Assignment On

“Classification Methods in Python”

Course Title: Data Mining and Data Warehousing Lab

Course Code: CSEL-4108

Submitted To,

Dr. Md. Manowarul Islam
Associate Professor
Department of CSE
Jagannath University, Dhaka

Submitted By,

Momtaj Hossain Mow
(B180305025)
&
Md. Abu Yousuf Siam
(B180305012)

Department of Computer Science & Engineering
Jagannath University, Dhaka
Date of Submission: 16/06/2023

Classification Methods in Python

- **What is Classification?**

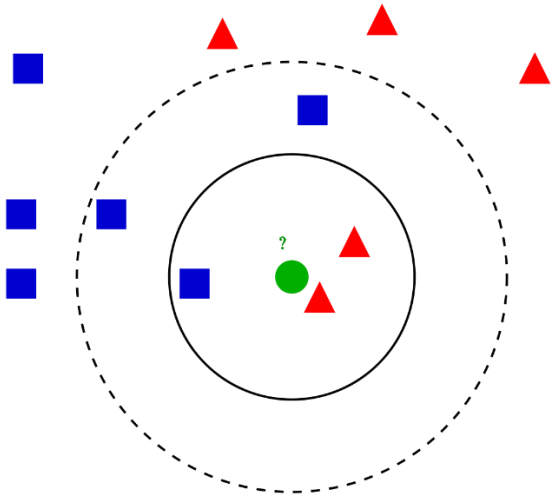
Classification is type of supervised learning. Supervised learning means that the data fed to the network is already labeled, with the important features/attributes already separated into distinct categories beforehand.

Here, the dataset is split up into training and testing sets, a set trains the classifier and a set the classifier has never seen before. The process of training a model is the process of feeding data into a neural network and letting it learn the patterns of the data. The testing process is where the patterns that the network has learned are tested. The features are given to the network, and the network must predict the labels.

Different types of classifiers are:

1) K-nearest Neighbors
2) Support Vector Machine
3) Decision Tree Classifiers
4) Naïve Bayes
5) Linear Discriminant Analysis
6) Logistic Regression

K-nearest Neighbors



K-nearest Neighbors operates by checking the distance from some test examples to the known values of some training example. The group of data points/class that would give the smallest distance between the training points and the testing point is the class that is selected.

Decision Trees

A Decision Tree Classifier functions by breaking down a dataset into smaller and smaller subsets based on different criteria. Different sorting criteria will be used to divide the dataset, with the number of examples getting smaller with every division. Once the network has divided the data down to one example, the example will be put into a class that corresponds to a key.

Naive Bayes

A Naive Bayes Classifier determines the probability that an example belongs to some class, calculating the probability that an event will occur given that some input event has occurred.

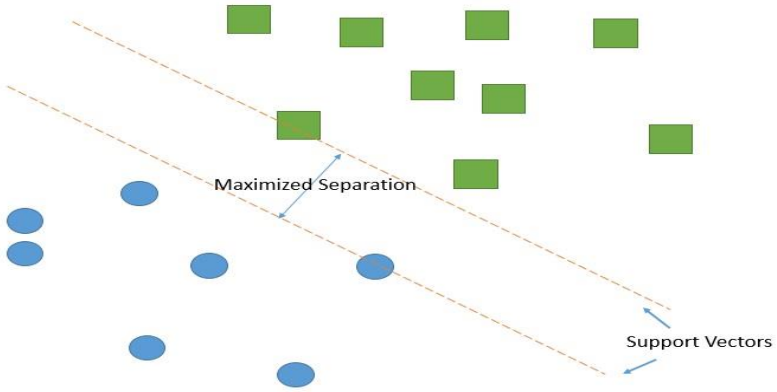
When it does this calculation, it is assumed that all the predictors of a class have the same effect on the outcome, that the predictors are independent

Linear Discriminant Analysis

Linear Discriminant Analysis works by reducing the dimensionality of the dataset, projecting all of the data points onto a line. Then it combines these points into classes based on their distance from a chosen point or centroid.

