Лабораторная работа №22

Линейные модели для мультиклассовой классификации

Многие линейные модели классификации предназначены лишь для бинарной классификации и не распространяются на случай мультиклассовой классификации (за исключением логистической регрессии). Общераспространенный подход, позволяющий распространить алгоритм бинарной классификации на случай мультиклассовой классификации называет подходом ***один против остальных*** *(one-vs.-rest)*. В подходе «один против остальных» для каждого класса строится бинарная модель, которая пытается отделить этот класс от всех остальных, в результате чего количество моделей определяется количеством классов. Для получения прогноза точка тестового набора подается на все бинарные классификаторы. Классификатор, который выдает по своему классу наибольшее значение, «побеждает» и метка этого класса возвращается в качестве прогноза.

Используя бинарный классификатор для каждого класса, мы получаем один *вектор коэффициентов (**w**)* и одну *константу (**b**)* по каждому классу. Класс, который получает наибольшее значение согласно нижеприведенной формуле, становится присвоенной меткой класса:

$$w[0] * x[0] \quad w[1] * x[1] \quad \ldots. \; w[p] * x[p] \quad b$$

Математический аппарат мультиклассовой логистической регрессии несколько отличается от подхода «один против остальных», однако он также дает один вектор коэффициентов и константу для каждого класса и использует тот же самый способ получения прогнозов.

Давайте применим метод «один против остальных» к простому набору данных с 3-классовой классификацией. Мы используем двумерный массив данных, где каждый класс задается данными, полученными из гауссовского распределения (см. рис. 1):

```
from sklearn.datasets import make_blobs

X, y = make_blobs(random_state=42)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)

plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.legend(["Класс 0", "Класс 1", "Класс 2"])
```

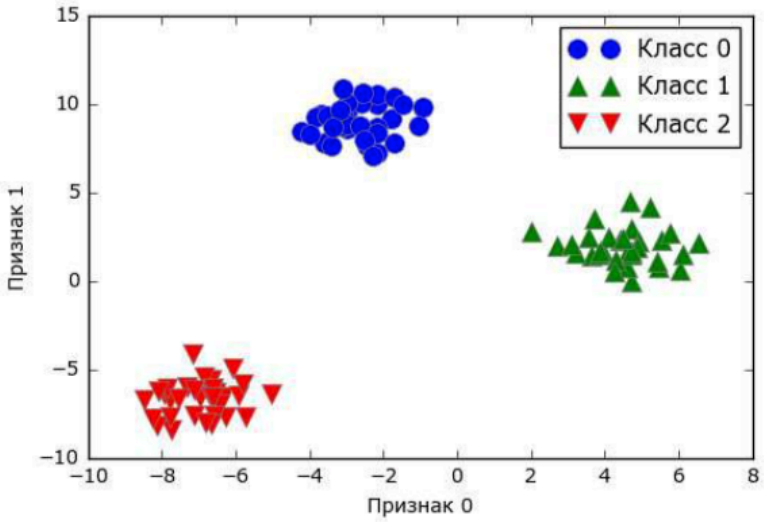

Рис. 1 Двумерный синтетический набор данных, содержащий три класса

Теперь обучаем классификатор **LinearSVC** на этом наборе данных:

```
linear_svm = LinearSVC().fit(X, y)
print("Форма коэффициента: ", linear_svm.coef_.shape)
print("Форма константы: ", linear_svm.intercept_.shape)
```

```
Форма коэффициента: (3, 2)
Форма константы: (3,)
```

Мы видим, что атрибут **coef_** имеет форму (3, 2), это означает, что *каждая строка coef_* содержит вектор коэффициентов для каждого из трех классов, а каждый столбец содержит коэффициент для конкретного признака (в этом наборе данных их два). Атрибут **intercept_** теперь является одномерным массивом, в котором записаны константы классов.

Давайте визуализируем линии (границы принятия решений), полученные с помощью трех бинарных классификаторов (рис. 2):

```
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_, ['b', 'r', 'g']):
    plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)

plt.ylim(-10, 15)
plt.xlim(-10, 8)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.legend(
   [
     'Класс 0', 'Класс 1', 'Класс 2', 'Линия класса 0', 'Линия класса 1', 'Линия класса 2'
   ],
   loc=(1.01, 0.3)
)
```

Видно, что все точки, принадлежащие классу 0 в обучающих данных, находятся выше линии, соответствующей классу 0. Это означает, что они отнесены к «классу 0» данного бинарного классификатора. Точки класса 0 находятся выше линии, соответствующей классу 2. Это означает, что они классифицируются бинарным классификатором для класса 2 как «остальные». Точки, принадлежащие классу 0, находятся слева от линии, соответствующей классу 1. Это означает, что бинарный классификатор для класса 1 также классифицирует их как «остальные». Таким образом, итоге любая точка в этой области будет отнесена к классу 0 (результат, получаемый по формуле для классификатора 0, больше нуля, тогда как для двух остальных классов он меньше нуля).

