Лабораторная работа №5

Линейные модели для классификации

Линейные модели также широко используются в задачах классификации. Давайте посмотрим сначала на бинарную классификацию. В этом случае прогноз получают с помощью следующей формулы:

$$y\hat{} \; w[0] * x[0] \;\; w[1] * x[1] \;\; \ldots . \; w[p] * x[p] \;\; b \;\; 0$$

Формула очень похожа на формулу линейной регрессии, но теперь вместо того, чтобы просто возвратить взвешенную сумму признаков, мы задаем для прогнозируемого значения порог, равный нулю. Если функция меньше нуля, мы прогнозируем класс –1, если она больше нуля, мы прогнозируем класс +1. Это прогнозное правило является общим для всех линейных моделей классификации. Опять же, есть много различных способов найти коэффициенты (w) и константу (b).

Для линейных моделей регрессии выход y^ является линейной функцией признаков: линией, плоскостью или гиперплоскостью (для большого количества измерений). Для линейных моделей классификации ***граница принятия решений*** *(decision boundary)* является линейной функцией аргумента. Другими словами, (бинарный) линейный классификатор – это классификатор, который разделяет два класса с помощью линии, плоскости или гиперплоскости. В этом разделе мы приведем кокретные примеры.

Существует масса алгоритмов обучения линейных моделей. Два критерия задают различия между алгоритмами:

- Измеряемые метрики качества подгонки обучающих данных
- Факт использования регуляризации и вид регуляризации, если она используется

Различные алгоритмы по-разному определяют, что значит «хорошая подгонка обучающих данных». В силу технико-математических причин невозможно скорректировать ***w*** и ***b***, чтобы минимизировать количество неверно классифицированных случаев, выдаваемое алгоритмами, как можно было бы надеяться. С точки зрения поставленных нами целей и различных сфер применения различные варианты метрик качества подгонки (так называемые ***функции потерь*** или *loss functions*) не имеют большого значения.

Двумя наиболее распространенными алгоритмами линейной классификации являются ***логистическая регрессия*** *(logistic regression)*, реализованная в классе ***linear_model.LogisticRegression***, и ***линейный метод опорных векторов*** *(linear support vector machines)* или линейный SVM, реализованный в классе ***svm.LinearSVC*** (SVC расшифровывается как **support vector classifier** – классификатор опорных векторов). Несмотря на свое название, логистическая регрессия является алгоритмом классификации, а не алгоритмом регрессии, и его не следует путать с линейной регрессией.

Мы можем применить модели LogisticRegression и LinearSVC к набору данных forge и визуализировать границу принятия решений, найденную линейными моделями (рис. 5.1):

```
from sklearn.linear_model import LogisticRegression from sklearn.svm import LinearSVC
X, y = mglearn.datasets.make_forge()
fig, axes = plt.subplots(1, 2, figsize=(10, 3))
for model, ax in zip([LinearSVC(), LogisticRegression()], axes):
    clf = model.fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=False, eps=0.5, ax=ax, alpha=.7)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{}".format(clf.__class__.__name__))
    ax.set_xlabel("Признак 0")
    ax.set_ylabel("Признак 1")
axes[0].legend()
```

