Apply Machine Learning Classification Models to Iris Flowers Dataset

Abdulaziz M. Alqumayzi

(DS-540) – Intro to python for computer science and data science: Learning to program with AI, Big Data and The Cloud

Colorado State University – Global Campus

Dr. Ernest Bonat

December 2, 2020

Apply Machine Learning Classification Models to Iris Flowers Dataset

Introduction

In this critical thinking four activity, we are going to write a program to implement Machine Learning classification models to Iris flowers dataset.

Discussion

iris.csv file download.

Download iris.csv file (https:/gist.github.com/netj/8836201) from the following connection. The label (target) from this file is specified by the column 'variety' and features 'epal.length,' 'sepal.width' and 'petal.length' as the column 'petal.width.'. In the figure 1, code that import Pandas library and read iris dataset into dataframe called df.

Figure 1

import Pandas library and read iris dataset code

```
import pandas as pd
df = pd.read_csv('iris.csv')
```

Preprocess the Iris.csv file with the target "variety" column encoding label.

In Figure 2 we want to predict "variety" with columns divided by X for functions and y for the variable using iloc() and values() for columns. The next step is the separation of the X and y into 80/20 sizes for training and tests. After that, standardize the attributes by eliminating the mean and scaling the unit variance. The features in X are processed with StandardScaler() and transform() functions for standardization are converted by centering and scaling on each attribute separately, measuring the appropriate sample figures for the training set.

Figure 2

separating features and target, splitting and standardize code

```
# features stored in X and variety in y
X = df.iloc[:,:-1].values
y = df.iloc[:,4].values
# splitting X and y to train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=11)
# Standardize features by removing the mean and scaling to unit variance
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Applying K Nearest Neighbors and Random Forests Machine Learning classification models.

K Nearest Neighbors (KNN) was computed using sklearn library with KNeighborsClassifier() function in figure 3. The KNN classifier was fitted on the X and y train then the X test predicted using predict() function. Also, the Random Forest applied as shown in figure 4. The Random Forest was fitted on both trains sets and the X train was predicted.

Figure 3

apply KNN machine learning code

```
# Applying K Nearest Neighbors Machine Learning classification model
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train,y_train)
y_pred_knn = knn.predict(X_test)
print('KNN prediction for the X test:\n')
print(y_pred_knn,'\n')
```

Figure 4

apply Random Forest machine learning code

```
# Applying Random Forests Machine Learning classification models
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train,y_train)
y_pred_rf = rf.predict(X_test)
print('Random Forest prediction for the X test:\n')
print(y_pred_rf,'\n')
```

To validate the model, measure the following evaluation metrics: accuracy score, confusion matrix and classification report.

the classification metrics Confusion Matrix, Classification Report, and Accuracy Score in figure 5 were imported from sklearn library and used to validate both machine learning models. In figure 5 the code shows the metrics applied to KNN model to validate the y test with y predicted. Also, applied to the random forest model as it shown in figure 6.

Figure 5

apply metrics on KNN code

```
# classification metrics for KNN
from sklearn.metrics import confusion_matrix, classification_report , accuracy_score
print("K Nearest Neighbors:")
print("Classification Report:\n")
print(classification_report(y_test,y_pred_knn),'\n')
print("Classification Confusion Matrix:\n")
print(confusion_matrix(y_test,y_pred_knn),'\n')
print("Classification Accuracy Score:\n")
print(accuracy_score(y_test, y_pred_knn))
```

Figure 6

apply metrics on Random Forest code

```
# classification metrics Random Forest
print("Random Forest:")
print("Classification Report:\n")
print(classification_report(y_test,y_pred_rf),'\n')
print("Classification Confusion Matrix:\n")
print(confusion_matrix(y_test,y_pred_rf),'\n')
print("Classification Accuracy Score:\n")
print(accuracy_score(y_test, y_pred_rf))
```

Comparing the two classification models.

In table 1 is the comparison between the two models. Less accurate means the model is less accurate compared to the other model as same as more accurate, equal means that both models provide the same result.

In the first test to validate both models, the classification report function used to provide us a table of classification metrics, which will be explained in the following sentences. The precision of the Random Forest model in macro and weighted averages performs better than the K Nearest Neighbors model as is shown in figures 7 and 8, so that what it meant by more and less accurate in table 1. Also, the recall and f1-score metrics show better performance in the Random Forest model. In the support metric, they are equal, as meant in Table 1. The next test is the confusion matrix, both diagonal and nonzero values performance as is shown in figures 7 and 8 the Random Forest model better and provided more accurate prediction. In the last test, the accuracy score, the Random Forest model again more accurate than the K Nearest Neighbors model.

Table 1: Comparing Classification Machine Learning Models

Metrics	K Nearest Neighbors	Random Forest
precision	Less accurate	More accurate
recall	Less accurate	More accurate
f1-score	Less accurate	More accurate
support	Equal	Equal
confusion matrix	Less accurate	More accurate
accuracy score	Less accurate	More accurate

Figure 7

KNN validating metrics results

```
K Nearest Neighbors:
Classification Report:

              precision    recall  f1-score   support

      Setosa       1.00      0.89      0.94         9
  Versicolor       0.77      1.00      0.87        10
   Virginica       1.00      0.82      0.90        11

    accuracy                           0.90        30
   macro avg       0.92      0.90      0.90        30
weighted avg       0.92      0.90      0.90        30


Classification Confusion Matrix:

[[ 8  1  0]
 [ 0 10  0]
 [ 0  2  9]]

Classification Accuracy Score:

0.9
```

Figure 8

Random Forest validating metrics results

```
Random Forest:
Classification Report:

              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00         9
  Versicolor       0.83      1.00      0.91        10
   Virginica       1.00      0.82      0.90        11

    accuracy                           0.93        30
   macro avg       0.94      0.94      0.94        30
weighted avg       0.94      0.93      0.93        30


Classification Confusion Matrix:

[[ 9  0  0]
 [ 0 10  0]
 [ 0  2  9]]

Classification Accuracy Score:

0.9333333333333333
```

References

Deitel, P. J., & Deitel, H. M. (2020). *Intro to Python for computer Science and data science learning to program with AI, big data and the cloud*. Hudson Street, NY: Pearson Education.

Kalb, I. (2016). *Learn to program with Python*. Berkeley, CA: Apress.

Mueller, J., & Emid, A. (2018). *Beginning programming with Python® for dummies®*. Hoboken, NJ: For Dummies, a Wiley brand.

Gramfort, A., Blondel, M., Grisel, O., Mueller, A., Martin, E., Patrini, G., Chang, E. (2017). Sklearn.preprocessing.StandardScaler¶. Retrieved December 01, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html