Lab 4

November 27, 2021

Lab 4: Supervised Learning - Naive Bayesian CLassifier

Classification in machine learning and statistics is a supervised learning approach in which the computer program learns from the data given to it and make new observations or classifications. Classification is a process of categorizing a given set of data into classes. The most common classification problems are – speech recognition, face detection, handwriting recognition, document classification, etc. It can be either a binary classification problem or a multi-class problem too. There are a bunch of machine learning algorithms for classification in machine learning. In this lab we will be looking at Naive Bayesian Classifier.

We will use a library called scikit-learn to build our model

[1]:
```
#import the numpy and scikit-learn library as follows
import numpy as np
import sklearn
```

We will be using built-in datasets of sklearn and sklearn comes with the following datasets

load_boston - Load and return the boston house-prices dataset (regression).

load_iris - Load and return the iris dataset (classification).

load_diabetes - Load and return the diabetes dataset (regression).

load_digits - Load and return the digits dataset (classification).

load_linnerud - Load and return the physical excercise linnerud dataset.

load_wine - Load and return the wine dataset (classification).

load_breast_cancer - Load and return the breast cancer wisconsin dataset (classification).

[2]:
```
#we begin by loading the iris dataset
from sklearn.datasets import load_iris
```

[3]:
```
data = load_iris()
```

The loaded data is in python dictionary format with the following keys.

target_names

target

feature_names

data

And, we will load the data into different variables as follows.

[4]:
```
label_names = data['target_names']
labels = data['target']
feature_names = data['feature_names']
features = data['data']
```

[5]:
```
print(label_names)
```

```
['setosa' 'versicolor' 'virginica']
```

[6]:
```
print(feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
```

[7]:
```
print(labels.shape)
```

```
(150,)
```

[8]:
```
print(features.shape)
```

```
(150, 4)
```

Now we will split the data into training set and test set. We train our classifier using training set data and we use the test set to see how well our model performs on the unseen data.

[9]:
```
from sklearn.model_selection import train_test_split
```

[10]:
```
X_train, X_test, y_train, y_test = train_test_split(features,labels,test_size =␣
↪0.330, random_state = 42)
```

We will be building our model. We are going to use Naïve Bayes algorithm for building the model.

In the case of categorical variables, such as counts or labels, a multinomial distribution can be used. If the variables are binary, such as yes/no or true/false, a binomial distribution can be used. If a variable is numerical, such as a measurement, often a Gaussian distribution is used.

Binary: Binomial distribution.

Categorical: Multinomial distribution.

Numeric: Gaussian distribution.

These three distributions are so common that the Naive Bayes implementation is often named after the distribution.For example:

Binomial Naive Bayes: Naive Bayes that uses a binomial distribution.

Multinomial Naive Bayes: Naive Bayes that uses a multinomial distribution.

Gaussian Naive Bayes: Naive Bayes that uses a Gaussian distribution.

[11]:
```
from sklearn.naive_bayes import GaussianNB
```

[12]:
```
gnb = GaussianNB()
```

[13]:
```
model = gnb.fit(X_train, y_train)
```

We are going to evaluate the model by making predictions on our test data. For making predictions, we will use the predict() function.

[14]:
```
preds = gnb.predict(X_test)
```

Now we check the prediction of our model with the actual output. Then we count the number of correctly classified instances and calculate the accuracy by dividing it by the size of our test data.

[15]:
```
result = y_test == preds
correct_preds = result.sum()
accuracy = correct_preds/y_test.size * 100
print("The accuracy is ",accuracy)
```

```
The accuracy is  96.0
```

We can also directly calculate the accuracy by using accuracy_score() from sklearn.metrics import accuracy_score

[16]:
```
from sklearn.metrics import accuracy_score
print("The accuracy is ",accuracy_score(y_test, preds)*100)
```

```
The accuracy is  96.0
```