Однако что насчет треугольника в середине графика? Все три бинарных классификатора относят точки, расположенные там, к «остальным». Какой класс будет присвоен точке, расположенной в треугольнике? Ответ – класс, получивший наибольшее значение по формуле классификации, то есть класс ближайшей линии.

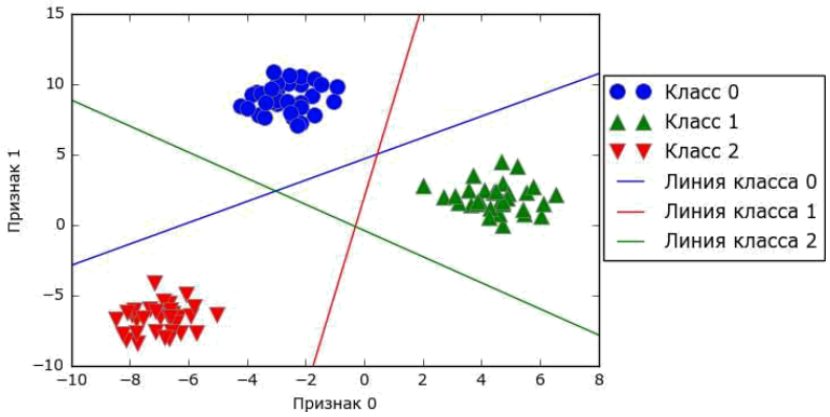

Рис. 2 Границы принятия решений, полученные с помощью трех бинарных классификаторов в рамках подхода «один против остальных»

Следующий пример (рис. 3) показывает прогнозы для всех областей двумерного пространства:

```
mglearn.plots.plot_2d_classification(linear_svm, X, fill=True, alpha=.7)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)

for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_, ['b', 'r', 'g']):

plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.legend(['Класс 0', 'Класс 1', 'Класс 2', 'Линия класса 0', 'Линия класса 1',
'Линия класса 2'], loc=(1.01, 0.3))
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
```

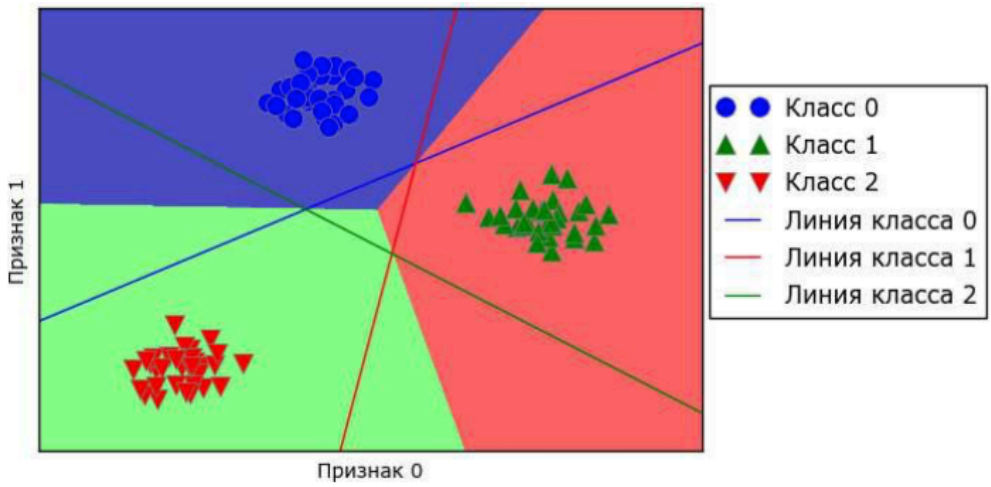

Рис. 3 Мультиклассовые границы принятия решений, полученные с помощью трех бинарных классификаторов в рамках подхода «один против остальных»

Преимущества, недостатки и параметры

Основной параметр линейных моделей – параметр регуляризации, называемый **alpha** в *моделях регрессии* и **C** в *LinearSVC* и *LogisticRegression*. Большие значения **alpha** или маленькие значения **C** означают простые модели. Конкретно для регрессионных моделей настройка этих параметров имеет весьма важное значение. Как правило, поиск **C** и **alpha** осуществляется по логарифмической шкале. Кроме того вы должны решить, какой вид регуляризации нужно использовать: **L1** или **L2**. Если вы полагаете, что на самом деле важны лишь некоторые признаки, следует использовать **L1**. В противном случае используйте установленную по умолчанию **L2** регуляризацию. Еще **L1** регуляризация может быть полезна, если интерпретируемость модели имеет важное значение. Поскольку **L1** регуляризация будет использовать лишь несколько признаков, легче будет объяснить, какие признаки важны для модели и каковы эффекты этих признаков.

