Практическая работа №3.Методы классификации.

Шаги для построения классификатора в Python
Для построения классификатора в Python мы будем использовать Python 3 и Scikit-learn, который является инструментом для машинного обучения. Выполните следующие шаги, чтобы построить классификатор в Python.

Шаг 1 — Импорт Scikit-Learn
Это был бы самый первый шаг для создания классификатора в Python. На этом этапе мы установим пакет Python под названием Scikit-learn, который является одним из лучших модулей машинного обучения в Python. Следующая команда поможет нам импортировать пакет.

```
import sklearn
```

Шаг 2 — Импорт набора данных Scikit-learn
На этом этапе мы можем начать работать с набором данных для нашей модели машинного обучения. Здесь мы собираемся использовать диагностическую базу данных рака молочной железы в Висконсине. Набор данных включает в себя различную информацию о раковых опухолях молочной железы, а также метки
классификации злокачественных или доброкачественных . Набор данных содержит 569 экземпляров или данных о 569 опухолях и содержит информацию о 30 атрибутах или признаках, таких как радиус опухоли, текстура, гладкость и площадь. С помощью следующей команды мы можем импортировать набор данных рака молочной железы Scikit-learn

```
from sklearn.datasets import load_breast_cancer
```

Теперь следующая команда загрузит набор данных.

```
data = load_breast_cancer()
```

Ниже приведен список важных ключей словаря

- Имена меток классификации (target_names)
- Фактические метки (цель)
- Имена атрибутов / функций (feature_names)
- Атрибут (данные)

Теперь с помощью следующей команды мы можем создать новые переменные для каждого важного набора информации и назначить данные. Другими словами, мы можем организовать данные с помощью следующих команд —

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

Теперь, чтобы сделать его более понятным, мы можем напечатать метки классов, метку первого экземпляра данных, имена наших функций и значение функции с помощью следующих команд:

```
print(label_names)
```

Приведенная выше команда напечатает имена классов, которые являются злокачественными и доброкачественными соответственно. Это показано как результат ниже

```
['malignant' 'benign']
```

Теперь команда ниже покажет, что они отображаются в двоичные значения 0 и 1. Здесь 0 представляет злокачественный рак, а 1 представляет доброкачественный рак. Вы получите следующий вывод —

```
print(labels[0])

0
```

Две команды, приведенные ниже, создадут имена и значения функций.

```
print(feature_names[0])

mean radius

print(features[0])

[ 1.79900000e+01 1.03800000e+01 1.22800000e+02 1.00100000e+03

  1.18400000e-01 2.77600000e-01 3.00100000e-01 1.47100000e-01

  2.41900000e-01 7.87100000e-02 1.09500000e+00 9.05300000e-01

  8.58900000e+00 1.53400000e+02 6.39900000e-03 4.90400000e-02

  5.37300000e-02 1.58700000e-02 3.00300000e-02 6.19300000e-03

  2.53800000e+01 1.73300000e+01 1.84600000e+02 2.01900000e+03

1.62200000e-01 6.65600000e-01 7.11900000e-01 2.65400000e-01

  4.60100000e-01 1.18900000e-01]
```

Из вышеприведенного вывода видно, что первый экземпляр данных представляет собой злокачественную опухоль, радиус которой составляет 1.7990000e + 01.

Шаг 3 — Организация данных в наборы

На этом этапе мы разделим наши данные на две части, а именно обучающий набор и тестовый набор. Разделение данных на эти наборы очень важно, потому что мы должны проверить нашу модель на невидимых данных. Чтобы разделить данные на наборы, в sklearn есть функция под названием **train_test_split ()** . С помощью следующих команд мы можем разделить данные в этих наборах:

```
from sklearn.model_selection import train_test_split
```

Приведенная выше команда импортирует функцию **train_test_split** из sklearn, а приведенная ниже команда разделит данные на данные обучения и тестирования. В приведенном ниже примере мы используем 40% данных для тестирования, а оставшиеся данные будут использованы для обучения модели.

```
train, test, train_labels, test_labels = train_test_split(features,labels,test_size =
0.40, random_state = 42)
```

Шаг 4 — Построение модели

На этом этапе мы будем строить нашу модель. Мы собираемся использовать наивный байесовский алгоритм для построения модели. Следующие команды могут быть использованы для построения модели

```
from sklearn.naive_bayes import GaussianNB
```

