Лабораторная работа №6

Оценка и улучшение качества модели.

Contents

[Ход работы: 1](#_Toc150725819)

[Перекрестная проверка 1](#_Toc150725820)

[Перекрестная проверка в scikit-learn 2](#_Toc150725821)

[Преимущества перекрестной проверки 3](#_Toc150725822)

[Стратифицированная k-блочная перекрестная проверка и другие стратегии 5](#_Toc150725823)

[Больше контроля над перекрестной проверкой 7](#_Toc150725824)

[Перекрестная проверка с исключением по одному 8](#_Toc150725825)

[Перекрестная проверка со случайными перестановками при разбиении 9](#_Toc150725826)

[Перекрестная проверка с использованием групп 10](#_Toc150725827)

[Решетчатый поиск 12](#_Toc150725828)

[Простой решетчатый поиск 12](#_Toc150725829)

[Опасность переобучения параметров и проверочный набор данных 13](#_Toc150725830)

[Решетчатый поиск с перекрестной проверкой 15](#_Toc150725831)

[Анализ результатов перекрестной проверки 20](#_Toc150725832)

[Экономичный решетчатый поиск 22](#_Toc150725833)

[Применение различных стратегий перекрестной проверки с помощью решетчатого поиска 23](#_Toc150725834)

[Вложенная перекрестная проверка 23](#_Toc150725835)

[Метрики качества моделей и их вычисление 25](#_Toc150725836)

[Метрики бинарной классификации 26](#_Toc150725837)

[Несбалансированный набор ошибок 27](#_Toc150725838)

[Матрица ошибок 29](#_Toc150725839)

[Связь с правильностью 33](#_Toc150725840)

[Точность, полнота и F-мера 33](#_Toc150725841)

[Кривые точности - полноты и ROC-кривые 35](#_Toc150725842)

[Метрики для мультиклассовой классификации 40](#_Toc150725843)

[Метрики регрессии 43](#_Toc150725844)

[Использование метрик оценки для отбора модели 43](#_Toc150725845)

[ЗАДАНИЕ: 45](#_Toc150725846)

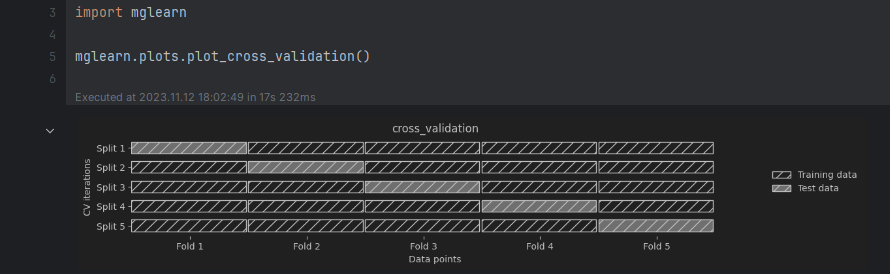
**Цель работы**: познакомиться с методом перекрестной проверки модели – надежным способом обобщающей способности для классификации и регрессии; рассмотреть метод решетчатого поиска (grid search), предназначенный для корректировки параметров модели для получения наилучшей обобщающей способности моделей контролируемого машинного обучения.

# Ход работы:

## Перекрестная проверка

**Перекрестная проверка** представляет собой статистический метод оценки обобщающей способности, который является более устойчивым и основательным, чем разбиение данных на обучающий и тестовый наборы.

