**Лабораторная работа №12**

Оценка неопределенностей для классификаторов

Еще одна полезная деталь интерфейса scikit-learn, о которой мы еще не говорили – это возможность вычислить оценки неопределенности прогнозов. Часто вас интересует не только класс, спрогнозированный моделью для определенной точки тестового набора, но и степень уверенности модели в правильности прогноза. В реальной практике различные виды ошибок приводят к очень разным результатам. Представьте себе медицинский тест для определения рака. Ложно положительный прогноз может привести к проведению дополнительных исследований, тогда как ложно отрицательный прогноз может привести пропуску серьезной болезни.

scikit-learn существует две различные функции, с помощью которых можно оценить неопределенность прогнозов: **decision\_function** и **predict\_proba**. Большая часть классификаторов (но не все) позволяет использовать по крайней мере одну из этих функций. Давайте применим эти две функции к синтетическому двумерному набору данных, построив классификатор **GradientBoostingClassifier**, который позволяет использовать как метод **decision\_function**, так и метод **predict\_proba**:

|  |
| --- |
| from sklearn.ensemble import GradientBoostingClassifier  from sklearn.datasets import make\_blobs, make\_circles  X, y = make\_circles(noise=0.25, factor=0.5, random\_state=1)  # мы переименовываем классы в «blue» и «red» для удобства  y\_named = np.array(["blue", "red"])[y]  # мы можем вызвать train\_test\_split с любым количеством массивов,  # все будут разбиты одинаковым образом  X\_train, X\_test, y\_train\_named, y\_test\_named, y\_train, y\_test = train\_test\_split(  X, y\_named, y, random\_state=0  )  # строим модель градиентного бустинга  gbrt = GradientBoostingClassifier(random\_state=0)  gbrt.fit(X\_train, y\_train\_named) |

Решающая функция

В бинарной классификации возвращаемое значение **decision\_function** имеет форму (n\_samples):

|  |
| --- |
| print("Форма массива X\_test: {}".format(X\_test.shape))  print("Форма решающей функции: {}".format(gbrt.decision\_function(X\_test).shape)) |

|  |
| --- |
| Форма массива X\_test: (25, 2)  Форма решающей функции: (25,) |

Возвращаемое значение представляет собой число с плавающей точкой для каждого примера:

|  |
| --- |
| # выведем несколько первых элементов решающей функции  print("Решающая функция:\n{}".format(gbrt.decision\_function(X\_test)[:6])) |

|  |
| --- |
| Решающая функция:  [ 4.136 -1.683 -3.951 -3.626 4.29 3.662] |

Значение показывает, насколько сильно модель уверена в том, что точка данных принадлежит «положительному» классу, в данном случае, классу 1. Положительное значение указывает на предпочтение в пользу позиционного класса, а отрицательное значение – на предпочтение в пользу «отрицательного» (другого) класса.

Мы можем судить о прогнозах, лишь взглянув на знак решающей функции.

|  |
| --- |
| print("Решающая функция с порогом отсечения:\n{}"  .format( gbrt.decision\_function(X\_test) > 0)  )  print("Прогнозы:\n{}".format(gbrt.predict(X\_test))) |

|  |
| --- |
| Решающая функция с порогом отсечения: [  True False False False True  True False True True True  False True True False True  False False False True True  True True True False False  ]  Прогнозы: [  'red' 'blue' 'blue' 'blue' 'red'  'red' 'blue' 'red' 'red' 'red'  'blue' 'red' 'red' 'blue' 'red'  'blue' 'blue' 'blue' 'red' 'red'  'red' 'red' 'red' 'blue' 'blue'  ] |

