Experiment-4

Yash Patel
2019130047
Batch C
TE Comps

Aim:
To train and test machine learning models using naive bayes algorithm.

Theory:
- The Bayes' Theorem is used to create a collection of classification algorithms known as Naive Bayes classifiers. It is a family of algorithms that share a similar idea, namely that each pair of features being classified is independent of the others.
- The Naive Bayes assumption is that each feature contributes equally and independently to the outcome.
- The Bayes' Theorem calculates the likelihood of an event occurring given the probability of a previous event.

Code:
```
# %%
import numpy as np
import pandas as pd

# %%
class NaiveBayesClassifier():
  # calculating prior probability
 def calc_prior_probability(self, features, target):
   self.prior = (features.groupby(target).apply(lambda x : len(x)) /
self.rows).to_numpy()
   return self.prior
  # calculating statistics
 def calc_statistics(self, features, target):
   self.mean = features.groupby(target).apply(np.mean).to_numpy()
   self.var = features.groupby(target).apply(np.var).to_numpy()
   return self.mean, self.var
  # naive bayes
 def gaussian_density(self, class_index, x):
   mean = self.mean[class_index]
   var = self.var[class_index]
```

```
    numerator = np.exp((-0.5) * ((x - mean) ** 2) / (2 * var))
    denominator = np.sqrt(2 * np.pi * var)
    probability = numerator / denominator
    return probability
   # calculating posterior probability
  def calc_posterior_probability(self, x):
    posteriors = []
    ## posterior probability for each class
    for i in range(self.count):
      prior = np.log(self.prior[i])
      conditional = np.sum(np.log(self.gaussian_density(i, x)))
      posterior = prior + conditional
      posteriors.append(posterior)
    return self.classes[np.argmax(posteriors)]  # classes with highest
posterior probability
  def fit(self, features, target):
    self.classes = np.unique(target)
    self.count = len(self.classes)
    self.features_numbers = features.shape[1]
    self.rows = features.shape[0]

    self.calc_statistics(features, target)
    self.calc_prior_probability(features, target)
  def predict(self, features):
    predictions = [self.calc_posterior_probability(x) for x in
features.to_numpy()]
    return predictions
  def accuracy(self, y_test, y_pred):
    accuracy = np.sum(y_pred == y_test) / len(y_test)
    return accuracy

# %%
# loading the dataset
data = pd.read_csv("Iris.csv")

# shuffling the dataset
```

```python
data = data.sample(frac=1, random_state=1).reset_index(drop=True)
data.drop("Id", axis="columns", inplace=True)

print(data.shape)

# setting the features and target
X, y = data.iloc[:, :-1], data.iloc[:, -1]

# splitting the dataset
X_train, y_train, X_test, y_test = X[:100], y[:100], X[100:], y[100:]

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

# %%
data

# %%
## Training the model
x = NaiveBayesClassifier()
x.fit(X_train, y_train)

# %%
x.classes, x.features_numbers, x.rows, x.count

# %%
print(x.calc_prior_probability(X_train, y_train))
x.prior

# %%
x.calc_statistics(X_train, y_train)

# %%
x.mean, x.var

# %%
```

```
X_train

# %%
predictions = x.predict(X_test)

# %%
y_test.value_counts(normalize=True)

# %%
x.accuracy(y_test, predictions)
```

Output:

```
Iris-setosa        0.38
Iris-versicolor    0.36
Iris-virginica     0.26
Name: Species, dtype: float64
```

```
Accuracy: 0.92
```

Conclusion:

- I learned about the basic Bayes theorem through the naive bayes experiment above. The likelihood of an event occurring in relation to any condition is described by Bayes' theorem. In the naive bayes method, we calculate the probability of each output category and choose the one with the highest probability.
- The naive bayes technique is based on two assumptions: each data point in the dataset adds to the dataset independently and equally.
- We can forecast the category with a fair accuracy of perhaps better than 90-95 percent using the naive bayes algorithm.