K-Nearest Neighbor and Decision Tree

Machine Learning: Project 3

Hossain, Rayhan (rhossai2)

Table of Contents

Introduction

In machine learning, decision making is a very common problem. For predicting the class of a given dataset we use different types of algorithm. KNN and Decision Tree are the widely used two algorithms for class prediction. KNN is the simplest one and easy to implement. The accuracy for both of them are good.

For our class project-3, this implements the KNN and Decision Tree algorithms without the support of sklearn or other machine learning tools and applies these for the breast cancer data to classify each data row as benign or malignant.

Dataset

The goal of this project was to apply the k-nearest-neighbors and decision trees to the diagnosis of breast cancer. It uses the given breast-cancer-wisconsin.data file. In the data there are following 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)

Data Preprocessing

The given dataset was almost clean except some missing values. For the simplicity, here all the missing values was replaced with zero. As the data file was in csv format, working directly with the file was super easy. The first column was id number, which had no effect on the results of our prediction algorithm. That's why this id column was removed from the original dataset.

```
1000025,5,1,1,1,2,1,3,1,1,2
1002945,5,4,4,5,7,10,3,2,1,2
1015425,3,1,1,1,2,2,3,1,1,2
1016277,6,8,8,1,3,4,3,7,1,2
1017023,4,1,1,3,2,1,3,1,1,2
1017122,8,10,10,8,7,10,9,7,1,4
```

Figure-1: raw data

K Nearest Neighbors

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.

A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.

Here, to measure the distance we use Euclidean distance. Formula for measuring the Euclidean distance is:

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Figure-2: Euclidean Distance

Decision Tree

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, decision tree algorithm can be used for solving regression and classification problems too.

Process to construct a decision tree:

1. Place the best attribute of the dataset at the root of the tree.
2. Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
3. Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.

To select, which data column should be the root node or the decision node, we take help of the information gain. To measure the information gain we use the impurity of data. From several ways, this project considers the following three to measure the randomness of data.

1. Entropy

$\phi(p, 1-p) = -p \log_2 p - (1-p) \log_2(1-p)$

2. *Gini index* (Breiman et al. 1984)

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

3. Misclassification error

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

Confusion and Performance Matrix

To evaluate the performance of our classification algorithm, we needed to compare the correct classification $\{r^t\}_{t=1}^N$ with our classifications $\{y^t\}_{t=1}^N$ (on the training, evaluation, or testing data). Therefore, implement a function that prints performance metrics determined as follows. Let TP = # true positives, TN = # true negatives, FP = # false positives, FN = # false negatives. We can print out a *confusion matrix* formatted as follows:

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

In addition to the confusion matrix, we also use following performance metrics:
- Accuracy = (TN + TP) / (TN + TP + FN + FP)
- TPR (true positive rate, recall, or sensitivity) = TP / (TP + FN)
- PPV (positive predictive value or precision) = TP / (TP + FP)
- TNR (true negative rate or specificity) = TN / (TN + FP)
- F1 Score = 2 × PPV × TPR / (PPV + TPR)

Implementation and Evaluation

Here we use Python 3.0 with Jupyter Notebook to implement the project. At first, we import the csv file and divided our whole dataset into training and test data set. After that, the KNN implementation process starts. Here in the project folder two Jupyter notebook files are added where we implement our KNN and Decision Tree.

The KNN needs to measure the distance to calculate the k nearest neighbors. The following function measures the Euclidean distance.

```python
import math

def euclidean_distance(data1, data2, length):
    distance = 0
    for x in range(length):
        v1 = data1[x]
        v2 = data2[x]
        if (v1 == '' or v1 == '?' or v1 == 'NA'):
            v1 = 0
        if (v2 == '' or v2 == '?' or v2 == 'NA'):
            v2 = 0
        distance += pow((float(v1) - float(v2)), 2)
    return math.sqrt(distance)
```