Для бинарной классификации «отрицательный» класс – это всегда первый элемент атрибута **classes\_**, а «положительный» класс – второй элемент атрибута **classes\_**. Таким образом, если вы хотите полностью просмотреть вывод метода **predict**, вам нужно воспользоваться атрибутом **classes\_**:

|  |
| --- |
| # переделаем булевы значения True/False в 0 и 1  greater\_zero = (gbrt.decision\_function(X\_test) > 0).astype(int)  # используем 0 и 1 в качестве индексов атрибута classes\_  pred = gbrt.classes\_[greater\_zero]  # pred идентичен выводу gbrt.predict  print("pred идентичен прогнозам: {}".format(np.all(pred == gbrt.predict(X\_test)))) |

|  |
| --- |
| pred идентичен прогнозам: True |

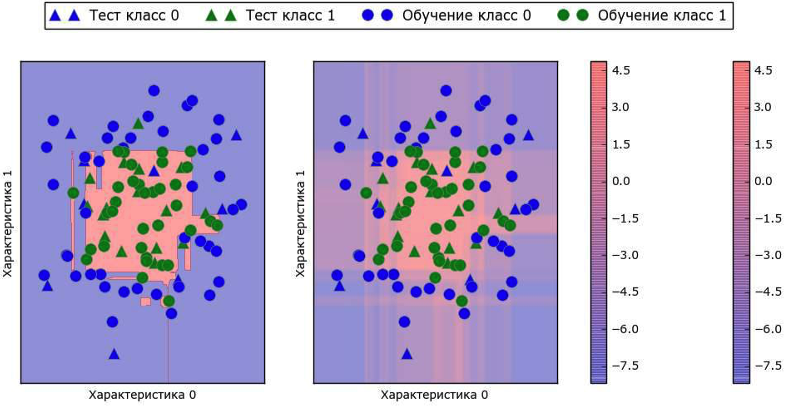
Диапазон значений **decision\_function** может быть произвольным и зависит от данных и параметров модели:

|  |
| --- |
| decision\_function = gbrt.decision\_function(X\_test)  print("Решающая функция минимум: {:.2f} максимум: {:.2f}"  .format( np.min(decision\_function), np.max(decision\_function))  ) |

|  |
| --- |
| Решающая функция минимум: -7.69 максимум: 4.29 |

Это произвольное масштабирование часто затрудняет интерпретацию вывода **decision\_function**. В следующем примере мы построим **decision\_function** для всех точек двумерной плоскости, используя цветовую кодировку и уже знакомую визуализацию решающей границы. Мы представим точки обучающего набора в виже кружков, а тестовые данные – в виде треугольников (рис. 12.1):

|  |
| --- |
| fig, axes = plt.subplots(1, 2, figsize=(13, 5))  mglearn.tools.plot\_2d\_separator(  gbrt,  X,  ax=axes[0],  alpha=.4,  fill=True,  cm=mglearn.cm2  )  scores\_image = mglearn.tools.plot\_2d\_scores(gbrt, X, ax=axes[1],  alpha=.4, cm=mglearn.ReBl)  for ax in axes:  mglearn.discrete\_scatter(X\_test[:, 0], X\_test[:, 1], y\_test,markers='^', ax=ax)  mglearn.discrete\_scatter(X\_train[:, 0], X\_train[:, 1], y\_train, markers='o', ax=ax)  ax.set\_xlabel("Характеристика 0")  ax.set\_ylabel("Характеристика 1")  cbar = plt.colorbar(scores\_image, ax=axes.tolist())  axes[0].legend(  ["Тест класс 0", "Тест класс 1", "Обучение класс 0", "Обучение класс 1"],  ncol=4,  loc=(.1, 1.1)  )  plt.show() |



**Рис. 12.1** Граница принятия решений (слева) и решающая функция (справа) модели градиентного бустинга, построенной на двумерном синтетическом наборе данных

Цветовая кодировка не только спрогнозированного результата, но степени определенности прогноза дает дополнительную информацию. Однако в этой визуализации трудно разглядеть границу между двумя классами.

Прогнозирование вероятностей

Вывод метода **predict\_proba** – это вероятность каждого класса и часто его легче понять, чем вывод метода **decision\_function**. Для бинарной классификации он имеет форму (n\_samples, 2):

|  |
| --- |
| print("Форма вероятностей: {}".format(gbrt.predict\_proba(X\_test).shape)) |

|  |
| --- |
| Форма вероятностей: (25, 2) |

Первый элемент строки – это оценка вероятности первого класса, а второй элемент строки – это оценка вероятности второго класса. Поскольку речь идет о вероятности, то значения в выводе **predict\_proba** всегда находятся в диапазоне между 0 и 1, а сумма значений для обоих классов всегда равна 1:

|  |
| --- |
| # выведем первые несколько элементов predict\_proba  print("Спрогнозированные вероятности:\n{}"  .format(gbrt.predict\_proba(X\_test[:6]))  ) |