В перекрестной проверке данные разбиваются несколько раз и строится несколько моделей. Наиболее часто используемый вариант перекрестной проверки – k-блочная кросс-проверка (k-fold cross-validation), в которой k – это задаваемое пользователем число, как правило, 5 или 10. При выполнении пятиблочной перекрестной проверки данные сначала разбиваются на пять частей (примерно) одинакового размера, называемых блоками (folds) складками. Затем строится последовательность моделей. Первая модель обучается, используя блок 1 в качестве тестового набора, а остальные блоки (2-5) выполняют роль обучающего набора. Модель строится на основе данных, расположенных в блоках 2-5, а затем на данных блока 1 оценивается ее правильность. Затем происходит обучение второй модели, на этот раз в качестве тестового набора используется блок 2, а данные в блоках 1, 3, 4, и 5 служат обучающим набором. Этот процесс повторяется для блоков 3, 4 и 5, выполняющих роль тестовых наборов. Для каждого из этих пяти разбиений (splits) данных на обучающий и тестовый наборы мы вычисляем правильность. В итоге мы зафиксировали пять значений правильности. Процесс показан на рис. 1:

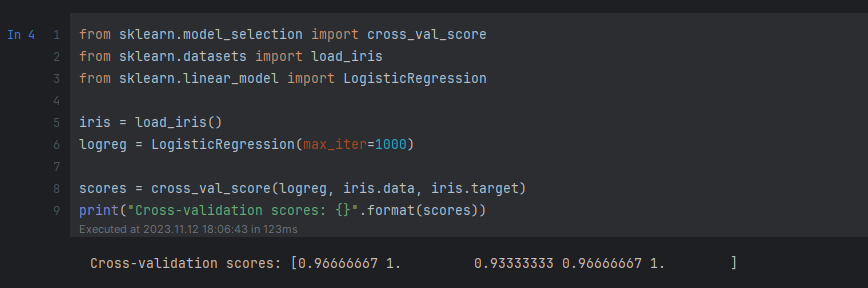


*Рис. 1 Разбиение данных в пятиблочной перекрестной проверке*

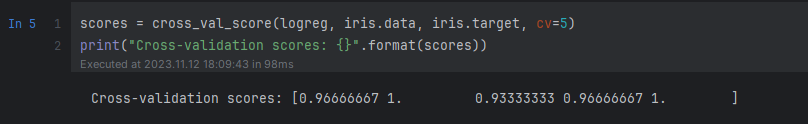
Как правило, первая пятая часть данных формирует первый блок, вторая пятая часть данных формирует второй блок и так далее.

### Перекрестная проверка в scikit-learn

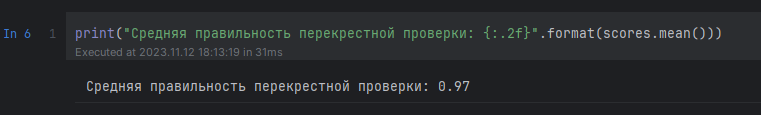
В scikit-learn перекрестная проверка реализована с помощью функции **cross\_val\_score** модуля **model\_selection**. Аргументами функции **cross\_val\_score** являются оцениваемая модель, обучающие данные и фактические метки. Давайте оценим качество модели **LogisticRegression** на наборе данных iris:



По умолчанию **cross\_val\_score** выполняет трехблочную перекрестную проверку, возвращая три значения правильности. Мы можем изменить количество блоков, задав другое значение параметра cv:



Наиболее распространенный способ подытожить правильность, вычисленную в ходе перекрестной проверки, – это вычисление среднего значения:



Используя усредненное значение правильности для перекрестной проверки, мы можем сделать вывод, что средняя правильность модели составит примерно 97%. Взглянув на все пять значений правильности, полученных в ходе пятиблочной перекрестной проверки, можно еще сделать вывод о том, что существует относительно высокий разброс значений правильности, вычисленных для блоков, от 100% до 93%. Подобный результат может означать, что модель сильно зависит от конкретных блоков, использованных для обучения, а также это может быть обусловлено небольшим размером набора данных.