Приведенная выше команда импортирует модуль GaussianNB. Теперь следующая команда поможет вам инициализировать модель.

```
gnb = GaussianNB()
```

Мы будем обучать модель, подгоняя ее к данным с помощью gnb.fit ().

```
model = gnb.fit(train, train_labels)
```

Шаг 5 — Оценка модели и ее точности

На этом этапе мы собираемся оценить модель, сделав прогнозы на наших тестовых данных. Тогда мы узнаем и его точность. Для прогнозирования мы будем использовать функцию предиката (). Следующая команда поможет вам сделать это —

```
preds = gnb.predict(test)
print(preds)
[1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1
 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0
 0 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 0 0 0 0
 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 1 0 1 1 0
 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0
 1 1 0 1 1 1 1 1 1 0 0 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 1 1 0
 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 1]
```

Приведенные выше серии 0 и 1 являются предсказанными значениями для классов опухолей — злокачественных и доброкачественных.

Теперь, сравнивая два массива, а именно **test_labels** и **preds** , мы можем выяснить точность нашей модели. Мы будем использовать функцию **precision_score ()** для определения точности. Рассмотрим следующую команду для этого

```
from sklearn.metrics import accuracy_score
print(accuracy_score(test_labels,preds))
0.951754385965
```

Результат показывает, что классификатор NaïveBayes имеет точность 95,17%.

Таким образом, с помощью описанных выше шагов мы можем построить наш классификатор в Python.

Наивный байесовский классификатор

Наивный байесовский метод — это метод классификации, используемый для построения классификатора с использованием теоремы Байеса. Предполагается, что предикторы независимы. Проще говоря, это предполагает, что наличие определенной функции в классе не связано с наличием любой другой функции. Для построения наивного байесовского классификатора нам нужно использовать библиотеку python под названием scikit learn. Существует три типа наивных байесовских моделей, названные **Gaussian, Multinomial и Bernoulli** .

Для построения наивной модели байесовского классификатора машинного обучения нам нужно следующее.

Dataset

Мы собираемся использовать набор данных с именем База данных диагностики рака молочной железы в Висконсине. Набор данных включает в себя различную информацию о

раковых опухолях молочной железы, а также метки классификации **злокачественных** или **доброкачественных** . Набор данных содержит 569 экземпляров или данных о 569 опухолях и содержит информацию о 30 атрибутах или признаках, таких как радиус опухоли, текстура, гладкость и площадь. Мы можем импортировать этот набор данных из пакета sklearn.

Наивная байесовская модель

Для построения наивного байесовского классификатора нам нужна наивная байесовская модель. Как говорилось ранее, существует три типа наивных байесовских моделей: **Гауссовская, Многочленовая** и **Бернуллиевская** . Здесь, в следующем примере мы будем использовать гауссовскую наивную байесовскую модель.

Используя вышеизложенное, мы собираемся построить наивную модель машинного обучения Байеса, чтобы использовать информацию о опухоли, чтобы предсказать, является ли опухоль злокачественной или доброкачественной.

Для начала нам нужно установить модуль sklearn. Это можно сделать с помощью следующей команды

```
import Sklearn
```

Теперь нам нужно импортировать набор данных с именем База данных диагностики рака молочной железы в Висконсине.

```
from sklearn.datasets import load_breast_cancer
```

Теперь следующая команда загрузит набор данных.

```
data = load_breast_cancer()
```

Данные могут быть организованы следующим образом —

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

Теперь, чтобы прояснить ситуацию, мы можем напечатать метки классов, метку первого экземпляра данных, имена наших функций и значение функции с помощью следующих команд:

```
print(label_names)
```

Приведенная выше команда напечатает имена классов, которые являются злокачественными и доброкачественными соответственно. Это показано как результат ниже

```
['malignant' 'benign']
```

Теперь приведенная ниже команда покажет, что они отображаются в двоичные значения 0 и 1. Здесь 0 представляет злокачественный рак, а 1 представляет доброкачественный рак. Это показано как результат ниже

```
print(labels[0])
0
```