Linear discriminant analysis is a linear classification algorithm and best used when the data has a linear relationship.

Support Vector Machine



Support Vector Machines work by drawing a line between the different clusters of data points to group them into classes. Points on one side of the line will be one class and points on the other side belong to another class.
The classifier will try to maximize the distance between the line it draws and the points on either side of it, to increase its confidence in which points belong to which class. When the testing points are plotted, the side of the line they fall on is the class they are put in.

Logistic Regression

Logistic Regression outputs predictions about test data points on a binary scale, zero or one. If the value of something is 0.5 or above, it is classified as belonging to class 1, while below 0.5 if is classified as belonging to 0.

Each of the features also has a label of only 0 or 1. Logistic regression is a linear classifier and therefore used when there is some sort of linear relationship between the data.

Implementing a Classifier:

The first step in implementing a classifier is to import the classifier we need into Python. Here are some import statements for the classifiers we discussed before:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
```

After this, the classifier must be instantiated. Instantiation is the process of bringing the classifier into existence within the Python program - to create an instance of the classifier/object.

This is typically done just by making a variable and calling the function associated with the classifier:

```
knn_clf = KNeighborsClassifier()
dtree_clf = DecisionTreeClassifier()
nbayes_clf = GaussianNB()
svc_clf = SVC()
linearDC_clf = LinearDiscriminantAnalysis()
logreg_clf = LogisticRegression()
```

The training features and the training labels are passed into the classifier with the fit command:

```
logreg_clf.fit(features, labels)
```

After the classifier model has been trained on the training data, it can make predictions on the testing data.
This is easily done by calling the predict command on the classifier and providing it with the parameters it needs to make predictions about, which are the features in our testing dataset:

```
logreg_clf.predict(test_features)
```

These steps: instantiation, fitting/training, and predicting are the basic workflow for classifiers in Scikit-Learn.

The Machine Learning Pipeline

The machine learning pipeline has the following steps:

1) Preparing data
2) Creating training and testing sets
3) Instantiating the classifier
4) Training the classifier
5) Making predictions
6) Evaluating performance, tweaking parameters.

Let's look at an example of the machine learning pipeline, going from data handling to evaluation.

Import Necessary libraries

```
# Begin by importing all necessary libraries
import pandas as pd
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
```

[66] ✓ 0.0s

Load the Dataset

The Pandas library has an easy way to load in data, read_csv():

```
#read dataset
data = pd.read_csv('iris.csv')
# It is a good idea to check and make sure the data is loaded as expected.
print(data.head(5))
```

[65] ✓ 0.0s

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Data Preprocessing

The "ID" column can be dropped, as it is just a representation of row. As this isn't helpful we could drop it from the dataset using the drop() function:

```
data.drop('Id', axis=1, inplace=True)
```

✓ 0.0s

We now need to define the features and labels. We can do this easily with Pandas by slicing the data table and choosing certain rows/columns with iloc():

```
#Pandas ".iloc" expects row_indexer, column_indexer
X = data.iloc[:,:-1].values
# Now let's tell the dataframe which column we want for the target/labels.
y = data['Species']
```

[58] ✓ 0.0s

The slicing notation above selects every row and every column except the last column (which is our label, the species).

Creating Training and Test Sets

Now that we have the features and labels we want, we can split the data into training and testing sets using sklearn's handy feature train_test_split():

```
# Test size specifies how much of the data we want to set aside for the testing set.
# Random_state parameter is a random seed we can use.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=27)
```

[59] ✓ 0.0s

Loading...

We can print the results to be sure our data is being parsed as we expected:

```
print(X_train)
print(y_train)
```

✓ 0.0s

Output:

```
[[6.3 2.7 4.9 1.8]
 [5.  3.5 1.3 0.3]
 [6.4 2.7 5.3 1.9]
 [6.2 2.9 4.3 1.3]
 [6.7 3.1 4.7 1.5]
 [4.4 3.  1.3 0.2]
 [6.8 2.8 4.8 1.4]
 [6.3 2.5 5.  1.9]
 [5.8 2.7 3.9 1.2]
 [4.8 3.1 1.6 0.2]
 [4.6 3.4 1.4 0.3]
 [6.3 2.9 5.6 1.8]
 [5.9 3.2 4.8 1.8]
 [7.2 3.2 6.  1.8]
 [4.6 3.6 1.  0.2]
 [6.  2.9 4.5 1.5]
 [6.8 3.  5.5 2.1]
 [7.4 2.8 6.1 1.9]
 [6.1 2.9 4.7 1.4]
```

Instantiating the classifier

Now we can instantiate the models. Let's try using four classifiers:

1) K-Nearest Neighbors Classifier
2) Decision Tree Classifier
3) Naïve Bayes
4) Support Vector Classifier

```
knn_clf = KNeighborsClassifier(n_neighbors=5)
dtree_clf = DecisionTreeClassifier()
nbayes_clf = GaussianNB()
svc_clf = SVC()
```

[80] ✓ 0.0s

Training the classifiers:

```
knn_clf.fit(X_train, y_train)
dtree_clf.fit(X_train, y_train)
nbayes_clf.fit(X_train, y_train)
svc_clf.fit(X_train, y_train)
```

[69] ✓ 0.0s

Making Predictions:

The call has trained the model, so now we can predict and store the prediction in a variable:

```
Dtree_prediction = dtree_clf.predict(X_test)
KNN_prediction = knn_clf.predict(X_test)
NB_prediction = nbayes_clf.predict(X_test)
SVC_prediction = svc_clf.predict(X_test)
```

[70] ✓ 0.0s

Evaluating Performance:

We should now evaluate how the classifier performed. There are multiple methods of evaluating a classifier's performance, Accuracy score is the simplest way to evaluate.

```
# Accuracy score is the simplest way to evaluate
print('Accuracy score of Decision tree classifier: ',accuracy_score(Dtree_prediction, y_test))
print('Accuracy score of Naive Bayes classifier: ',accuracy_score(NB_prediction, y_test))
print('Accuracy score of KNN classifier: ',accuracy_score(KNN_prediction, y_test))
print('Accuracy score of SVC classifier: ',accuracy_score(SVC_prediction, y_test))
```

[77] ✓ 0.0s

Output:

```
Accuracy score of Decision tree classifier:  0.9
Accuracy score of Naive Bayes classifier:  0.9
Accuracy score of KNN classifier:  0.9666666666666667
Accuracy score of SVC classifier:  0.9333333333333333
```

But Confusion Matrix and Classification Report give more details about performance, Let's see an example for Naïve bayes classification:

```
#Confusion Matrix and Classification Report give more details about performance
print('Confusion Matrix of Naive Bayes Classification: ')
print(confusion_matrix(NB_prediction, y_test))

print('\nClassification report of Naive Bayes Classification: ')
print(classification_report(NB_prediction, y_test))
```

[83] ✓ 0.0s

Output:

```
Confusion Matrix of Naive Bayes Classification:
[[ 7  0  0]
 [ 0 10  2]
 [ 0  1 10]]

Classification report of Naive Bayes Classification:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.91      0.83      0.87        12
 Iris-virginica       0.83      0.91      0.87        11

       accuracy                           0.90        30
      macro avg       0.91      0.91      0.91        30
   weighted avg       0.90      0.90      0.90        30
```

The classification of data helps to categorize the huge amount of data to target categories. This enables us to identify areas with potential risks or profit by providing a better insight into the data.

References

[1] Overview of Classification Methods in Python with Scikit-Learn [Link: https://stackabuse.com/overview-of-classification-methods-in-python-with-scikit-learn/]

[2] Solving a Simple Classification Problem Using Python [Link: https://towardsdatascience.com/solving-a-simple-classification-problem-with-python-fruits-lovers-edition-d20ab6b071d2]

[3] Classification in Data Mining Explained: Types, Classifiers & Applications [Link: https://www.upgrad.com/blog/classification-in-data-mining/]