### Преимущества перекрестной проверки

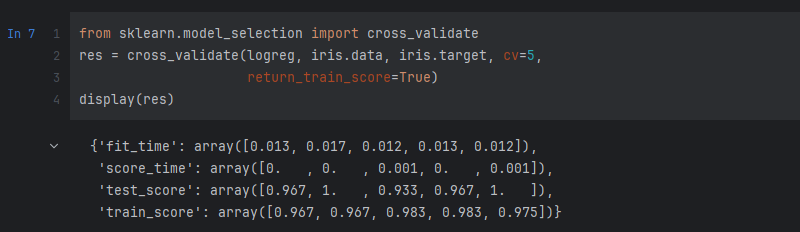
По сравнению с однократным разбиением данных на обучающий и тестовый наборы использование перекрестной проверки имеет несколько преимуществ. Во-первых, вспомним что train\_test\_split выполняет случайное разбиение данных. Представьте себе, что при выполнении случайного разбиения данных нам «повезло», и все трудно классифицируемые примеры в конечном итоге попали в обучающий набор. В этом случае в тестовый набор попадут только «легкие» примеры, и правильность на тестовом наборе будет неправдоподобно высокой. И, наоборот, если нам «не повезло», все трудно классифицируемые примеры попадают в тестовый набор и поэтому мы получаем неправдоподобно низкую правильность. Однако при использовании перекрестной проверки на каждой итерации в тестовый набор, использующийся для проверки модели, попадают разные примеры. Таким образом, модель должна хорошо обобщать все примеры в наборе данных, чтобы все значения правильности (или их среднее) были высокими.

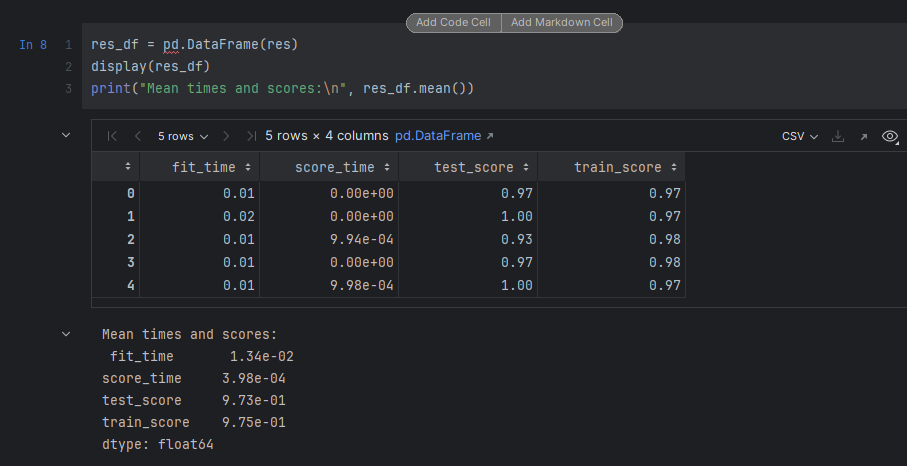
Кроме того, наличие нескольких разбиений дает определенную информацию о том, насколько наша модель чувствительна к выбору обучающего набора данных. Для набора данных iris мы увидели разброс значений правильности от 90% до 100%. Это довольно широкий диапазон значений, и он позволяет нам судить о том, как модель будет работать в худшем и лучшем случае, когда мы применим ее к новым данным.

Еще одно преимущество перекрестной проверки по сравнению с однократным разбиением данных заключается в том, что мы используем наши данные более эффективно. Применяя train\_test\_split, мы обычно используем 75% данных для обучения и 25% данных для оценки качества. Применяя пятиблочную перекрестную проверку, на каждой итерации для подгонки модели мы можем использовать 4/5 данных (80%). При использовании 10-блочной перекрестной проверки мы можем использовать для подгонки модели 9/10 данных (90%). Больший объем данных, как правило, приводит к построению более точных моделей.

**Основной недостаток перекрестной проверки** – увеличение стоимости вычислений. Поскольку теперь мы обучаем k моделей вместо одной модели, перекрестная проверка будет выполняться примерно в k раз медленнее, чем однократное разбиение данных.