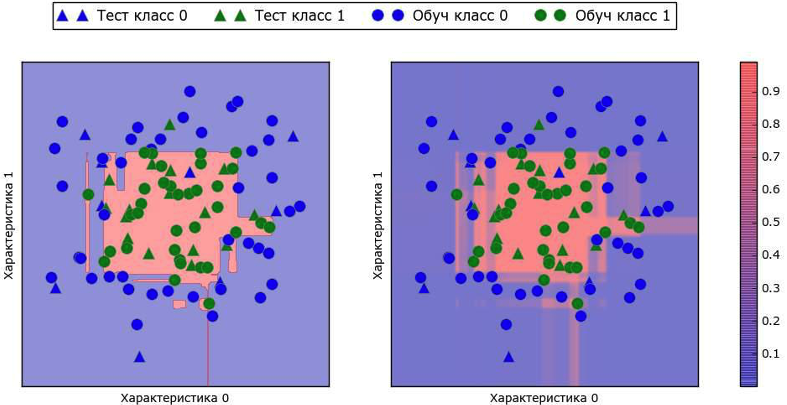
|  |
| --- |
| Спрогнозированные вероятности: [  [0.016 0.984]  [0.843 0.157]  [0.981 0.019]  [0.974 0.026]  [0.014 0.986]  [0.025 0.975]  ] |

Поскольку вероятности обоих классов в сумме дают 1, один из классов всегда будет иметь определенность, превышающую 50%. Этот класс и будет спрогнозирован.

В предыдущем выводе видно, что большинство точек отнесены к тому или иному классу с высокой долей определенности. Соответствие спрогнозированной неопределенности фактической зависит от модели и параметров. Для переобученной модели характерна высокая доля определенности прогнозов, даже если они и ошибочные. Модель с меньшей сложностью обычно характеризуется высокой долей неопределенности своих прогнозов. Модель называется калиброванной (**calibrated**), если вычисленная неопределенность соответствует фактической: в калиброванной модели прогноз, полученный с 70%-ной определенностью, будет правильным в 70% случаев.

В следующем примере (рис. 12.2) мы снова покажем границу принятия решения для набора данных, а также вероятности для класса 1:

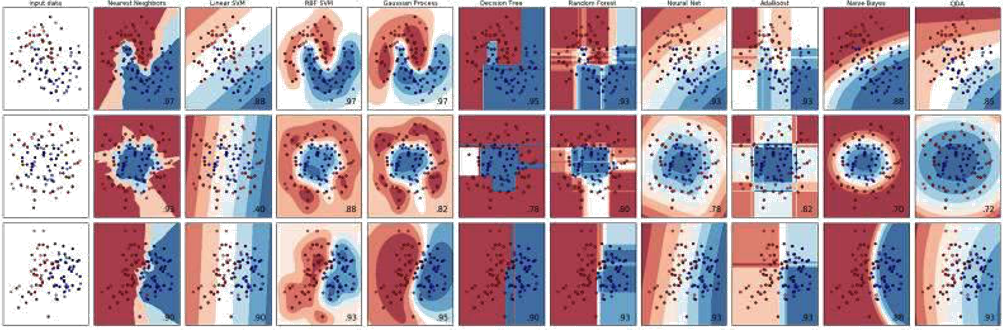
|  |
| --- |
| fig, axes = plt.subplots(1, 2, figsize=(13, 5))  mglearn.tools.plot\_2d\_separator(  gbrt,  X,  ax=axes[0],  alpha=.4,  fill=True,  cm=mglearn.cm2  )  scores\_image = mglearn.tools.plot\_2d\_scores(  gbrt,  X,  ax=axes[1],  alpha=.5,  cm=mglearn.ReBl,  function='predict\_proba'  )  for ax in axes:  mglearn.discrete\_scatter(X\_test[:, 0], X\_test[:, 1], y\_test,markers='^', ax=ax)  mglearn.discrete\_scatter(  X\_train[:, 0], X\_train[:, 1], y\_train, markers='o', ax=ax  )  ax.set\_xlabel("Характеристика 0")  ax.set\_ylabel("Характеристика 1")  cbar = plt.colorbar(scores\_image, ax=axes.tolist())  axes[0].legend(  ["Тест класс 0", "Тест класс 1", "Обуч класс 0", "Обуч класс 1"],  ncol=4,  loc=(.1, 1.1)  )  plt.show() |



**Рис. 12.2** Граница принятия решений (слева) спрогнозированные вероятности для модели градиентного бустинга, показанной на рис. 12.1

Границы на этом рисунке определены гораздо более четко, а небольшие участки неопределенности отчетливо видны.

