Лабораторная работа №2

Алгоритм К ближайших соседей

Библиотеке scikit-learn имеется довольно много алгоритмов классификации, которые мы могли бы использовать для построения модели. В данном примере мы будем использовать классификатор на основе метода k ближайших соседей, который легко интерпретировать. Построение этой модели заключается лишь в запоминании обучающего набора. Для того, чтобы сделать прогноз для новой точки данных, алгоритм находит точку в обучающем наборе, которая находится ближе всего к новой точке. Затем он присваивает метку, принадлежащую этой точке обучающего набора, новой точке данных.

"К" в методе k ближайших соседей означает, что вместо того, чтобы использовать лишь ближайшего соседа новой точки данных, мы в ходе обучения можем рассмотреть любое фиксированное число (k) соседей (например, рассмотреть ближайшие три или пять соседей). Тогда мы можем сделать прогноз для точки данных, используя класс, которому принадлежит большинство ее соседей.
scikit-learn все модели машинного обучения реализованы в собственных классах, называемых классами Estimator. Алгоритм классификации на основе метода k ближайших соседей реализован в классификаторе ***KNeighborsClassifier*** модуля neighbors. Прежде чем использовать эту модель, нам нужно создать объект-экземпляр класса. Это произойдет, когда мы зададим параметры модели. Самым важным параметром KNeighborsClassifier является количество соседей, которые мы установим равным 1:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

Объект ***knn*** включает в себя алгоритм, который будет использоваться для построения модели на обучающих данных, а также алгоритм, который сгенерирует прогнозы для новых точек данных. Он также будет содержать информацию, которую алгоритм извлек из обучающих данных. В случае с KNeighborsClassifier он будет просто хранить обучающий набор.

Для построения модели на обучающем наборе, мы вызываем метод fit объекта ***knn***, который принимает в качестве аргументов массив *NumPy X_train*, содержащий обучающие данные, и массив *NumPy y_train*, соответствующий обучающим меткам:

```
print("Ключи iris_dataset: \n{}".format(iris_dataset.keys()))
```

```
KNeighborsClassifier(
  algorithm='auto',
  leaf_size=30,
  metric='minkowski',
  metric_params=None,
  n_jobs=1,
  n_neighbors=1,
  p=2,
  weights='uniform'
)
```

Метод *fit* возвращает сам объект ***knn*** (и изменяет его), таким образом, мы получаем строковое представление нашего классификатора. Оно показывает нам, какие параметры были использованы при создании модели. Почти все параметры имеют значения по умолчанию, но вы также можете обнаружить параметр *n_neighbor=1*, заданный нами. Большинство моделей в scikit-learn имеют массу параметров, но большая часть из них связана с оптимизацией скорости вычислений или предназначена для особых случаев использования. Вам не нужно беспокоиться о других параметрах, приведенных здесь. Вывод модели в scikit-learn может быть очень длинным, но не нужно пугаться его.

Получение прогноза

Теперь мы можем получить прогнозы, применив эту модель к новым данным, по которым мы еще не знаем правильные метки. Представьте, что мы нашли в дикой природе ирис с длиной чашелистика 5 см, шириной чашелистика 2.9 см, длиной лепестка 1 см и шириной лепестка 0.2 см. К какому сорту ириса нужно отнести этот цветок? Мы можем поместить эти данные в массив NumPy, снова вычисляя форму массива, т .е. количество примеров (1), умноженное на количество признаков (4):

```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("форма массива X_new: {}".format(X_new.shape))
```

```
форма массива X_new: (1, 4)
```

Обратите внимание, что мы записали измерения по одному цветку в двумерный массив NumPy, поскольку *scikit-learn работает с двумерными массивами данных*.

Чтобы сделать прогноз, мы вызываем метод ***predict*** объекта knn:

```
prediction = knn.predict(X_new)
print("Прогноз: {}".format(prediction))
print("Спрогнозированная метка: {}".format(
    iris_dataset['target_names'][prediction])
)
```

```
Прогноз: [0]
Спрогнозированная метка: ['setosa']
```

Наша модель предсказывает, что этот новый цветок ириса принадлежит к классу 0, что означает сорт setosa. Но как узнать, можем ли мы доверять нашей модели? Правильный сорт ириса для этого примера нам неизвестен, а ведь именно получение правильных прогнозов и является главной задачей построения модели!

Оценка качества модели

Это тот самый момент, когда нам понадобится созданный ранее тестовый набор. Эти данные не использовались для построения модели, но мы знаем правильные сорта для каждого ириса в тестовом наборе.