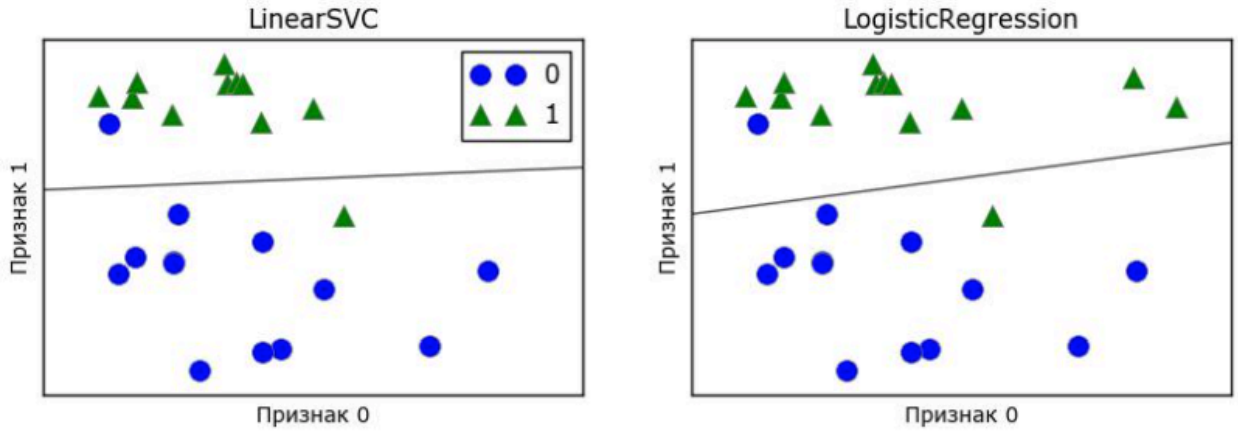

Рис. 5.1 Границы принятия решений линейного SVM и логистической регрессии для набора данных forge (использовались значения параметров по умолчанию)

На этом рисунке, как и раньше, первый признак набора данных forge отложен по оси x, а второй признак – по оси y. Здесь показаны границы принятия решений, найденные LinearSVC и LogisticRegression соответственно. Они представлены в виде прямых линий, отделяющих область значений, классифицированных как класс 1 (в верхней части графика) от области значений, классифицированных как класс 0 (в нижней части графика). Другими словами, любая новая точка данных, которая лежит выше черной линии будет отнесена соответствующей моделью к классу 1, тогда как любая точка, лежащая ниже черной линии, будет отнесена к классу 0.

Обе модели имеют схожие границы принятия решений. Обратите внимание, что обе модели неправильно классифицировали две точки. *По умолчанию обе модели используют L2 регуляризацию*, тот же самый метод, который используется в гребневой регрессии.

Для LogisticRegression и LinearSVC компромиссный параметр, который определяет степень регуляризации, называется C, и более высокие значения C соответствуют *меньшей* регуляризации. Другими словами, когда вы используете высокое значение параметра C, LogisticRegression и LinearSVC пытаются подогнать модель к обучающим данным как можно лучше, тогда как при низких значениях параметра C модели делают больший акцент на поиске вектора коэффициентов (w), близкого к нулю.

Существует еще одна интересная деталь, связанная с работой параметра C. Использование низких значений C приводит к тому, что алгоритмы пытаются подстроиться под «большинство» точек данных, тогда как использование более высоких значений C подчеркивает важность того, чтобы каждая отдельная точка данных была классифицирована правильно. Ниже приводится иллюстрация использования LinearSVC (рис. 5.2):

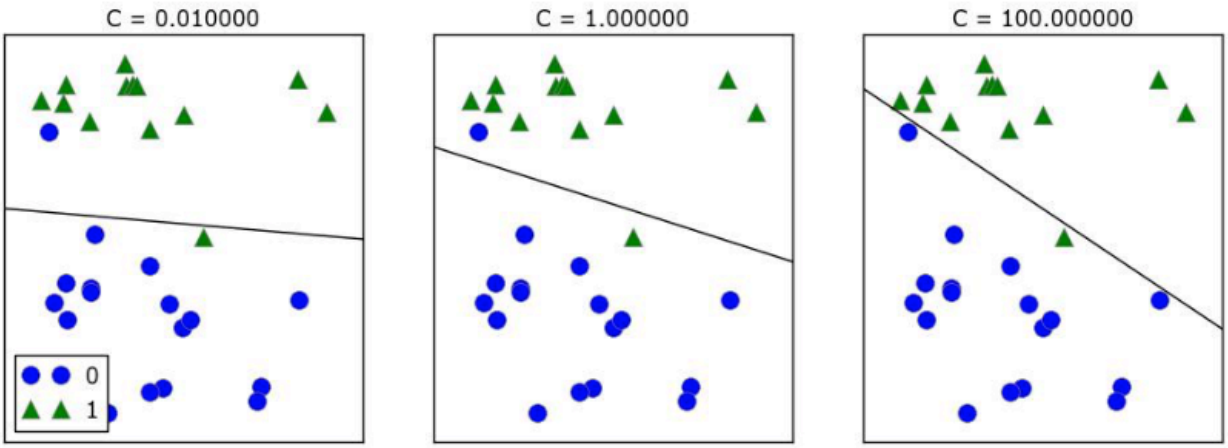

Рис. 5.2 Границы принятия решений линейного SVM с различными значениями C для набора данных forge

