p) = -p\log_2(p) - (1-p)\log_2(1-p), \tag{3}$$

where p is the probability of class 0. The equation for *Gini Index* is

$$\phi(p) = 2p(1-p), \tag{4}$$

and the equation for *misclassification error* is

$$\phi(p) = 1 - \max(p, 1-p). \tag{5}$$

The implementation used for this project allows for the impurity measure, maximum tree depth, and impurity threshold to be set prior to training. During training, it constructs a binary decision tree where decision rules are less-than and greater-than-or-equal-to inequalities about the same splitting value. Data is classified by traversing the decision tree. The pseudo-code for the implementation for training is in algorithm 2.

After implementing the model, cross-validation was performed using the same statistics as for k-nearest-neighbors. For decision trees, these statistics were calculated for all combinations of impurity measures, maximum depths, and impurity thresholds. All statistics are not listed due to the amount of data. After analyzing the statistics, it was determined that the following three combinations produce the best results (all are the same):

1. Gini Index with maximum depth of 6 or 7 and impurity threshold of 0.0

2. Entropy with maximum depth of 6 or 7 and impurity threshold of 0.0, 0.1, or 0.2

3. Misclassification Error with maximum depth of 6 or 7 and impurity threshold of 0.0

Table 1 lists the statistics for all of these combinations.

Algorithm 2 Decision Tree Pseudo-Code

```
procedure getBestSplit($feature$)
    $best\_impurity \Leftarrow 2$
    $best\_split \Leftarrow$ -1
    for splitPoint in range(min, max, 0.5) do
        $p \Leftarrow \frac{NumberofClass0inSplit1}{NumberofElementsinSplit1}$
        $q \Leftarrow \frac{NumberofClass0inSplit2}{NumberofElementsinSplit2}$
        $imp1 \Leftarrow$ Impurity from $p$
        $imp2 \Leftarrow$ Impurity from $q$
        if $imp1 + imp2 < best\_impurity$ then
            $best\_impurity \Leftarrow imp1 + imp2$
            $best\_split \Leftarrow splitPoint$
        end if
    end for
    Return best_impurity and best_split
end procedure
procedure addNode($data$)
    $splits \Leftarrow$ []
    for f in features do
        Append $getBestSplit(f)$ to splits
    end for
    Sort splits by impurity
    if splits[0] can have non-leaf children (based on max depth and impurity
threshold) then
        Add splits[0] to tree
        $addNode$(data[data ¡ splitPoint])
        $addNode$(data[data $\geq$ splitPoint)
    else
        Add splits[0] to tree
        Add leaf (most probable class in split 1) as left child
        Add leaf (most probable class in split 2) as right child
    end if
end procedure
procedure train($train\_data$, $train\_labels$)
    $tdata \Leftarrow$ merge train_data and train_labels
    $addNode(tdata)$
end procedure
```

Statistics	Values
True Negative	113
False Negative	7
True Positive	54
False Positive	1
Accuracy	0.9542857142857143
True Positive Rate	0.8852459016393442
Positive Predictive Value	0.9818181818181818
True Negative Rate	0.9912280701754386
F_1 Score	0.9310344827586207

Table 1: Best Statistics for Decision Trees

5 Principal Component Analysis (PCA)

To try to improve the decision tree algorithm, Principal Component Analysis was applied to the data.

To begin, the preprocessed data, X, was factored using singular value decomposition (SVD). This process converts the data into the following

$$X = U\Sigma V^T, \tag{6}$$

where U is an mxm unitary matrix, Σ is a diagonal mxn matrix of eigenvalues, and V is a nxn unitary matrix of eigenvectors. The diagonal values of Σ^2 represent the variance that each principal component covers. By dividing by the sum of variances, the percentage of variance that each primary component covers was obtained. These percentages were plotted against the number of the principal component, as shown in Figs. 1 and 2. There are two graphs because the data was already split between training and testing sets, so the PCA was performed on each set individually.