На сайте scikit-learn дается сравнение различных моделей и визуализации оценок неопределенности для этих моделей. Мы воспроизвели их на рис. 12.3 и рекомендуем ознакомиться с ними.



**Рис. 12.3** Сравнение нескольких классификаторов scikit-learn, построенных на синтетических наборах данных (изображение предоставлено http://scikit-learn.org)

Неопределенность в мультиклассовой классификации

До сих пор мы говорили только об оценках неопределенности в бинарной классификации. Однако методы **decision\_function** и **predict\_proba** также можно применять в мультиклассовой классификации. Давайте применим их к набору данных Iris, который представляет собой пример 3-классовой классификации:

|  |
| --- |
| from sklearn.datasets import load\_iris  iris = load\_iris()  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  iris.data,  iris.target,  random\_state=42  )  gbrt = GradientBoostingClassifier(learning\_rate=0.01, random\_state=0)  gbrt.fit(X\_train, y\_train)  print("Форма решающей функции: {}".format(gbrt.decision\_function(X\_test).shape))  print("Решающая функция:\n{}".format(gbrt.decision\_function(X\_test)[:6, :])) |

|  |
| --- |
| Форма решающей функции: (38, 3)  Решающая функция: [  [-0.529 1.466 -0.504 ]  [ 1.512 -0.496 -0.503 ]  [-0.524 -0.468 1.52 ]  [-0.529 1.466 -0.504 ]  [-0.531 1.282 0.215 ]  [ 1.512 -0.496 -0.503 ]  ] |

В мультиклассовой классификации **decision\_function** имеет форму (n\_samples, n\_classes) и каждый столбец показывает «оценку определенности» для каждого класса, где высокая оценка означает большую вероятность данного класса, а низкая оценка означает меньшую вероятность этого класса. Вы можете получить прогнозы, исходя из этих оценок, с помощью функции np.argmax. Она возвращает индекс максимального элемента массива для каждой точки данных:

|  |
| --- |
| print("Argmax решающей функции:\n{}"  .format(np.argmax(gbrt.decision\_function(X\_test), axis=1))  )  print("Прогнозы:\n{}"  .format(gbrt.predict(X\_test))  ) |

|  |
| --- |
| Argmax решающей функции:  [10211012112000012112020222220000100210]  Прогнозы:  [10211012112000012112020222220000100210] |

Вывод **predict\_proba** имеет точно такую же форму (n\_samples, n\_classes). И снова вероятности возможных классов для каждой точки данных дают в сумме 1:

|  |
| --- |
| print("Спрогнозированные вероятности:\n{}"  .format(gbrt.predict\_proba(X\_test)[:6])  )  print("Суммы: {}"  .format(gbrt.predict\_proba(X\_test)[:6].sum(axis=1))  ) |

|  |
| --- |
| Спрогнозированные вероятности: [  [ 0.107 0.784 0.109]  [ 0.789 0.106 0.105]  [ 0.102 0.108 0.789]  [ 0.107 0.784 0.109]  [ 0.108 0.663 0.228]  [ 0.789 0.106 0.105]  ]  Суммы:  [ 1. 1. 1. 1. 1. 1.] |

Мы вновь можем получить прогнозы, вычислив argmax для **predict\_proba**:

|  |
| --- |
| print("Argmax спрогнозированных вероятностей:\n{}"  .format( np.argmax(gbrt.predict\_proba(X\_test), axis=1))  )  print("Прогнозы:\n{}"  .format(gbrt.predict(X\_test))  ) |

|  |
| --- |
| Argmax спрогнозированных вероятностей:  [10211012112000012112020222220000100210]  Прогнозы:  [10211012112000012112020222220000100210] |

Подводя итог, отметим, что **predict\_proba** и **decision\_function** всегда имеют форму (n\_samples, n\_classes), за исключением **decision\_function** в случае бинарной классификации. В бинарной классификации **decision\_function** имеет только один столбец, соответствующий «положительному» классу classes\_[1].