**Пример для датасета iris**

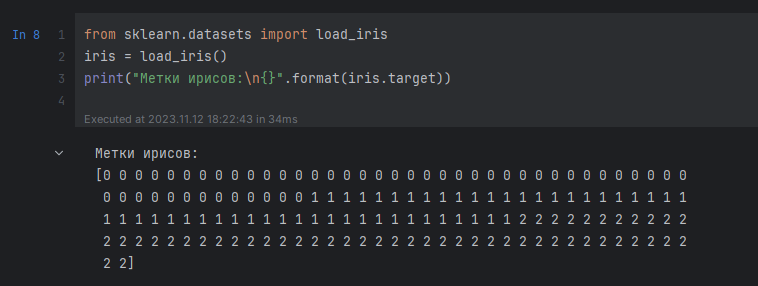


**

*Важно помнить, что кросс-валидация не является способом построения модели, которую можно применить к новым данным. Перекрестная проверка не возвращает модель. При вызове cross\_val\_score строится несколько внутренних моделей, однако цель перекрестной проверки заключается только в том, чтобы оценить обобщающую способность данного алгоритма, обучив на определенном наборе данных.*

## Стратифицированная k-блочная перекрестная проверка и другие стратегии

Описанное выше разбиение данных на k блоков, начиная с первого k-го блока, не всегда является хорошей идеей. Для примера давайте посмотрим на набор данных iris:



Как видно, первая треть данных – это класс 0, вторая треть – класс 1, а последняя треть – класс 2.

Допустим, что сделает с этим набором данных трехблочная перекрестная проверка. Первый блок будет состоять из примеров, относящихся только к классу 0, поэтому при первом разбиении данных тестовый набор станет полностью классом 0, а обучающий набор будет содержать примеры, относящиеся только к классам 1 и 2. Поскольку классы в обучающем и тестовом наборах будут разными во всех трех разбиениях, правильность трехблочной перекрестной проверки для этого набора данных будет равна нулю. Данный сценарий не является оптимальным, поскольку для набора данных iris мы можем получить правильность существенно выше 0%.

Поскольку обычная k-блочная стратегия в данном случае терпит неудачу, вместо нее библиотека scikit-learn предлагает использовать для классификации стратифицированную k-блочную перекрестную проверку (stratified k-fold cross-validation). В стратифицированной перекрестной проверке мы разбиваем данные таким образом, чтобы пропорции классов в каждом блоке в точности соответствовали пропорциям классов в наборе данных, как это показано на рис. 2:

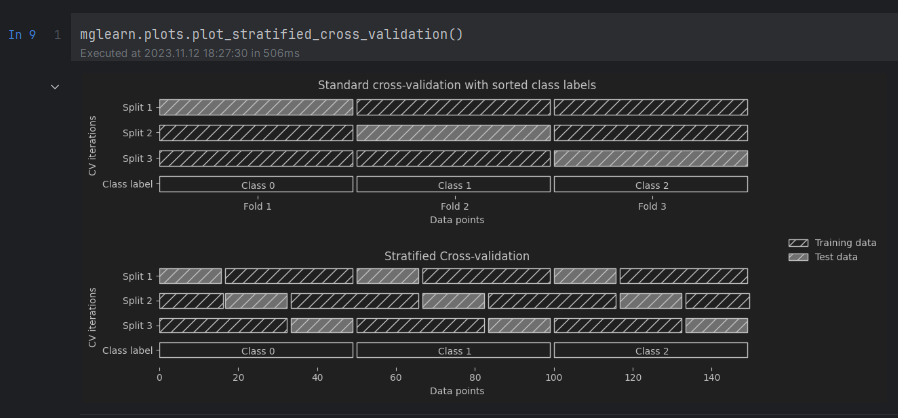


Рис. 2 Сравнение стандартной перекрестной проверки и стратифицированной перекрестной проверки, когда данные упорядочены по меткам классов

Например, если 90% примеров относятся к классу А, а 10% примеров – к классу В, то стратифицированная перекрестная проверка выполняется так, чтобы в каждом блоке 90% примеров принадлежали к классу А, а 10% примеров – к классу B.