Figure-3: Function for Euclidean ditance

The main part of the KNN algorithm is to determine the k nearest neighbors of a given data point. The following function does this thing.

```python
import operator

def determine_class(neighbors):
    class_votes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in class_votes:
            class_votes[response] += 1
        else:
            class_votes[response] = 1
    sorted_votes = sorted(class_votes.items(), key = operator.itemgetter(1), reverse = True)
    return sorted_votes[0][0]
```

Figure-4: Function for finding the class

After that, running the KNN with different values of k produce different but almost nearby accuracy rate. Some of the results are stated below.

```
Running KNN with k= 2
Total Data:  699
Training Set 427
Test Set 272
TN:  184
FP:  4
FN:  5
TP:  83
KNN Accuracy: 96.73913043478261 %
True positive rate: 94.31818181818183 %
Positive predictve value: 95.40229885057471 %
True negative rate: 97.87234042553192 %
F1 Score: 94.85714285714285 %
```

```
Running KNN with k= 3
Total Data:  699
Training Set 388
Test Set 311
TN:  193
FP:  6
FN:  6
TP:  110
KNN Accuracy: 96.19047619047619 %
True positive rate: 94.82758620689656 %
Positive predictve value: 94.82758620689656 %
True negative rate: 96.98492462311557 %
F1 Score: 94.82758620689656 %
```

```
Running KNN with k= 4
Total Data:  699
Training Set 432
Test Set 267
TN:  163
FP:  5
FN:  10
TP:  93
KNN Accuracy: 94.4649446494465 %
True positive rate: 90.29126213592234 %
Positive predictve value: 94.89795918367348 %
True negative rate: 97.02380952380952 %
F1 Score: 92.53731343283582 %
```

```
Running KNN with k= 8
Total Data:  699
Training Set 421
Test Set 278
TN:  185
FP:  7
FN:  2
TP:  88
KNN Accuracy: 96.80851063829788 %
True positive rate: 97.77777777777777 %
Positive predictve value: 92.63157894736842 %
True negative rate: 96.35416666666666 %
F1 Score: 95.13513513513513 %
```

For different values of k, we plot the performance matrix values to determine the best one. K vs Accuracy, K vs TPR, K vs PPV, K vs F1 Score is shown below. It seems to me that the values k = 33 produces consistent results.