Для количества столбцов, равного **n\_classes**, вы можете получить прогноз, вычислив **argmax** по столбцам. Однако будьте осторожны, если ваши классы – строки или вы используете целые числа, которые не являются последовательными и начинаются не с 0. Если вы хотите сравнить результаты, полученные с помощью **predict**, с результатами **decision\_function** или **predict\_proba**, убедитесь, что используете атрибут **classes\_** для получения фактических названий классов:

|  |
| --- |
| from sklearn.linear\_model import LogisticRegression  logreg = LogisticRegression()  named\_target = iris.target\_names[y\_train]  logreg.fit(X\_train, named\_target)  print("уникальные классы в обучающем наборе: {}"  .format(logreg.classes\_)  )  print("прогнозы: {}"  .format(logreg.predict(X\_test)[:10])  )  argmax\_dec\_func = np.argmax(logreg.decision\_function(X\_test), axis=1)  print("argmax решающей функции: {}"  .format(argmax\_dec\_func[:10])  )  print("argmax объединенный с классами\_: {}"  .format(logreg.classes\_[argmax\_dec\_func][:10])  ) |

|  |
| --- |
| уникальные классы в обучающем наборе:  ['setosa' 'versicolor' 'virginica']  прогнозы: [  'versicolor'  'setosa'  'virginica'  'versicolor'  'versicolor'  'setosa'  'versicolor'  'virginica'  'versicolor'  'versicolor'  ]  argmax решающей функции:  [1 0 2 1 1 0 1 2 1 1]  argmax объединенный с классами\_: [  'versicolor'  'setosa'  'virginica'  'versicolor'  'versicolor'  'setosa'  'versicolor'  'virginica'  'versicolor'  'versicolor'  ] |

Выводы и перспективы

Мы начали эту главу с обсуждения такого понятия, как сложность модели, а затем рассказали об обобщающей способности (generalization), то есть о построении такой модели, которая может хорошо работать на новых, ранее неизвестных данных. Это привело нас к понятиям «недообучение», когда модель не может описать изменчивость обучающих данных, и «переобучение», когда модель слишком много внимания уделяет обучающим данным и не способна хорошо обобщить новые данные.

Затем мы рассмотрели широкий спектр моделей машинного обучения для классификации и регрессии, их преимущества и недостатки, настройки сложности для каждой модели. Мы увидели, что для достижения хорошего качества работы во многих алгоритмах важное значение имеет установка правильных параметров. Кроме того, некоторые алгоритмы чувствительны к типу входных данных, и, в частности, к тому, как масштабированы признаки. Поэтому слепое применение алгоритма к данным без понимания исходных предположений модели и принципов работы параметров редко приводит к построению точной модели.

Эта глава содержит много информации об алгоритмах, но вам необязательно помнить все эти детали, чтобы понимать содержание следующих глав. Тем не менее некоторая информация о моделях, упомянутых здесь, и контексте использования этих моделей, имеет важное значение для успешного применения машинного обучения на практике. Ниже дается краткий обзор случаев использования той или иной модели:

**Ближайшие соседи**

Подходит для небольших наборов данных, хорош в качестве базовой модели, прост в объяснении.

**Линейные модели**

Считается первым алгоритмом, который нужно попробовать, хорош для очень больших наборов данных, подходит для данных с очень высокой размерностью.

**Наивный байесовский классификатор**

Подходит только для классификации. Работает даже быстрее, чем линейные модели, хорош для очень больших наборов данных и высокоразмерных данных. Часто менее точен, чем линейные модели.

**Деревья решений**

Очень быстрый метод, не нужно масштабировать данные, результаты можно визуализировать и легко объяснить.

**Случайные леса**

Почти всегда работают лучше, чем одно дерево решений, очень устойчивый и мощный метод. Не нужно масштабировать данные. Плохо работает с данными очень высокой размерности и разреженными данными.

