CSC 240: Data Mining – Homework #4

Naïve Bayes and Decision Tree Classifiers

Sammy Potter
spott14@u.rochester.edu

Justin Lee
jlee363@u.rochester.edu

Abstract

The goal of this project was to predict the class of a given wine based on its attributes. On the given data set (which can be found here), we trained two machine learning models by following online resources using two different techniques: naïve bayesian classification and decision tree induction. We then compare and provide an analysis of the results of these two models.

Introduction

In this paper, we explore the training and performance of two machine learning models for predicting the class of wine based on its attributes. First, we describe the data set we use and the steps in pre-processing it. We then discuss the two different training techniques we used for the two models, naïve bayesian classification and decision tree induction, and the underlying concepts behind them. Afterwards, we discuss the actual steps we took in training and evaluating the two models. Finally, we compare the results of the two models and provide an analysis of their results via ROC/AUC and k-arry cross validation.

Methods

Pre-processing

The given data set describes the chemical analysis of wines grown in the same region in Italy but are created from different plants. In total, there are 3 different classes of wine, and each wine has 13 additional attributes describing different parts of its chemical composition. The given data set is a spreadsheet where each line represents a different wine. Each line has 14 columns in total. There are no missing/empty attributes and there are no nominal/ordinal attribute values, so our code does not account for them.

Note that in the interest of remaining concise, any code included in the paper does not include certain segments and lines present in our Google Colab. In any case, we import the following libraries to assist with pre-processing the data, training the models, and analysis of the results:

SEGMENT I: IMPORTING LIBRARIES

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import accuracy_score
from sklearn.naïve_bayes import GaussianNB
from sklearn.metrics import roc_auc_score
from sklearn import preprocessing
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
```

We then read in our data into a Pandas DataFrame:

SEGMENT II: DEFINING DATAFRAME

```
df = pd.read_csv("wine.data", header = None)

df.columns = [
'class', 'alcohol', 'malic acid', 'ash', 'alcalinity',  'magnesium', 'total phenols',         'flavanoids',
'nonflavanoid phenols', 'proanthocyanins', 'color intensity', 'hue',      '0D280-0D315', 'proline'
]
```

After reading in the data and converting it into a mutable data structure, we split the data into multiple sets to be used for either training or testing:

SEGMENT III: PARTITIONING DATA INTO TRAINING AND TESTING SETS

```
X = df.drop(['class'], axis=1)
y = df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

For each set, we use the RobustScaler from the Python library sklearn in order to scale all of our data:

SEGMENT IV: SCALING DATA

```
scaler = RobustScaler()
cols = X_train.columns
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])
```

These four sets that were processed from the original given data set will be used in training and testing both our naïve bayesian classifier and our decision tree classifier.

Naïve Bayesian Classifier

Naïve Bayesian classification is a simple algorithm built upon Bayes' Theorem that is used for classification tasks. Essentially, given information about an item's attributes, Bayes' Theorem is used to calculate the probability of and predict that item's other attributes.

In order to train and test our naïve Bayesian model, we run the following code (Prashant111, 2020). We create a confusion matrix to track performance:

SEGMENT V: TRAINING AND TESTING NAIVE BAYESIAN MODEL

```
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
y_pred
y_pred_train = gnb.predict(X_train)
y_pred_train

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

cm = confusion_matrix(y_test, y_pred)
```

Decision Tree Classifier

A decision tree is a tree-like model of decisions and their consequences that "split[s] the instances into two or more homogeneous sets based on [the] most significant splitter/differentiator in input values" (Gandhi, 2019). To train, test, and analyze our results, we write the following code:

SEGMENT VI:

```
def multiclass_roc_auc_score(y_test, y_pred, average="macro"):
        lb = preprocessing.LabelBinarizer()
        lb.fit(y_test)
        y_test = lb.transform(y_test)
        y_pred = lb.transform(y_pred)
        return roc_auc_score(y_test, y_pred, average=average)
y_pred_prob = gnb.predict_proba(X_test)[0:10]
gnb.predict_proba(X_test)[0:10, 1]
y_pred1 = gnb.predict_proba(X_test)[:, 2]
ROC_AUC = multiclass_roc_auc_score(y_test, y_pred, average="macro")
scores = cross_val_score(gnb, X_train, y_train, cv = 10, scoring='accuracy')
dtc = DecisionTreeClassifier(max_depth=5)
scores1 = cross_val_score(dtc, X_train, y_train, cv = 10, scoring='accuracy')
dtc.fit(X_train, y_train)
y_pred1 = dtc.predict(X_test)
ROC_AUC1 = multiclass_roc_auc_score(y_test, y_pred1, average="macro")
```

Results/Discussion

For our GaussianNB classifier, we received good performance metrics. Our GNB had a classification accuracy of 0.95 and an error of only 0.05. Our precision measured at 1.000, and recall/sensitivity at 0.9048. Our true-positive rate was 0.9048, our false-positive rate was zero, and our specificity was 1.000. These results initially show that the model is performing well. We calculated the ROC-AUC score for our model, receiving an average score of 0.96368 for the GNB. We calculated a set of scores for our model using k-fold cross-validation (we used 10-fold cross-validation). Our average cross-validation score was 0.9756.

We generated a confusion matrix for our results. We found an equal number of true positives and true negatives, and found zero false positives and only two false negatives. The confusion graph is as follows:

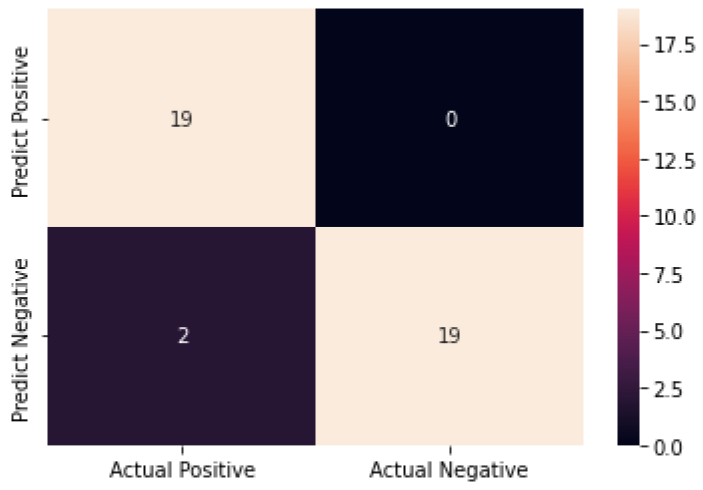

Figure 1: Generated confusion matrix

For our Decision Tree classifier, we also performed k-fold cross-validation (k=10), and received an average cross-validation score of 0.9032. We also calculated the ROC AUC score for our decision tree; we received 0.94806.

Our testing shows good performance for both of our classification methods.

Appendix

Code

The full code can be found here.

References

Aeberhard, S. (1991, July). *UCI Machine Learning Repository: Wine Data Set*. UCI Machine Learning Repository. Retrieved March 16, 2023, from https://archive.ics.uci.edu/ml/datasets/wine

Gandhi, V. (2019, November 6). *A Guide to Decision Trees for Beginners*. Kaggle. Retrieved March 19, 2023, from https://www.kaggle.com/code/vipulgandhi/a-guide-to-decision-trees-for-beginners

Prashant111. (2020, August 28). *Naive Bayes Classifier in Python*. Kaggle. Retrieved March 19, 2023, from https://www.kaggle.com/code/prashant111/naive-bayes-classifier-in-python