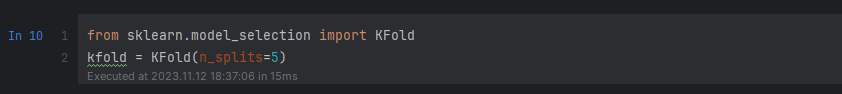
Использование для оценки классификатора стратифицированной k-блочной перекрестной проверки вместо обычной k-блочной перекрестной является хорошей идеей, поскольку позволяет получить более надежные оценки обобщающей способности. В ситуации, когда лишь 10% примеров принадлежат к классу В, использование стандартной k-блочной перекрестной проверки может привести к тому, что один из блоков будет полностью состоять из примеров, относящихся к классу А. Использование этого блока в качестве тестового набора не даст особой информации о качестве работы классификатора.

**Для регрессии в scikit-learn** по умолчанию используется стандартная k-блочная кросс-проверка. Можно было бы еще попытаться создать блоки, представляющие различные значения количественной зависимой переменной, но данный метод не является общераспространенной стратегией и был бы неожиданностью для большинства пользователей.

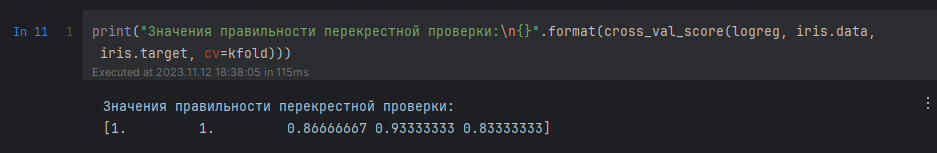
### Больше контроля над перекрестной проверкой

Можно настроить количество блоков, используемое в **cross\_val\_score**, с помощью параметра **cv**. Однако scikit-learn позволяет значительно точнее настроить процесс перекрестной проверки, используя в качестве параметра **cv** генератор разбиений перекрестной проверки (**cross-validation splitter**).

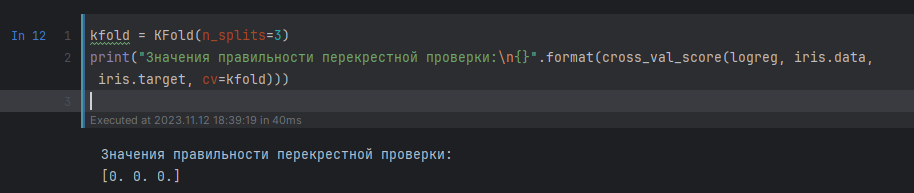
В большинстве случаев значения параметров, выставленные по умолчанию для k-блочной перекрестной проверки в случае регрессии и стратифицированной k-блочной проверки в случае классификации дают хорошие результаты, однако бывают ситуации, когда вы, возможно, захотите использовать другую стратегию. Допустим, мы хотим применить k-блочную перекрестную проверку к классификационному набору данных, чтобы воспроизвести чьи-то результаты. Для этого мы должны сначала импортировать класс **KFold** из модуля **model\_selection** и создать его экземпляр, задав нужное количество блоков:



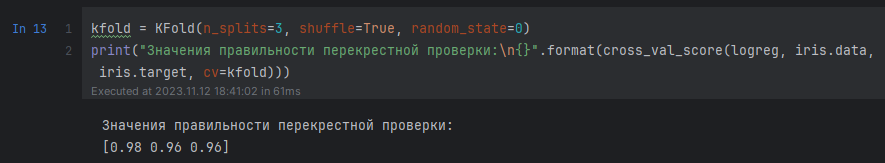
Затем мы можем передать генератор разбиений kfold в качестве параметра cv в функцию cross\_val\_score.



