2020

Lab 9: Classification using Decision Trees & Naïve Bayes

Assoc. Prof. Dr. Mohammed Al-Sarem
Taibah University, Information System Department

3/26/2020

Lab 9: Classification using Decision Trees & Naïve Bayes

Lab Objectives:

The goal of this lab is to explain how to employ decision tree algorithm as well as Naïve Bayes as classification methods. After completion of this lab, you will be able to:

- Understand classification task
- Implement decision trees and naïve Bayes
- Evaluate classification model
- Visualize decision trees

Methodology

In order to complete the tasks of this tutorial, you have first be sure that the required libraries are installed. In this lab, we will work as usual with `pandas` and `sklearn`. Thus, we expected that both libraries are installed properly. We need also `Graphviz` library to visualize the generated decision trees. As well you installed the required libraries, be ensure that you can load iris data set and store it in pandas `dataframe`. Next, split dataset into training dataset and testing dataset. The training dataset is used to learn the classifier. Whereas the testing dataset is used to evaluate the goodness of the used classifier (`Naïve Bayes, Decision Tree`).

In class task:

At the end of this lab, the student will be able to:

- Deal with Dataset
- Implement Naïve Bayes and Decision Trees.
- Visualize the decision trees.
- Evaluate the classifiers.

home task:

Complete your **Course Project** (See Home task in lab 5).

References:

- https://graphviz.readthedocs.io/en/stable/manual.html
- https://scikit-learn.org/ stable/modules/generated/sklearn.model_selection.train_test_split.html
- Github:

Lab 9: Classification using Decision Trees & Naïve Bayes

1. Decision Trees

1.1 Installing the necessary libraries

Decision Trees are classification methods that are able to extract simple rules about the data features which are inferred from the input dataset. Several algorithms for decision tree induction are available in the literature. Scikit-learn contains the implementation of the `CART` (Classification and Regression Trees) induction algorithm.

- Import the required libraries. Since the `Graphvis` is not installed yet, write the following command in your `jupyter` cell and hit Enter.

```
pip install Graphviz
```

  (`Graphviz` library is needed to visualize the generated decision trees)

- Now, you are ready to import all the necessary libraries as follows:

```
import pandas as pd
import graphviz
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Setting random seed.
seed = 10
```

- In this lab, you need IRIS dataset. Thus, as usual, load the data using `read_csv()` method. Write code below!

```
In [4]: from sklearn import datasets
        iris = datasets.load_iris()
        df=pd.read_csv(r'D:\Programs\anaconda\lib\site-packages\sklearn\datasets\data\iris.csv', delimiter=',', header= 0,
                                        names=['sepal length(cm)',
                                               'sepal width (cm)',
                                               'petal length (cm)',
                                               'petal width (cm)',
                                               'Variety'])

        df.head()
```

Out[4]:

	sepal length(cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Variety
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

1.2 Dealing with dataset the necessary libraries

— Next, as we loaded the Iris dataset, extract its values and labels and split them into train and test sets.

```
# Creating a LabelEncoder and fitting it to the dataset labels.
le = LabelEncoder()
le.fit(df['Variety'].values)
# Converting dataset str labels to int labels.
y = le.transform(df['Variety'].values)
# Extracting the instances data.
X = df.drop('Variety', axis=1).values
# Splitting into train and test sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.34, stratify=y, random_state=seed)
```

Note: if your class lable contains only contenuous values, we don't need to write the following code:

```
le = LabelEncoder()
le.fit(df['Variety'].values)
# Converting dataset str labels to int labels.
 y = le.transform(df['Variety'].values)
```

You can move directly to line 4:

```
 X = df.drop('Variety', axis=1).values
# Splitting into train and test sets.
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.34, stratify=y, random_state=seed)
```

— Then, we will fit and test a `DecisionTreeClassifier`. Scikit-learn does not implement any post-prunning step. So, to avoid overfitting, we can control the tree size with the parameters `min_samples_leaf`, `min_samples_split` and `max_depth`.

```
# Creating a DecisionTreeClassifier.
# The criterion parameter indicates the measure used (possible values: 'gini' for the Gini index and
# 'entropy' for the information gain).
# The min_samples_leaf parameter indicates the minimum of objects required at a leaf node.
# The min_samples_split parameter indicates the minimum number of objects required to split an internal node.
# The max_depth parameter controls the maximum tree depth. Setting this parameter to None will grow the
# tree until all leaves are pure or until all leaves contain less than min_samples_split samples.
tree = DecisionTreeClassifier(criterion='gini',
                              min_samples_leaf=5,
                              min_samples_split=5,
                              max_depth=None,
                              random_state=seed)
tree.fit(X_train, y_train)
y_pred = tree.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('DecisionTreeClassifier accuracy score: {}'.format(accuracy))
```