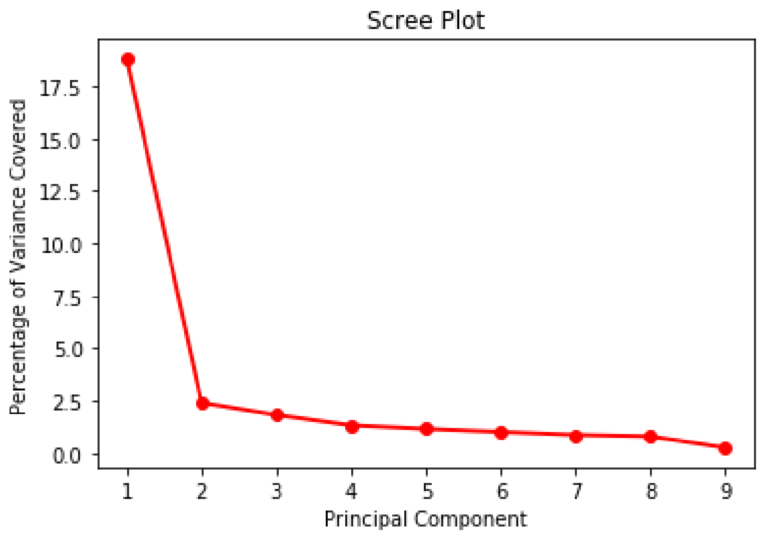

Figure 1: Scree Graph for SVD on the Training Set

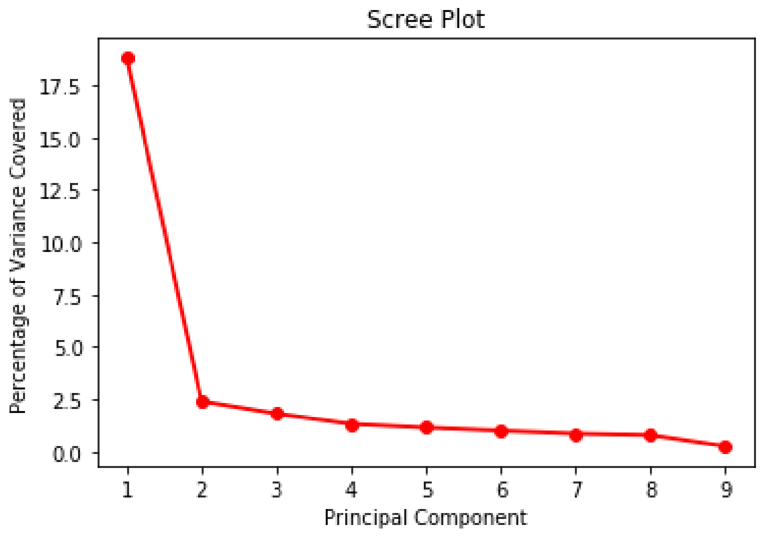

Figure 2: Scree Graph for SVD on the Testing Set

From Figs. 1 and 2, the first primary component clearly covered the most of the data's variance. However, none of the other features were clearly more important than the others. For visual purposes, the PCA was first performed to produce a two-dimensional reduced dataset. The equations for PCA are as follows

$$W = XV' = U'S', \tag{7}$$

where X is the initial dataset, the prime symbol indicates that a matrix has been reduced to its first k columns, and W is the final reduced dataset.

After implementing this equation using numpy, a graph (Figs. 3 and 4) was made of the two principal components, with one on each axis.