Таким образом, мы можем убедиться, что использование трехблочной (нестратифицированной) перекрестной проверки для набора данных iris действительно является очень плохой идеей:



Вспомним, что в наборе данных iris каждый блок соответствует одному классу и поэтому, применив нестратифицированную перекрестную проверку, мы ничего не сможем узнать о правильности модели. Еще один способ решения этой проблемы состоит в том, чтобы вместо стратификации перемешать данные и тем самым нарушить порядок сортировки примеров, определяемый их метками. Мы можем сделать это, передав генератору KFold параметр shuffle=True. Если мы перемешиваем данные, нам необходимо зафиксировать random\_state, чтобы воспроизвести результат перемешивания. В противном случае каждый прогон cross\_val\_score будет давать разный результат, поскольку каждый раз используется разное разбиение (это не является проблемой, но может привести к неожиданным результатам). Перемешивание данных перед их разбиением дает гораздо лучший результат:



### Перекрестная проверка с исключением по одному

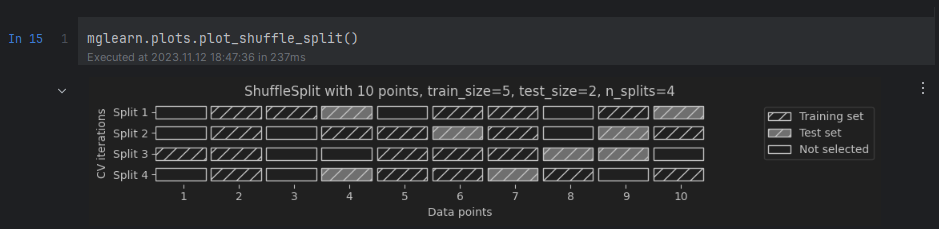
Еще один часто используемый метод перекрестной проверки – исключение по одному (leave-one-out).

Перекрестную проверку с исключением по одному можно представить в виде k-блочной перекрестной проверки, в которой каждый блок представляет собой отдельный пример. По каждому разбиению вы выбираете одну точку данных в качестве тестового набора. Этот вид проверки может занимать очень много времени, особенно при работе с большими наборами данных, однако иногда позволяет получить более точные оценки на небольших наборах данных:



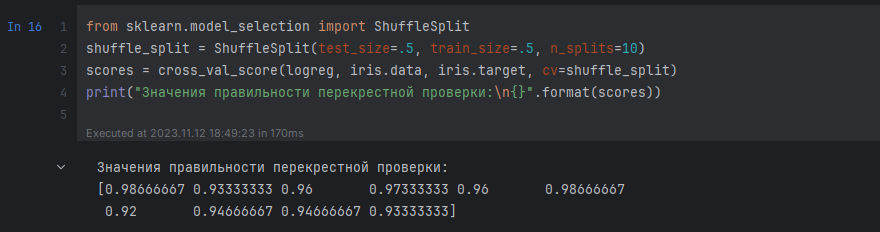
### Перекрестная проверка со случайными перестановками при разбиении

Еще одной, очень гибкой стратегией перекрестной проверки является перекрестная проверка со случайными перестановками при разбиении (**shuffle-split cross-validation**). В этом виде проверки каждое разбиение выбирает train\_size точек для обучающего набора и test\_size точек для тестового набора (при этом обучающее и тестовое подмножества не пересекаются). Точки выбираются с возвращением. Разбиение повторяется **n\_iter** раз. Рис. 3 иллюстрирует четырехпроходное разбиение набора данных, состоящего из 10 точек, на обучающий набор из 5 точек и тестовый набор из 2 точек (чтобы задать абсолютные размеры этих подмножеств вы можете использовать для train\_size и test\_size целочисленные значения, либо числа с плавающей точкой, чтобы задать доли от общей выборки):



*Рис. 3 Перекрестная проверка со случайными перестановками при разбиении для набора данных из 10 точек, train\_size=5, test\_size=2 и n\_iter=4*

Программный код, приведенный ниже, 10 раз разбивает данные на 50%-ный обучающий набор и 50%-ный тестовый набор:



Перекрестная проверка со случайными перестановками при разбиении позволяет задавать количество итераций независимо от размеров обучающего и тестового наборов, что иногда может быть полезно. Кроме того, этот метод позволяет использовать на каждой итерации лишь часть данных (значения **train\_size** и **test\_size** необязательно должны в сумме давать 1). Подобное прореживание данных может быть полезно при работе с большими наборами данных.