На графике слева показана модель с очень маленьким значением C, соответствующим большой степени регуляризации. Большая часть точек класса 0 находятся в нижней части графика, а большинство точек класса 1 находятся в верхней части. Сильно регуляризованная модель дает относительно горизонтальную линию, неправильно классифицируя две точки. На центральном графике значение C немного выше и модель в большей степени фокусируется на двух неправильно классифицированных примерах, наклоняя границу принятия решений. Наконец, на графике справа очень высокое значение C модели наклоняет границу принятия решений еще сильнее, теперь правильно классифицируя все точки класса 0. Одна из точек класса 1 по-прежнему неправильно классифицирована, поскольку невозможно правильно классифицировать все наблюдения этого набора данных с помощью прямой линии. Модель на графике справа старается изо всех сил правильно классифицировать все точки, но не может дать хорошего обобщения сразу для обоих классов. Другими словами, эта модель скорее всего переобучена.

Как и в случае с регрессией, линейные модели классификации могут показаться слишком строгими в условиях низкоразмерного пространства, предлагая границы принятия решений в виде прямых линий или плоскостей. Опять же, при наличии большого числа измерений линейные модели классификации приобретают высокую прогнозную силу и с увеличением числа признаков защита от переобучения становится все более важной.

Более подробно разберем работу LogisticRegression на наборе данных Breast Cancer:

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target, random_state=42)
logreg = LogisticRegression().fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.3f}".format(logreg.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(logreg.score(X_test, y_test)))
```

```
Правильность на обучающем наборе: 0.953
Правильность на тестовом наборе: 0.958
```

Значение по умолчанию C=1 обеспечивает неплохое качество модели, правильность на обучающем и тестовом наборах составляет 95%. Однако поскольку качество модели на обучающем и тестовом наборах примерно одинако, вполне вероятно, что мы недообучили модель. Давайте попробуем увеличить C, чтобы подогнать более гибкую модель:

```
logreg100 = LogisticRegression(C=100).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.3f}".format(logreg100.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(logreg100.score(X_test, y_test)))
```

```
Правильность на обучающем наборе: 0.972
Правильность на тестовом наборе: 0.965
```

Использование C=100 привело к более высокой правильности на обучающей выборке, а также немного увеличилась правильность на тестовой выборке, что подтверждает наш довод о том, что более сложная модель должна сработать лучше.

Кроме того, мы можем выяснить, что произойдет, если мы воспользуемся более регуляризованной моделью (установив C=0.01 вместо значения по умолчанию C=1):

```
logreg001 = LogisticRegression(C=0.01).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.3f}".format(logreg001.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(logreg001.score(X_test, y_test)))
```

```
Правильность на обучающем наборе: 0.934
Правильность на тестовом наборе: 0.930
```

Как и следовало ожидать, когда мы получили недообученную модель переместились влево по шкале, правильность как на обучающем, так и на тестовом наборах снизилась по сравнению с правильностью, которую мы получили, использовав параметры по умолчанию.

И, наконец, давайте посмотрим на коэффициенты логистической регрессии, полученные с использованием трех различных значений параметра регуляризации C (рис. 5.3):