**Градиентный бустинг деревьев решений**

Как правило, немного более точен, чем случайный лес. В отличие от случайного леса медленнее обучается, но быстрее предсказывает и требует меньше памяти. По сравнению со случайным лесом требует настройки большего числа параметров.

**Машины опорных векторов**

Мощный метод для работы с наборами данных среднего размера и признаками, измеренными в едином масштабе. Требует масштабирования данных, чувствителен к изменению параметров.

**Нейронные сети**

Можно построить очень сложные модели, особенно для больших наборов данных. Чувствительны к масштабированию данных и выбору параметров. Большим моделям требуется много времени для обучения.

При работе с новым набором данных лучше начать с простой модели, например, с линейной модели, наивного байесовского классификатора или классификатора ближайших соседей, и посмотреть, как далеко можно продвинуться с точки зрения качества модели. Лучше изучив данные, вы можете выбрать алгоритм, который может строить более сложные модели, например, случайный лес, градиентный бустинг деревьев решений, SVM или нейронную сеть.

Теперь у вас уже есть некоторое представление о том, как применять, настраивать и анализировать модели, которые мы здесь рассмотрели. В этой главе мы сосредоточились на бинарном классификации, поскольку ее, как правило, легче всего интерпретировать. Большинство представленных алгоритмов могут решать задачи регрессии и классификации вариантов, при этом все алгоритмы классификации поддерживают как бинарную, так и мультиклассовую классификацию. Попробуйте применить любой из этих алгоритмов к наборам данных, включенным в scikit-learn, например, к наборам для регрессии boston\_housing или diabetes, или к набору digits для мультиклассовой классификации. Экспериментирование с алгоритмами на различных наборах данных позволит вам лучше понять, насколько быстро обучаются различные алгоритмы, насколько легко анализировать построенные с их помощью модели и насколько эти алгоритмы чувствительны к типу данных.