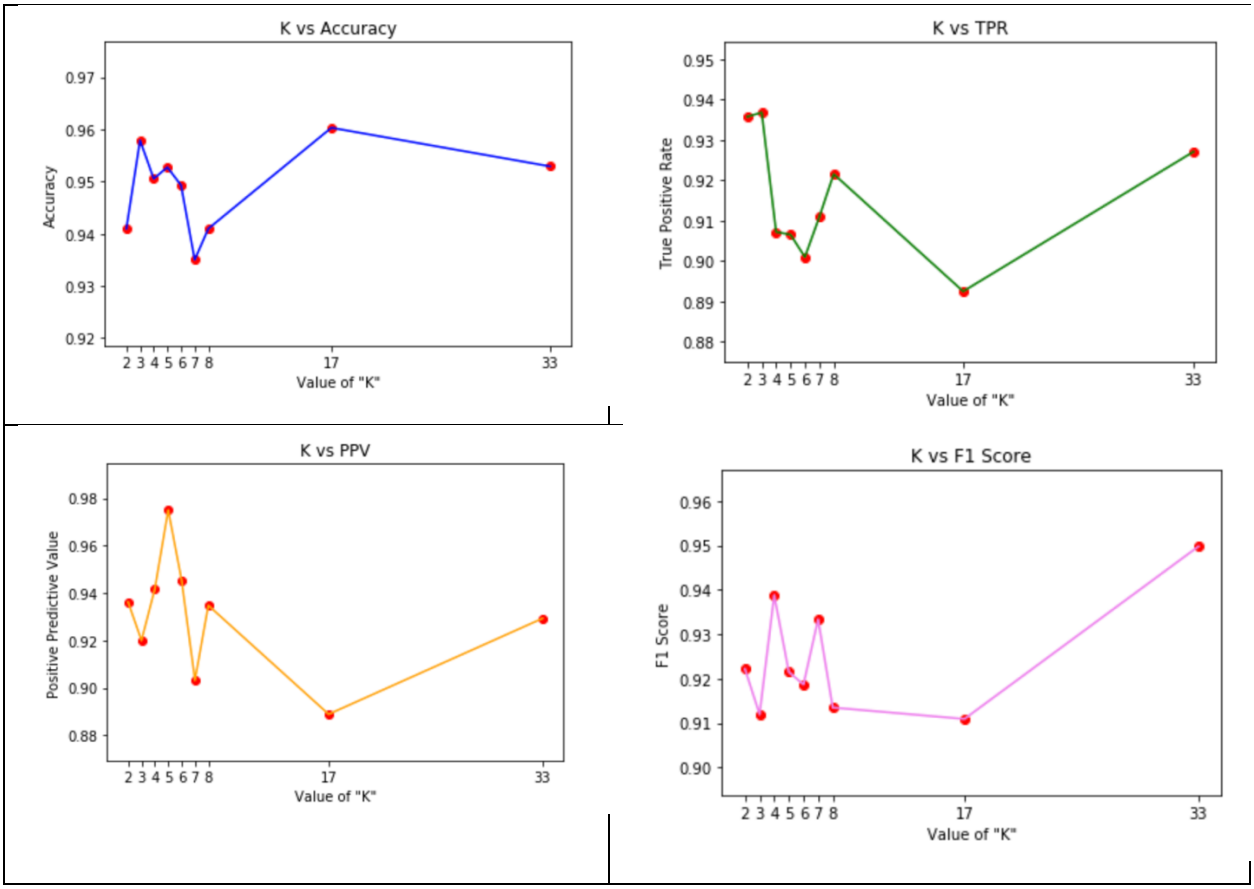

Figure-5: K vs Performance Matrix

For building the decision tree, one of the important steps is to calculate the impurity of dataset. As the requirements of the project, here it calculates in three different ways- Entropy, Gini, and Misclassification error.

```
def entropy(rows):

    if len(rows) == 0:
        return 0
    counts = class_counts(rows)
    for label in counts:
        p = counts[label] / float(len(rows))
        break

    if p is 0 or p is 1:
        return 0
    elif p is 0.5:
        return 1
    else:
        return -(1 * p * np.log2(p)) - ((1 - p) * np.log2((1 - p)))

```

Figure-6: Function for Entropy

```
def gini(rows):
    counts = class_counts(rows)
    for label in counts:
        p = counts[label] / float(len(rows))
        break

    if p is 0 or p is 1:
        return 0
    elif p is 0.5:
        return 1
    else:
        return 2.0 * p * (1-p)
```

Figure-7: Function for Gini

```
def misclassification_error(rows):
    counts = class_counts(rows)
    for label in counts:
        p = counts[label] / float(len(rows))
        break

    if p is 0 or p is 1:
        return 0
    elif p is 0.5:
        return 1
    else:
        return 1 - np.max([p, 1 - p])
```

Figure-8: Function for Miss classification Error

Below the impurity for different methods are given using our dataset:

```
Entropy: 0.9236293018025431
Gini: 0.4480047533660835
Misclassification Error: 0.3387622149837134
```

Calculating the randomness of data, we measure the information gain for the features and choose the best data node for splitting the data set. In the next step, it builds a decision tree after considering all the best splits and recursively repeating the process until getting the maximum depth or information gain = 0.

For taking the true/ false decision we consider the condition of value >= 5. As all of our values were in range 1 to 10, we chose as the easy condition to break.

One example decision tree built using our project and dataset is shown below. Here, the misclassification error impurity measurement is used.