Следующие две команды создадут имена и их значения.

```
print(feature_names[0])
```

```
mean radius
print(features[0])

[ 1.79900000e+01 1.03800000e+01 1.22800000e+02 1.00100000e+03
  1.18400000e-01 2.77600000e-01 3.00100000e-01 1.47100000e-01
  2.41900000e-01 7.87100000e-02 1.09500000e+00 9.05300000e-01
  8.58900000e+00 1.53400000e+02 6.39900000e-03 4.90400000e-02
  5.37300000e-02 1.58700000e-02 3.00300000e-02 6.19300000e-03
  2.53800000e+01 1.73300000e+01 1.84600000e+02 2.01900000e+03
1.62200000e-01 6.65600000e-01 7.11900000e-01 2.65400000e-01
  4.60100000e-01 1.18900000e-01]
```

Из вышеприведенного вывода видно, что первый экземпляр данных представляет собой злокачественную опухоль, основной радиус которой составляет 1.7990000e + 01.
Для тестирования нашей модели на невидимых данных нам нужно разделить наши данные на данные обучения и тестирования. Это можно сделать с помощью следующего кода

```
from sklearn.model_selection import train_test_split
```

Приведенная выше команда импортирует функцию **train_test_split** из sklearn, а приведенная ниже команда разделит данные на данные обучения и тестирования. В приведенном ниже примере мы используем 40% данных для тестирования, а данные напоминания будут использованы для обучения модели.

```
train, test, train_labels, test_labels =
train_test_split(features,labels,test_size = 0.40, random_state = 42)
```

Теперь мы строим модель с помощью следующих команд:

```
from sklearn.naive_bayes import GaussianNB
```

Приведенная выше команда импортирует модуль GaussianNB . Теперь с помощью команды, приведенной ниже, нам нужно инициализировать модель.

```
gnb = GaussianNB()
```

Мы будем обучать модель, подгоняя ее к данным с помощью **gnb.fit ()** .

```
model = gnb.fit(train, train_labels)
```

Теперь оцените модель, сделав прогноз на тестовых данных, и это можно сделать следующим образом:

```
preds = gnb.predict(test)
print(preds)

[1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0
 0 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0 1 1 1 1 1 1 1 0 1 1 0 0 0 0
0 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 1 0 1 1 0
 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0
 1 1 0 1 1 1 1 1 1 0 0 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 1 1 0
 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 1]
```

Вышеуказанные серии 0 и 1 являются предсказанными значениями для классов опухолей, т.е. злокачественных и доброкачественных.
Теперь, сравнивая два массива, а именно test_labels и preds , мы можем выяснить точность нашей модели. Мы будем использовать функцию precision_score () для определения точности. Рассмотрим следующую команду

```
from sklearn.metrics import accuracy_score
print(accuracy_score(test_labels,preds))
0.951754385965
```

Результат показывает, что классификатор NaïveBayes имеет точность 95,17%.
Это был классификатор машинного обучения, основанный на наивной модели Байса-Гаусса.

Классификатор дерева решений
Дерево решений — это, по сути, блок-схема двоичного дерева, где каждый узел разделяет группу наблюдений в соответствии с некоторой характеристической переменной.
Здесь мы строим классификатор дерева решений для прогнозирования мужчин или женщин. Мы возьмем очень маленький набор данных, имеющий 19 образцов. Эти образцы будут состоять из двух признаков — «рост» и «длина волос».

Необходимое условие
Для построения следующего классификатора нам нужно установить pydotplus и graphviz. По сути, graphviz — это инструмент для рисования графики с использованием точечных файлов, а pydotplus— это модуль для языка Graphviz’s Dot. Его можно установить с помощью менеджера пакетов или pip.
Теперь мы можем построить классификатор дерева решений с помощью следующего кода Python. Для начала давайте импортируем некоторые важные библиотеки следующим образом:

```
import pydotplus
from sklearn import tree
from sklearn.datasets import load_iris
from sklearn.metrics import classification_report
from sklearn import cross_validation
import collections
```

Теперь нам нужно предоставить набор данных следующим образом:

```
X = [[165,19],[175,32],[136,35],[174,65],[141,28],[176,15],[131,32],
[166,6],[128,32],[179,10],[136,34],[186,2],[126,25],[176,28],[112,38],
[169,9],[171,36],[116,25],[196,25]]

Y = ['Man','Woman','Woman','Man','Woman','Man','Woman','Man','Woman',
'Man','Woman','Man','Woman','Woman','Woman','Man','Woman','Woman','Man']
data_feature_names = ['height','length of hair']

X_train, X_test, Y_train, Y_test = cross_validation.train_test_split
(X, Y, test_size=0.40, random_state=5)
```