```
plt.plot(logreg.coef_.T, 'o', label="C=1")
plt.plot(logreg100.coef_.T, '^', label="C=100")
plt.plot(logreg001.coef_.T, 'v', label="C=0.001")
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.hlines(0, 0, cancer.data.shape[1])
plt.ylim(-5, 5)
plt.xlabel("Индекс коэффициента")
plt.ylabel("Оценка коэффициента")
plt.legend()
```

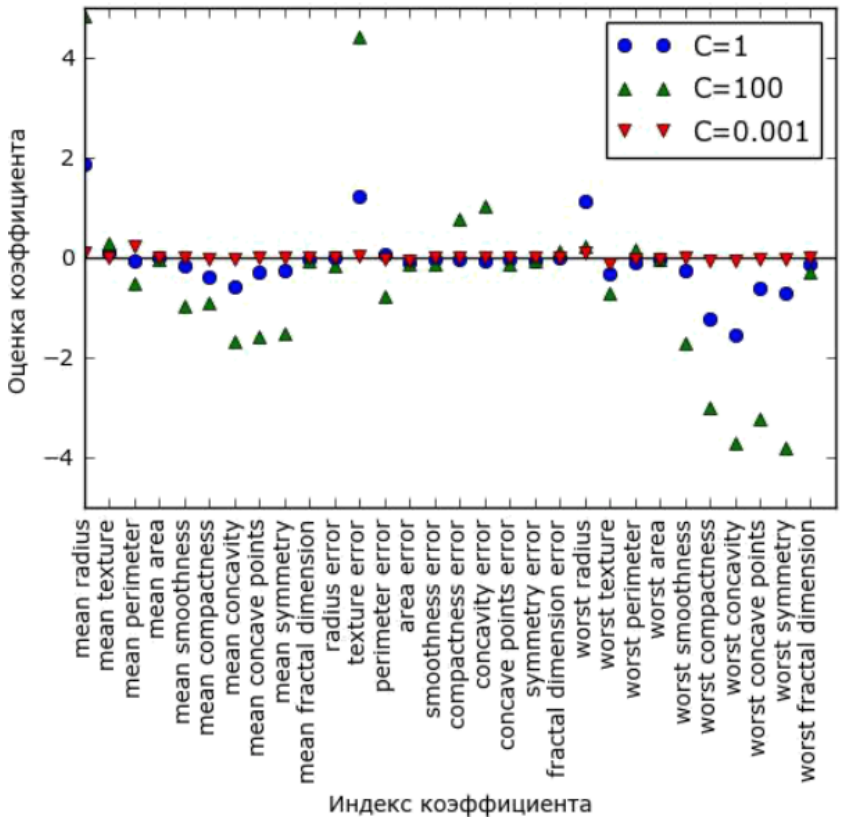

Рис. 5.3 Коэффициенты, полученные с помощью логистической регрессии с разными значениями C для набора данных Breast Cancer

Если мы хотим получить более интерпретабельную модель, нам может помочь L1 регуляризация, поскольку она ограничивает модель использованием лишь нескольких признаков. Ниже приводится график коэффициентами и оценками правильности для L1 регуляризации (рис. 5.4):

```
for C, marker in zip([0.001, 1, 100], ['o', '^', 'v']):
  lr_l1 = LogisticRegression(C=C, penalty="l1").fit(X_train, y_train)
  print("Правильность на обучении для логрегрессии l1 с C={:.3f}: {:.2f}".format( C,
lr_l1.score(X_train, y_train)))
  print("Правильность на тесте для логрегрессии l1 с C={:.3f}: {:.2f}".format( C, lr_l1.score(X_test,
y_test)))

plt.plot(lr_l1.coef_.T, marker, label="C={:.3f}".format(C))
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.hlines(0, 0, cancer.data.shape[1])
plt.xlabel("Индекс коэффициента")
plt.ylabel("Оценка коэффициента")