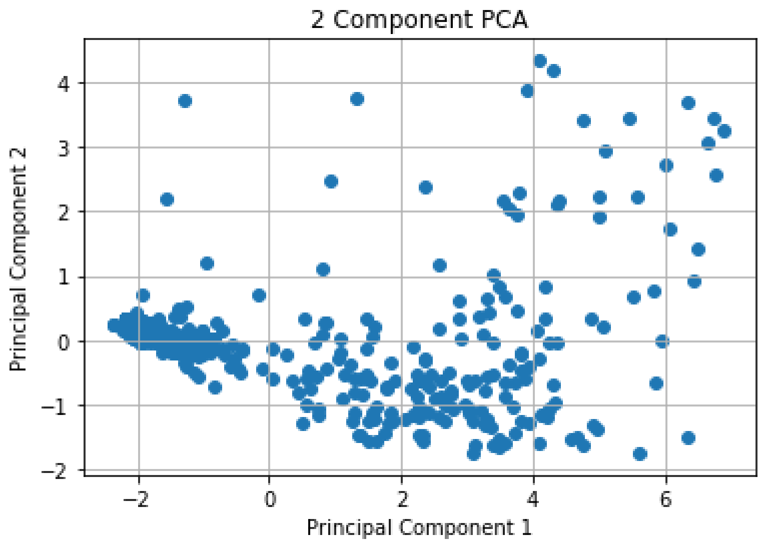

Figure 3: Relationship between the two primary components in the training data

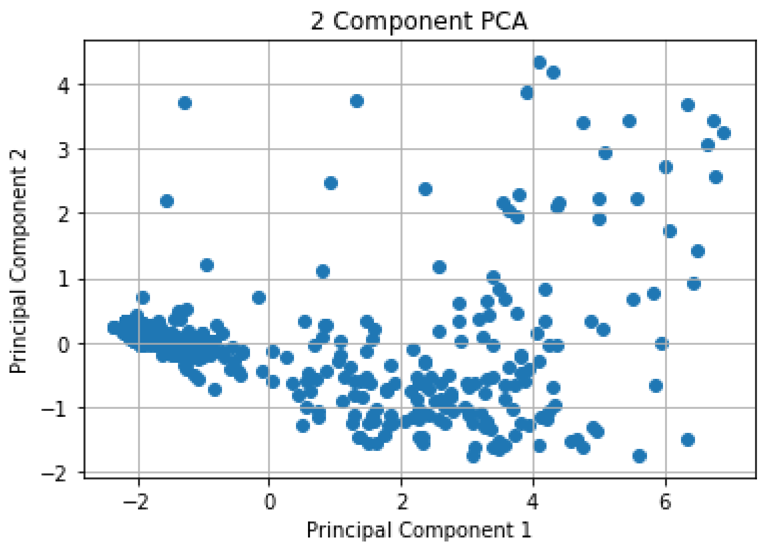

Figure 4: Relationship between the two primary components in the testing data

Finally, the decision tree cross-validation is repeated for PCA data obtained from values of k (number of reduced dimensions) between 2 and 9, inclusive. After analyzing the statistics, all values of k produced the same statistics: one set for *Gini Impurity* and *entropy* and another for *misclassification error*. The first set of statistics is in Table 2, and the second set is in Table 3.

Statistics	Values
True Negative	84
False Negative	38
True Positive	23
False Positive	30
Accuracy	0.6114285714285714
True Positive Rate	0.3770491803278688
Positive Predictive Value	0.4339622641509434
True Negative Rate	0.7368421052631579
F_1 Score	0.40350877192982454

Table 2: Statistics for *Gini Index* and *Entropy* for PCA data

Statistics	Values
True Negative	88
False Negative	40
True Positive	21
False Positive	26
Accuracy	0.6228571428571429
True Positive Rate	0.3442622950819672
Positive Predictive Value	0.44680851063829785
True Negative Rate	0.7719298245614035
F_1 Score	0.38888888888888884

Table 3: Statistics for *Misclassification Error* for PCA data

6 Conclusions

From the data above, both k-nearest-neighbors and decision trees are shown to be good classifiers for potentially cancerous breast tumors. Although both produce very similar results, this implementation of k-nearest-neighbors appears to be slightly more accurate than this implementation of a decision tree classifier. Both models could probably be improved if there were more unique elements in the dataset. For k-nearest-neighbors, the model could also possibly be improved by using a different distance measure, such as Mahalanobis or Chi Square distance. The decision tree model could probably be improved by using a different tree structure, such as a B-Tree. However, the high accuracy of the decision tree model on the unmodified data combined with the low accuracy of the model on the PCA data suggests that there is an issue in the PCA algorithm. The issue likely stems from either the normalizing of the data prior to applying PCA or the linear algebra used in implementing the PCA. However, given the high accuracy of the two models, a fully-functioning PCA would likely not produce significantly better results than the models on the raw data. As a result, applying PCA would likely not be needed.