После предоставления набора данных нам нужно подобрать модель, что можно сделать следующим образом:

```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X,Y)
```

Предсказание может быть сделано с помощью следующего кода Python

```
prediction = clf.predict([[133,37]])

print(prediction)
```

Мы можем визуализировать дерево решений с помощью следующего кода Python

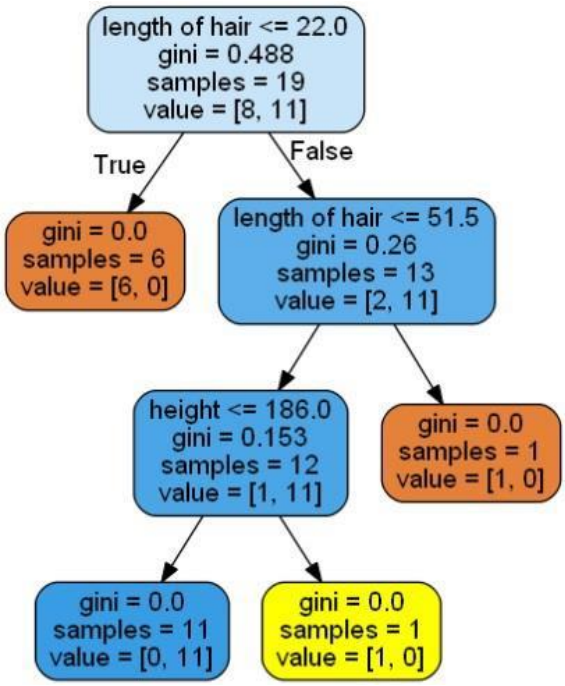
```
dot_data = tree.export_graphviz(clf,feature_names = data_feature_names,
            out_file = None,filled = True,rounded = True)
graph = pydotplus.graph_from_dot_data(dot_data)
colors = ('orange', 'yellow')
edges = collections.defaultdict(list)

for edge in graph.get_edge_list():
edges[edge.get_source()].append(int(edge.get_destination()))

for edge in edges: edges[edge].sort()

for i in range(2):dest = graph.get_node(str(edges[edge][i]))[0]
dest.set_fillcolor(colors[i])
graph.write_png('Decisiontree16.png')
```

Это даст прогноз для приведенного выше кода как [‘Woman’] и создаст следующее дерево решений



Мы можем изменить значения функций в прогнозе, чтобы проверить его.

Случайный лесной классификатор

Как мы знаем, методы ансамбля — это методы, которые объединяют модели машинного обучения в более мощную модель машинного обучения. Случайный лес, коллекция деревьев решений, является одним из них. Это лучше, чем одно дерево решений, потому что, сохраняя предсказательные полномочия, оно может уменьшить чрезмерную подгонку путем усреднения результатов. Здесь мы собираемся внедрить модель случайного леса в наборе данных по обучению рака.

Импортируйтенеобходимыепакеты

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
import matplotlib.pyplot as plt
import numpy as np
```

Теперь нам нужно предоставить набор данных, который можно сделать следующим образом

```
cancer = load_breast_cancer()
X_train, X_test, y_train,
y_test = train_test_split(cancer.data, cancer.target, random_state = 0)
```

После предоставления набора данных нам нужно подобрать модель, что можно сделать следующим образом:

```
forest = RandomForestClassifier(n_estimators = 50, random_state = 0)
forest.fit(X_train,y_train)
```

Теперь получите точность обучения и подмножества тестирования: если мы увеличим количество оценщиков, точность подмножества тестирования также будет увеличена.

```
print('Accuracy on the training subset:(:.3f)',format(forest.score(X_train,y_train)))
print('Accuracy on the training subset:(:.3f)',format(forest.score(X_test,y_test)))
```

Теперь, как и дерево решений, случайный лес имеет модуль **feature_importance,** который обеспечит лучшее представление о весе объекта, чем дерево решений. Это можно построить и визуализировать следующим образом:

```
n_features = cancer.data.shape[1]
plt.barh(range(n_features),forest.feature_importances_, align='center')
plt.yticks(np.arange(n_features),cancer.feature_names)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.show()
```