plt.ylim(-5, 5)
plt.legend(loc=3)
```

```
Правильность на обучении для логрегрессии l1 с C=0.001: 0.91
Правильность на тесте для логрегрессии l1 с C=0.001: 0.92
Правильность на обучении для логрегрессии l1 с C=1.000: 0.96
Правильность на тесте для логрегрессии l1 с C=1.000: 0.96
Правильность на обучении для логрегрессии l1 с C=100.000: 0.99
Правильность на тесте для логрегрессии l1 с C=100.000: 0.98
```

Видно, что существует много параллелей между линейными моделями для бинарной классификации и линейными моделями для регрессии. Как и в регрессии, основное различие между моделями – в параметре penalty, который влияет на регуляризацию и определяет, будет ли модель использовать все доступные признаки или выберет лишь подмножество признаков.

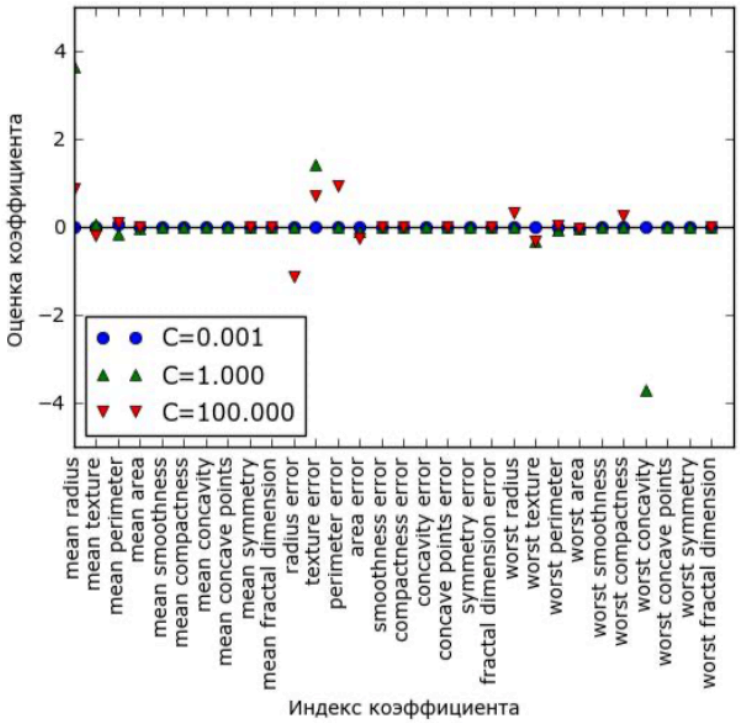

Рис. 5.4 Коэффициенты логистической регрессии с L1 штрафом для набора данных Breast Cancer (использовались различные значения C)

Код к лабораторной работе:

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

from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
X, y = mglearn.datasets.make_forge()
fig, axes = plt.subplots(1, 2, figsize=(10, 3))

for model, ax in zip([LinearSVC(), LogisticRegression()], axes):
    clf = model.fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=False, eps=0.5,
    ax=ax, alpha=.7)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{}".format(clf.__class__.__name__))
    ax.set_xlabel("Признак 0")
    ax.set_ylabel("Признак 1")
axes[0].legend()
plt.show()

mglearn.plots.plot_linear_svc_regularization()
plt.show()

from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)
logreg = LogisticRegression().fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.3f}".format(logreg.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(logreg.score(X_test, y_test)))

logreg100 = LogisticRegression(C=100).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.3f}".format(logreg100.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(logreg100.score(X_test, y_test)))

logreg001 = LogisticRegression(C=0.01).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.3f}".format(logreg001.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(logreg001.score(X_test, y_test)))

plt.plot(logreg.coef_.T, 'o', label="C=1")
plt.plot(logreg100.coef_.T, '^', label="C=100")
plt.plot(logreg001.coef_.T, 'v', label="C=0.001")
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.hlines(0, 0, cancer.data.shape[1])
plt.ylim(-5, 5)
plt.xlabel("Индекс коэффициента")
plt.ylabel("Оценка коэффициента")
```

```
plt.legend()
plt.show()

for C, marker in zip([0.001, 1, 100], ['o', '^', 'v']):
    lr_l1 = LogisticRegression(C=C, penalty="l2").fit(X_train, y_train)
    print("Правильность на обучении для логрегрессии l1 с C={:.3f}: {:.2f}".format( C,
lr_l1.score(X_train, y_train)))
    print("Правильность на тесте для логрегрессии l1 с C={:.3f}: {:.2f}".format( C,
lr_l1.score(X_test, y_test)))

plt.plot(lr_l1.coef_.T, marker, label="C={:.3f}".format(C))
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.hlines(0, 0, cancer.data.shape[1])
plt.xlabel("Индекс коэффициента")
plt.ylabel("Оценка коэффициента")
plt.ylim(-5, 5)
plt.legend(loc=3)
plt.show()
```