Exercise 1.1: What is the accuracy of the model?
96.15384615384616

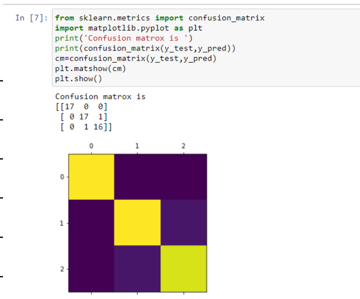
— We also can get `confusion matrix` of the model. To do so, write the code below:

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
```

```
  print('Confusion Matrix is')
  print(confusion_matrix(y_test, y_pred))
cm=confusion_matrix(y_test, y_pred)
plt.matshow(cm)
plt.show()
```

Exercise 1.2: Execute the code listed above and write down what did you note?

in this drawing, we can see the difference in colors



Exercise 1.3: Using the generated confusion matrix, compute the following:

Recall of class 0: 1

Precision of class 1: 0.94

Positive True of class 2: 0.94

Accuracy of all classes: 0.96

```
8]: from sklearn.metrics import classification_report
    print(classification_report(y_test, y_pred, labels=df['Variety'].unique()))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       0.94      0.94      0.94        18
           2       0.94      0.94      0.94        17

    accuracy                           0.96        52
   macro avg       0.96      0.96      0.96        52
weighted avg       0.96      0.96      0.96        52
```

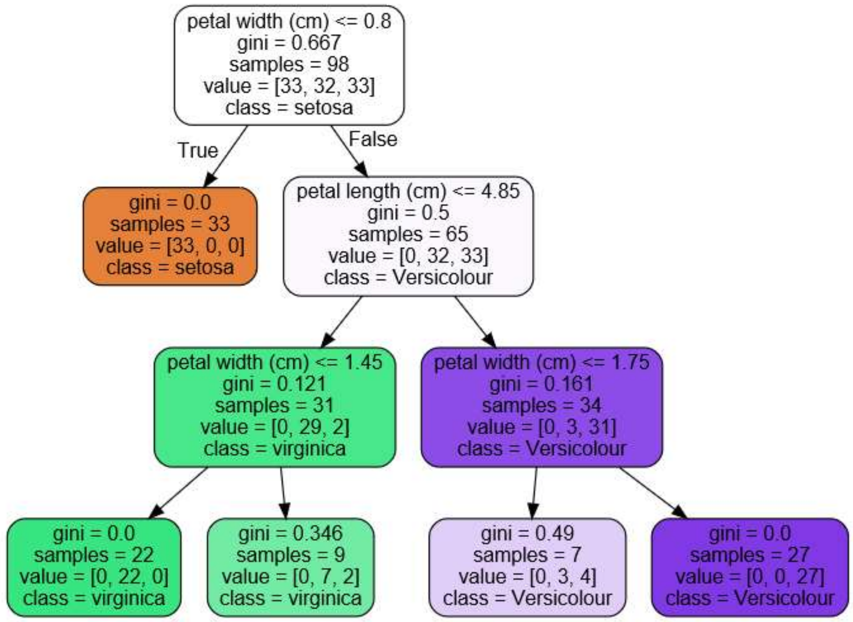
1.3 Visualizing Tree

— Finally, we can plot the obtained tree to visualize the rules extracted from the dataset.

```
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'


def plot_tree(tree, dataframe, label_col, label_encoder, plot_title):
    label_names = ['setosa','virginica','Versicolour']
    # Obtaining plot data.
    graph_data = export_graphviz(tree,
                                 feature_names=dataframe.drop(label_col, axis=1).columns,
                                 class_names=label_names,
                                 filled=True,
                                 rounded=True,
                                 out_file=None)
    # Generating plot.
    graph = graphviz.Source(graph_data)
    graph.render(plot_title)
    return graph
tree_graph = plot_tree(tree, df, 'Variety', le, 'Iris')
tree_graph
```

Here is the final decision tree



2. Naïve Bayes

Bayes' Theorem provides a way that we can calculate the probability of a piece of data belonging to a given class, given our prior knowledge. Bayes' Theorem is stated as:

$$P(Class|data) = \frac{P(data|Class) * P(Class)}{P(data)}$$

Where $P(Class|data)$is the probability of class given the provided data.

Naive Bayes is a classification algorithm for binary (two-class) and multiclass classification problems. It is called Naive Bayes because the calculations of the probabilities for each class are simplified to make their calculations tractable. Rather than attempting to calculate the probabilities of each attribute value, they are assumed to be `conditionally independent` given the class value.

This is a very strong assumption that is most unlikely in real data, i.e. that the attributes do not interact. Nevertheless, the approach performs surprisingly well on data where this assumption does not hold.

2.1 Importing the required libraries

To execute Naïve Bayes method, first you have to import the right algorithm. Assume your data contains only 2 class level at the target class which means that we are working with binary classification task. Thus, the preferable classifier type is `Bernoulli` classifier. You can also try `Gaussian` Naive Bayes model. Here is the code:

```
from sklearn.naive_bayes import GaussianNB, BernoulliNB
```

Lab 9: Classification using Decision Trees & Naïve Bayes

Suppose, we have the following data set as shown in the figure below:

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

For simplicity, we work with "`Outlook`", "`Temp`" features. Let the "`Play Golf`" is our target class.

Exercise 2.1: Load into dataframe only "`Outlook`", "`Temp`" as descriptive features and "`Play Golf`" as target class?

In [12]:
```
import pandas as pd
data={'weather':['Rainy','Rainy','Overcast','Sunny','Sunny','Sunny','Overcast','Rainy','Rainy','Sunny','Rainy','Overcast',
                 'Overcast','Sunny'],
      'temp':['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild'],
      'play':['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']}
data=pd.DataFrame(data)
data
```

Out[12]:

	weather	temp	play
0	Rainy	Hot	No
1	Rainy	Hot	No
2	Overcast	Hot	Yes
3	Sunny	Mild	Yes
4	Sunny	Cool	Yes
5	Sunny	Cool	No
6	Overcast	Cool	Yes
7	Rainy	Mild	No
8	Rainy	Cool	Yes
9	Sunny	Mild	Yes
10	Rainy	Mild	Yes
11	Overcast	Mild	Yes
12	Overcast	Hot	Yes
13	Sunny	Mild	No

2.2 Encoding Features

First, you need to convert these string labels into numbers. for example: 'Overcast', 'Rainy', 'Sunny' as 0, 1, 2. This is known as `label encoding`. Scikit-learn provides `LabelEncoder` library for encoding labels with a value between 0 and one less than the number of discrete classes (Refer to section 1.2).

```
# Converting string labels into numbers.
wheather_encoded=le.fit_transform(df['weather'])
print(wheather_encoded)
```

Exercise 2.2: Encode the remain data using LabelEncoder method()? Write the output here!

```
In [16]: weather_encoded=le.fit_transform(dataset['weather'])
         temp_encoded=le.fit_transform(dataset['Temp'])
         label=le.fit_transform(dataset['Play Golf'])
         print("weather :",weather_encoded)
         print("Temp :",temp_encoded)
         print("Play Golf :",label)

         weather : [1 1 0 2 2 2 0 1 1 2 1 0 0 2]
         Temp : [1 1 1 2 0 0 0 2 0 2 2 2 1 2]
         Play Golf : [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

Now combine both the features (weather and temp) in a single variable (list of tuples).

```
#Combinig weather and temp into single listof tuples
features=zip(wheather_encoded,temp_encoded)
features_ls= list(features)
print(features_ls)
```

2.3 Generating Naïve Bayes Model

Generate a model using naive Bayes classifier in the following steps:

- Create naive Bayes classifier
- Fit the dataset on classifier
- Perform prediction

Let we try first Bernoulli naïve Bayes classifier (Here, 1 indicates that players can 'play'.)

```
#Import Bernoulli Naive Bayes model
from sklearn.naive_bayes import GaussianNB, BernoulliNB

#Create a Bernoulli Classifier
model = BernoulliNB()

# Train the model using the training sets
model.fit(features_ls,label)

#Predict Output
predicted= model.predict([[0,2]]) # 0:Overcast, 2:Mild
print ("Predicted Value:", predicted)

Predicted Value: [1]
```

Exercise 2.3: Repeat the experiment again but now using Gaussian Naïve Bayes (just replace `model= GaussianNB()`)! Is the result the same? **Yes , the same result**

```
In [14]: model=BernoulliNB()
         model.fit(features_ls,label)

         predicted=model.predict([[1,0]])
         print("Predicted Value:",predicted)

         Predicted Value: [1]
```

```
In [15]: model = GaussianNB()
         model.fit(features_ls,label)

         predicted=model.predict([[1,0]])
         print("Predicted Value:",predicted)

         Predicted Value: [1]
```

Good luck