7 Code

The following is a breakdown of the code files for this project:

- **driver.ipynb**: a Jupyter Notebook that is used as the "main" program
- **kNN.py**: the implementation of k-nearest-neighbors (as a class) and the model's supporting functions
- **decision_tree.py**: the implementation of the decision tree model (as a pair of classes) and the model's supporting functions
- **PCA.py**: the implementation of PCA and its supporting and visualization functions

8 Appendix A

1. $k = 2$
    - True Negative: 112
    - False Negative: 4
    - True Positive: 57
    - False Positive: 2
    - Accuracy: 0.9657142857142857
    - True Positive Rate: 0.9344262295081968
    - Positive Predictive Value: 0.9661016949152542
    - True Negative Rate: 0.9824561403508771
    - F_1 Score: 0.95
2. $k = 3$
    - True Negative: 112
    - False Negative: 3
    - True Positive: 58
    - False Positive: 2
    - Accuracy: 0.9714285714285714
    - True Positive Rate: 0.9508196721311475
    - Positive Predictive Value: 0.9666666666666667
    - True Negative Rate: 0.9824561403508771
    - F_1 Score: 0.9586776859504132
3. $k = 4$
    - True Negative: 112
    - False Negative: 3
    - True Positive: 58
    - False Positive: 2
    - Accuracy: 0.9714285714285714

- True Positive Rate: 0.9508196721311475
- Positive Predictive Value: 0.9666666666666667
- True Negative Rate: 0.9824561403508771
- F_1 Score: 0.9586776859504132

4. $k = 5$

   - True Negative: 112
   - False Negative: 3
   - True Positive: 58
   - False Positive: 2
   - Accuracy: 0.9714285714285714
   - True Positive Rate: 0.9508196721311475
   - Positive Predictive Value: 0.9666666666666667
   - True Negative Rate: 0.9824561403508771
   - F_1 Score: 0.9586776859504132

5. $k = 6$

   - True Negative: 112
   - False Negative: 3
   - True Positive: 58
   - False Positive: 2
   - Accuracy: 0.9714285714285714
   - True Positive Rate: 0.9508196721311475
   - Positive Predictive Value: 0.9666666666666667
   - True Negative Rate: 0.9824561403508771
   - F_1 Score: 0.9586776859504132

6. $k = 7$

   - True Negative: 112
   - False Negative: 3

- True Positive: 58
- False Positive: 2
- Accuracy: 0.9714285714285714
- True Positive Rate: 0.9508196721311475
- Positive Predictive Value: 0.9666666666666667
- True Negative Rate: 0.9824561403508771
- F_1 Score: 0.9586776859504132

7. $k = 8$
   - True Negative: 112
   - False Negative: 3
   - True Positive: 58
   - False Positive: 2
   - Accuracy: 0.9714285714285714
   - True Positive Rate: 0.9508196721311475
   - Positive Predictive Value: 0.9666666666666667
   - True Negative Rate: 0.9824561403508771
   - F_1 Score: 0.9586776859504132

8. $k = 17$
   - True Negative: 112
   - False Negative: 3
   - True Positive: 58
   - False Positive: 2
   - Accuracy: 0.9714285714285714
   - True Positive Rate: 0.9508196721311475
   - Positive Predictive Value: 0.9666666666666667
   - True Negative Rate: 0.9824561403508771
   - F_1 Score: 0.9586776859504132

9. $k = 33$

- True Negative: 112
- False Negative: 3
- True Positive: 58
- False Positive: 2
- Accuracy: 0.9714285714285714
- True Positive Rate: 0.9508196721311475
- Positive Predictive Value: 0.9666666666666667
- True Negative Rate: 0.9824561403508771
- F_1 Score: 0.9586776859504132