*Код к лабораторной работе:*

|  |
| --- |
| import mglearn  import sklearn  import matplotlib.pyplot as plt  import numpy as np  import graphviz  from IPython.display import display  display(mglearn.plots.plot\_single\_hidden\_layer\_graph())  line = np.linspace(-3, 3, 100)  plt.plot(line, np.tanh(line), label="tanh")  plt.plot(line, np.maximum(line, 0), label="relu")  plt.legend(loc="best")  plt.xlabel("x")  plt.ylabel("relu(x), tanh(x)")  plt.show()  mglearn.plots.plot\_two\_hidden\_layer\_graph()  plt.show()  from sklearn.model\_selection import train\_test\_split  from sklearn.neural\_network import MLPClassifier  from sklearn.datasets import make\_moons  X, y = make\_moons(n\_samples=100, noise=0.25, random\_state=3)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, random\_state=42)  mlp = MLPClassifier(solver='lbfgs', random\_state=0).fit(X\_train, y\_train)  mglearn.plots.plot\_2d\_separator(mlp, X\_train, fill=True, alpha=.3)  mglearn.discrete\_scatter(X\_train[:, 0], X\_train[:, 1], y\_train)  plt.xlabel("Признак 0")  plt.ylabel("Признак 1")  plt.show()  mlp = MLPClassifier(solver='lbfgs', random\_state=0, hidden\_layer\_sizes=[10])  mlp.fit(X\_train, y\_train)  mglearn.plots.plot\_2d\_separator(mlp, X\_train, fill=True, alpha=.3)  mglearn.discrete\_scatter(X\_train[:, 0], X\_train[:, 1], y\_train)  plt.xlabel("Признак 0")  plt.ylabel("Признак 1")  plt.show()  mlp = MLPClassifier(solver='lbfgs', random\_state=0, hidden\_layer\_sizes=[10, 10])  mlp.fit(X\_train, y\_train)  mglearn.plots.plot\_2d\_separator(mlp, X\_train, fill=True, alpha=.3)  mglearn.discrete\_scatter(X\_train[:, 0], X\_train[:, 1], y\_train)  plt.xlabel("Признак 0")  plt.ylabel("Признак 1")  plt.show()  mlp = MLPClassifier(solver='lbfgs', activation='tanh',  random\_state=0, hidden\_layer\_sizes=[10, 10])  mlp.fit(X\_train, y\_train)  mglearn.plots.plot\_2d\_separator(mlp, X\_train, fill=True, alpha=.3)  mglearn.discrete\_scatter(X\_train[:, 0], X\_train[:, 1], y\_train)  plt.xlabel("Признак 0")  plt.ylabel("Признак 1")  plt.show()  fig, axes = plt.subplots(2, 4, figsize=(20, 8))  for axx, n\_hidden\_nodes in zip(axes, [10, 100]):  for ax, alpha in zip(axx, [0.0001, 0.01, 0.1, 1]):  mlp = MLPClassifier(solver='lbfgs', random\_state=0,  hidden\_layer\_sizes=[n\_hidden\_nodes, n\_hidden\_nodes], alpha=alpha)  mlp.fit(X\_train, y\_train)  mglearn.plots.plot\_2d\_separator(mlp, X\_train, fill=True, alpha=.3, ax=ax)  mglearn.discrete\_scatter(X\_train[:, 0], X\_train[:, 1], y\_train, ax=ax)  ax.set\_title("n\_hidden=[{}, {}]\nalpha={:.4f}".format(  n\_hidden\_nodes, n\_hidden\_nodes, alpha))  plt.show()  fig, axes = plt.subplots(2, 4, figsize=(20, 8))  for i, ax in enumerate(axes.ravel()):  mlp = MLPClassifier(solver='lbfgs', random\_state=i, hidden\_layer\_sizes=[100, 100])  mlp.fit(X\_train, y\_train)  mglearn.plots.plot\_2d\_separator(mlp, X\_train, fill=True, alpha=.3, ax=ax)  mglearn.discrete\_scatter(X\_train[:, 0], X\_train[:, 1], y\_train, ax=ax)  plt.show()  from sklearn.datasets import load\_breast\_cancer  cancer = load\_breast\_cancer()  print("Максимальные значения характеристик:\n{}".format(cancer.data.max(axis=0)))  X\_train, X\_test, y\_train, y\_test = train\_test\_split( cancer.data, cancer.target, random\_state=0)  mlp = MLPClassifier(random\_state=42)  mlp.fit(X\_train, y\_train)  print("Правильность на обучающем наборе: {:.2f}".format(mlp.score(X\_train, y\_train)))  print("Правильности на тестовом наборе: {:.2f}".format(mlp.score(X\_test, y\_test)))  min\_on\_training = X\_train.min(axis=0)  range\_on\_training = (X\_train - min\_on\_training).max(axis=0)  X\_train\_scaled = (X\_train - min\_on\_training) / range\_on\_training  mean\_on\_train = X\_train.mean(axis=0)  std\_on\_train = X\_train.std(axis=0)  X\_test\_scaled = (X\_test - mean\_on\_train) / std\_on\_train  mlp = MLPClassifier(random\_state=0)  mlp.fit(X\_train\_scaled, y\_train)  print("Правильность на обучающем наборе: {:.3f}".format( mlp.score(X\_train\_scaled, y\_train)))  print("Правильность на тестовом наборе: {:.3f}".format(mlp.score(X\_test\_scaled, y\_test)))  mlp = MLPClassifier(max\_iter=1000, random\_state=0)  mlp.fit(X\_train\_scaled, y\_train)  print("Правильность на обучающем наборе: {:.3f}".format( mlp.score(X\_train\_scaled, y\_train)))  print("Правильность на тестовом наборе: {:.3f}".format(mlp.score(X\_test\_scaled, y\_test)))  mlp = MLPClassifier(max\_iter=1000, alpha=1, random\_state=0)  mlp.fit(X\_train\_scaled, y\_train)  print("Правильность на обучающем наборе: {:.3f}".format( mlp.score(X\_train\_scaled, y\_train)))  print("Правильность на тестовом наборе: {:.3f}".format(mlp.score(X\_test\_scaled, y\_test)))  plt.figure(figsize=(20, 5))  plt.imshow(mlp.coefs\_[0], interpolation='none', cmap='viridis')  plt.yticks(range(30), cancer.feature\_names)  plt.xlabel("Столбцы матрицы весов")  plt.ylabel("Входная характеристика")  plt.colorbar()  plt.show() |