Таким образом, мы можем сделать прогноз для каждого ириса в тестовом наборе и сравнить его с фактической меткой (уже известным сортом). Мы можем оценить качество модели, вычислив правильность (accuracy) – процент цветов, для которых модель правильно спрогнозировала сорта:

```
y_pred = knn.predict(X_test)
print("Прогнозы для тестового набора:\n {}".format(y_pred)
print("Правильность на тестовом наборе:
{:.2f}".format(np.mean(y_pred == y_test)))
```

```
Прогнозы для тестового набора:
[210202011121111011002100200110210221021]

Правильность на тестовом наборе: 0.97
```

Кроме того, мы можем использовать метод ***score*** объекта knn, который вычисляет правильность модели для тестового набора:

```
print("Правильность на тестовом наборе: {:.2f}".format(
   knn.score(X_test, y_test)
))
```

```
Правильность на тестовом наборе: 0.97
```

Правильность этой модели для тестового набора составляет около 0.97, что означает, что мы дали правильный прогноз для 97% ирисов в тестовом наборе. При некоторых математических допущениях, это означает, что мы можем ожидать, что наша модель в 97% случаев даст правильный прогноз для новых ирисов. Для нашего ботаника-любителя этот высокий уровень правильности означает, что наша модель может быть достаточно надежной в использовании. В следующих главах мы обсудим, как можно улучшить эффективность модели, и с какими подводными камнями можно столкнуться при настройке модели.

Мы использовали набор данных, в котором эксперт уже предварительно классифицировал ирисы для построения модели, таким образом, мы решали задачу обучения с учителем. Было три возможных сорта ирисов – *setosa*, *versicolor* и *virginica*, что делало нашу задачу задачей ***3-классовой классификации***. В задаче классификации возможные сорта ирисов называются *классами (classes)* а сами названия сортов – *метками (labels)*.

Набор данных Iris состоит из *двух массивов NumPy*: один содержит данные и в scikit-learn обозначается как X, другой содержит правильные или нужные ответы и обозначается как y. Массив X представляет собой двумерный массив признаков, в котором одна строка соответствует одной

точке данных, а один столбец – одному признаку. Массив y представляет собой одномерный массив, который для каждого примера содержит метку класса, целое число от 0 до 2.

Мы разделили наш набор данных на обучающий набор (training set), чтобы построить нашу модель, а также тестовый набор (test set), чтобы оценить, насколько хорошо наша модель будет классифицировать новые, ранее незнакомые ей данные.

Мы выбрали *алгоритм классификации k ближайших соседей*, который генерирует прогноз для новой точки данных, рассматривая ее ближайшего соседа(ей) в обучающем наборе. Все это реализовано в классе KNeighborsClassifier, который содержит алгоритм, строящий модель, а также алгоритм, который дает прогнозы, используя построенную модель. Мы создали объект-экземпляр класса, задав параметры. Затем мы построили модель, вызвав метод fit и передав обучающие данные (X_train) и обучающие ответы (y_train) в качестве параметров. Мы оценили качество модели с использованием метода score, который вычисляет правильность модели. Мы применили метод score к тестовым данным и тестовым ответам и обнаружили, что наша модель демонстрирует правильность около 97%. Это означает, что модель выдает правильные прогнозы для 97% наблюдений тестового набора.

Это убедило нас в том, что модель можно применить к новым данным (в нашем примере это измерения характеристик новых цветов), и мы надеемся, что эта модель даст правильные прогнозы в 97% случаев.

Ниже приводится краткое изложение программного кода, необходимого для всей процедуры обучения и оценки модели:

```
X_train, X_test, y_train, y_test = train_test_split(
iris_dataset['data'], iris_dataset['target'], random_state=0)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

print("Правильность на тестовом наборе:
{:.2f}".format(knn.score(X_test, y_test)))
```

```
Правильность на тестовом наборе: 0.97
```

Этот фрагмент содержит базовый код, необходимый для применения любого алгоритма машинного обучения с помощью scikit-learn. Методы **fit**, **predict** и **score** являются общими для моделей контролируемого обучения в scikit-learn и, используя принципы, приведенные в этой главе, вы можете применить эти модели для решения различных задач машинного обучения.

Код к лабораторной работе:

```
import pandas as pds
import mglearn
import matplotlib.pyplot as plt
import numpy as np

from sklearn.datasets import load_iris
iris_dataset = load_iris()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
iris_dataset['data'], iris_dataset['target'], random_state=0)

print("форма массива X_train: {}".format(X_train.shape))
print("форма массива y_train: {}".format(y_train.shape))
print("форма массива X_test: {}".format(X_test.shape))
print("форма массива y_test: {}".format(y_test.shape))

iris_dataframe = pds.DataFrame(X_train, columns=iris_dataset.feature_names)

from pandas.plotting import scatter_matrix

grr = scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15), marker='o',
        hist_kwds={'bins': 20}, s=60, alpha=.8, cmap=mglearn.cm3)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train, y_train)

X_new = np.array([[5, 2.9, 1, 0.2]])
print("форма массива X_new: {}".format(X_new.shape))

prediction = knn.predict(X_new)
print("Прогноз: {}".format(prediction))
print("Спрогнозированная метка: {}".format(iris_dataset['target_names'][prediction]))

y_pred = knn.predict(X_test)
print("Прогнозы для тестового набора:\n {}".format(y_pred))
print("Правильность на тестовом наборе: {:.2f}".format(np.mean(y_pred == y_test)))
print("Правильность на тестовом наборе: {:.2f}".format(knn.score(X_test, y_test)))

X_train, X_test, y_train, y_test = train_test_split(
        iris_dataset['data'], iris_dataset['target'], random_state=0)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
print("Правильность на тестовом наборе: {:.2f}".format(knn.score(X_test, y_test)))
```