Arun Mathew Kurian Follow

Oct 23, 2018 · 4 min read · Listen

Build a gender classifier in Python using Scikit-learn

This is my very first blog on Machine Learning (ML). This is a beginner level post, which is based on the Gender Classification challenge from Siraj Raval's Introduction — Learn Python for Data Science #1 video. The video introduces the Python library sklearn or scikit-learn that is used to perform various ML operations.

In this problem we have data regarding a person's height, weight, and shoe size. We will be making a classifier which predicts the gender of a new person, using this data. The classification should be done using multiple classifiers and the most accurate one should be identified.

So, the first thing to do after setting up Python and pip, is to install **scikit-learn.**

scikit-learn is a simple and efficient tool for data mining and data analysis. It is built on NumPy, SciPy, and matplotlib.

scikit-learn can be installed using the command

```
pip install scikit-learn
```

Now let us create our **gender_classifier.py** file.

First, we need to declare the data and labels we are giving the classifier.

```
X = [[181, 80, 44], [177, 70, 43], [160, 60, 38],
     [154, 54, 37],[166, 65, 40], [190, 90, 47], [175, 64, 39],
     [177, 70, 40], [159, 55, 37], [171, 75, 42],
     [181, 85, 43], [168, 75, 41], [168, 77, 41]]

Y = [‘male’, ‘male’, ‘female’, ‘female’, ‘male’, ‘male’,
    ‘female’,‘female’,‘female’, ‘male’, ‘male’,
     ‘female’, ‘female’]
```

Here `X` is a collection of height, weight, and shoe size combinations. `Y` contains the gender labels associated with each combination.

We can now declare the `test_data`, or the data to be classified.

```
test_data = [[190, 70, 43],[154, 75, 38],[181,65,40]]
test_labels = [‘male’,’female’,’male’]
```

We are going to use four classifiers provided by scikit-learn. They are:

1. decision tree
2. random forest
3. logistic regression
4. Support Vector Classifier (SVC)

Decision tree classifier

The decision tree classifier iteratively divides the working area (plot) into subpart by identifying lines. TK I divide the work area until it has divided into classes that are pure, or some criteria of classifier attributes are met.

There are three key terms related to decision tree classifiers:

1. Impurity

Impurity is when we have a traces of one class division into other.

2. Entropy

Entropy is a degree of randomness of elements. In other words, it is a measure of impurity.

It is the negative summation of probability times the log of the probability of item x.

3. Information gain

Information Gain (n) = Entropy(x) — ([weighted average] * entropy(children for feature))

At every stage, a decision tree selects the one that gives the best information gain. An information gain of 0 means the feature does not divide the working set at all.

To add this classifier to our program first we need to import the library tree from sklearn.

```
from sklearn import tree
```

The code for the classification is as follows

```
#DecisionTreeClassifier
dtc_clf = tree.DecisionTreeClassifier()
dtc_clf = dtc_clf.fit(X,Y)
dtc_prediction = dtc_clf.predict(test_data)
print dtc_prediction
```

Random Forest Classifier

Random Forest classifier is an ensemble algorithm. Ensemble algorithms are algorithms which combines more than one algorithms of the same or different kind for classifying objects. Random forest classifier creates a set of decision trees from a randomly selected subset of the training set. It then aggregates the votes from different decision trees to decide the final class of the test object.

Alternatively, the random forest can apply weight concept for considering the impact of result from any decision tree. Tree with high error rate is given low weight value and vice-versa. This would increase the decision impact of trees with low error rate.

To add this classifier to our program first we need to import.

```
from sklearn.ensemble import RandomForestClassifier
```

The code for the classification is as follows

```
#RandomForestClassifier
rfc_clf = RandomForestClassifier()
rfc_clf.fit(X,Y)
rfc_prediction = rfc_clf.predict(test_data)
print rfc_prediction
```

Logistic Regression

Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature. Dichotomous means there are only two possible classes. For example, it can be used for cancer detection problems. It computes the probability of an event occurrence.

It is a special case of linear regression where the target variable is categorical in nature. It uses a log of odds as the dependent variable. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function.

To add this classifier to our program first we add the header line.

```
from sklearn.linear_model import LogisticRegression
```

The code for the classification is as follows

```
#LogisticRegression
l_clf = LogisticRegression()
l_clf.fit(X,Y)
l_prediction = l_clf.predict(test_data)
print l_prediction
```

SVC (Support Vector Classifier)

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. The given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space, this hyperplane is a line dividing a plane into two parts wherein each class lay in either side.

To add this classifier to our program first we add the header line.

```
from sklearn.svm import SVC
```

The code for the classification is as follows

```
#Support Vector Classifier
s_clf = SVC()
s_clf.fit(X,Y)
s_prediction = s_clf.predict(test_data)
print s_prediction
```

Now to find the most accurate classifier we need to add the package **numpy** and **accuracy_score**

```
import numpy as np
from sklearn.metrics import accuracy_score
```

The accuracy can be calculated as

```
#accuracy scores
dtc_tree_acc = accuracy_score(dtc_prediction,test_labels)
```

```
rfc_acc = accuracy_score(rfc_prediction,test_labels)
l_acc = accuracy_score(l_prediction,test_labels)
s_acc = accuracy_score(s_prediction,test_labels)
```

And the most accurate one can be obtained by the following code

```
classifiers = [‘Decision Tree’, ‘Random Forest’, ‘Logistic
Regression’ , ‘SVC’]
accuracy = np.array([dtc_tree_acc, rfc_acc, l_acc, s_acc])
max_acc = np.argmax(accuracy)
print(classifiers[max_acc] + ‘ is the best classifier for this
problem’)
```

Now we can execute the program as

> *test_data = [[190, 70, 43],[154, 75, 38],[181,65,40]]*
>
> *test_labels = [‘male’,’female’,’male’]*

Output:

> *[‘male’ ‘female’ ‘female’]*
>
> *[‘male’ ‘female’ ‘female’]*

> *['female' 'female' 'female']*
> *['female' 'female' 'female']*
>
> *Decision Tree is the best classifier for this problem*

The completed code can be found at

https://github.com/amkurian/gender_classification/blob/master/gender_classifier.py

This is a good exercise to get introduced to various classifiers used in machine learning.

Machine Learning Python Scikit Sklearn Classification