Decision Tree Using Misclassification Error	Decision Tree using gini

Is Bare Nuclei >= 5?
--> Ture:
  Is Clump Thickness >= 5?
  --> Ture:
  Predict {'4': 73, '2': 2}
  --> False:
  Is Bland Chromatin >= 5?
  --> Ture:
  Is Normal Nucleoli >= 5?
  --> Ture:
  Predict {'4': 3}
  --> False:
  Is Uniformity of Cell Shape >= 5?
  --> Ture:
  Predict {'4': 4}
  --> False:
  Predict {'2': 1}
  --> False:
  Predict {'4': 4, '2': 1}
--> False:
  Is Uniformity of Cell Size >= 5?
  --> Ture:
  Is Mitoses >= 5?
  --> Ture:
  Predict {'4': 1}
  --> False:
  Is Clump Thickness >= 5?
  --> Ture:
  Predict {'4': 13, '2': 1}
  --> False:
  Predict {'4': 1}
  --> False:
  Is Bland Chromatin >= 5?
  --> Ture:
  Is Clump Thickness >= 5?
  --> Ture:
  Predict {'4': 4, '2': 1}
  --> False:
  Predict {'2': 1}
  --> False:
  Is Mitoses >= 5?
  --> Ture:
  Predict {'4': 1}
  --> False:
  Predict {'2': 196}
	Is Bare Nuclei >= 5?
	--> Ture:
	  Is Uniformity of Cell Shape >= 5?
	  --> Ture:
	  Is Bland Chromatin >= 5?
	  --> Ture:
	  Predict {'4': 47}
	  --> False:
	  Is Marginal Adhesion >= 5?
	  --> Ture:
	  Is Normal Nucleoli >= 5?
	  --> Ture:
	  Predict {'4': 5}
	  --> False:
	  Is Single Epithelial Cell Size >= 5?
	  --> Ture:
	  Predict {'2': 1}
	  --> False:
	  Predict {'4': 2}
	  --> False:
	  Predict {'4': 13}
	  --> False:
	  Is Clump Thickness >= 5?
	  --> Ture:
	  Is Bland Chromatin >= 5?
	  --> Ture:
	  Predict {'4': 12}
	  --> False:
	  Is Marginal Adhesion >= 5?
	  --> Ture:
	  Predict {'2': 1}
	  --> False:
	  Predict {'4': 3}
	  --> False:
	  Is Normal Nucleoli >= 5?
	  --> Ture:
	  Predict {'4': 1}
	  --> False:
	  Is Bland Chromatin >= 5?
	  --> Ture:
	  Predict {'2': 1}
	  --> False:
	  Predict {'4': 1, '2': 1}
	--> False:
	  Is Uniformity of Cell Size >= 5?
	  --> Ture:
	  Is Marginal Adhesion >= 5?
	  --> Ture:
	  Is Normal Nucleoli >= 5?
	  --> Ture:
	  Is Bland Chromatin >= 5?
	  --> Ture:
	  Predict {'4': 3, '2': 1}

Figure-9: Decision Tree using Miss Error and Gini

Impurity Measure	Accuracy	TPR	PPV	TNR	F1 Score
Gini	0.90384615	0.83720930	0.92307692	0.95081967	0.87804878
Miss Error	0.86538461	0.86046511	0.82222222	0.86885245	0.84090909
Entropy	0.93686868	0.88489208	0.93181818	0.96498054	0.90774907

Figure-10: Performance Matrix of Decision Tree

Dimensionality Reduction

I started with the PCA and Dimensionality reduction. After doing all the process we received data with two PCs. But I was confused the negative values of my eigen vectors. That's why I did the dimensionality reduction portion but did not use it with the Decision tree.

Conclusion

The KNN was easy to implement and interesting to analyze. For the decision tree, it took long time to implement with different impurity functions and parameters. But, one thing about the data property I was confused- how to break the data value conditions. Should I consider 1 to 10 discrete values or I can consider two classes: 1-5 or 6-10. For this project, I consider the second one.

Reference

[1] http://dataaspirant.com/2017/01/30/how-decision-tree-algorithm-works/
[2] https://en.wikipedia.org/wiki/Euclidean_distance
[3] https://www.saedsayad.com/k_nearest_neighbors.htm
[4] https://www.youtube.com/watch?v=qDcl-FRnwSU&t=2442s