Линейные модели очень быстро обучаются, а также быстро прогнозируют. Они масштабируются на очень большие наборы данных, также хорошо работают с разреженными данными. При работе с данными, состоящими из сотен тысяч или миллионов примеров, вас, возможно, заинтересует опция solver='sag' в LogisticRegression и Ridge, которая позволяет получить результаты быстрее, чем настройки по умолчанию. Еще пара опций – это класс SGDClassifier и класс SGDRegressor, реализующие более масштабируемые версии описанных здесь линейных моделей.

Еще одно преимущество линейных моделей заключается в том, что они позволяют относительно легко понять, как был получен прогноз, при помощи формул, которые мы видели ранее для регрессии и классификации. К сожалению, часто бывает совершенно не понятно, почему были получены именно такие коэффициенты. Это особенно актуально, если ваш набор данных содержит высоко коррелированные признаки, в таких случаях коэффициенты сложно интерпретировать.

Как правило, линейные модели хорошо работают, когда количество признаков превышает количество наблюдений. Кроме того, они часто используются на очень больших наборах данных, просто потому, что не представляется возможным обучить другие модели. Вместе с тем в

низкоразмерном пространстве альтернативные модели могут показать более высокую обобщающую способность. В разделе «Ядерные машины опорных векторов» мы рассмотрим несколько примеров, в которых использование линейных моделей не увенчалось успехом.

*Цепочка методов (**method chaining**)*

Во всех моделях scikit-learn метод **fit** возвращает **self**. Это позволяет писать код, приведенный ниже и уже широко использованный нами в этой главе:

```
# создаем экземпляр модели и подгоняем его в одной строке
logreg = LogisticRegression().fit(X_train, y_train)
```

Здесь мы использовали значение, возвращаемое методом **fit (self)**, чтобы присвоить обученную модель переменной logreg. Эта конкатенация вызовов методов (в данном случае **_init_**, а затем **fit**) известна как *цепочка методов* **(method chaining).** Еще одно обшераспространенное применение цепочки методов в scikit-learn – это **связывание методов** fit и predict в **одной строке**:

```
logreg = LogisticRegression()
y_pred = logreg.fit(X_train, y_train).predict(X_test)
```

Наконец, вы можете создать экземпляр модели, подогнать модель и получить прогнозы в одной строке:

```
y_pred = LogisticRegression().fit(X_train, y_train).predict(X_test)
```

Однако этот очень короткий вариант не идеален. В одной строке происходит масса всего, что может сделать код трудночитаемым. Кроме того, подогнанная модель логистической регрессии не сохранена в какой-то определенной переменной, поэтому мы не можем проверить или использовать ее, чтобы получить прогнозы для других данных.

Код к лабораторной работе:

```
import mglearn
import sklearn
import matplotlib.pyplot as plt
import numpy as np


from sklearn.datasets import make_blobs
X, y = make_blobs(random_state=42)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.legend(["Класс 0", "Класс 1", "Класс 2"])
plt.show()

from sklearn.svm import LinearSVC
linear_svm = LinearSVC().fit(X, y)
print("Форма коэффициента: ", linear_svm.coef_.shape)
print("Форма константы: ", linear_svm.intercept_.shape)


mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)

for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_, ['b', 'r', 'g']):
    plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.ylim(-10, 15)
plt.xlim(-10, 8)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")

plt.legend(['Класс 0', 'Класс 1', 'Класс 2', 'Линия класса 0', 'Линия класса 1', 'Линия класса
2'], loc=(1.01, 0.3))
plt.show()


mglearn.plots.plot_2d_classification(linear_svm, X, fill=True, alpha=.7)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_, ['b', 'r', 'g']):
    plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.legend(['Класс 0', 'Класс 1', 'Класс 2', 'Линия класса 0', 'Линия класса 1',
'Линия класса 2'], loc=(1.01, 0.3))
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()
```