Существует также стратифицированный вариант ShuffleSplit, названный StratifiedShuffleSplit, который позволяет получить более надежные результаты при решении задач классификации.

### Перекрестная проверка с использованием групп

Еще одна весьма распространенная настройка для перекрестной проверки применяется, когда данные содержат сильно взаимосвязанные между собой группы.

Допустим, необходимо построить систему распознавания эмоций по фотографиям лиц и для этого вы собрали набор изображений 100 человек, в котором каждый человек сфотографирован несколько раз, чтобы зафиксировать разные эмоции. Цель заключается в том, чтобы построить классификатор, который сможет правильно определить эмоции людей, не включенных в этот набор изображений.

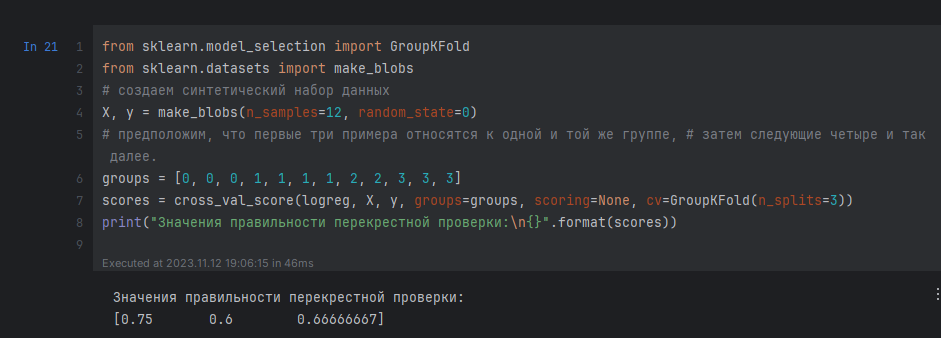
В данном случае для оценки качества работы классификатора вы можете использовать традиционную стратифицированную перекрестную проверку. Однако, вполне вероятно, что фотографии одного и того же человека попадут как в обучающий, так и в тестовый наборы. По сравнению с совершенно новым лицом классификатору намного проще будет определить эмоции по лицу, которое уже встречалось ему в обучающем наборе. Чтобы точно оценить способность модели обобщать результат на новые лица, необходимо убедиться в том, что обучающий и тестовый наборы содержат изображения разных людей.

Для решения этой задачи мы можем воспользоваться **GroupKFold**, принимающий в качестве аргумента массив groups.

С помощью него указываем, какой человек изображен на снимке. В данном случае массив groups указывает группы данных, которые не следует разбивать при создании обучающего и тестового наборов, при этом их не следует путать с метками классов.

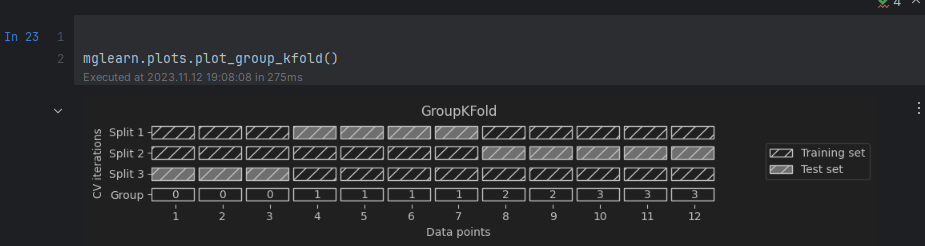
*Подобные группы данных часто встречаются в медицинской практике, когда у вас, возможно, есть несколько наблюдений по одному и тому же пациенту, но вы заинтересованы в обобщении результатов на новых пациентов. Аналогично в задачах распознавания речи у вас может быть несколько записей одного и того же человека, но вас интересует точность распознавания речи новых людей.*

Ниже приведен пример с использованием синтетического набора данных, группировка данных задана массивом groups. Набор данных состоит из 12 точек данных, и для каждой точки массив groups задает группу (допустим, пациента), к которой относится эта точка. У нас существуют четыре группы, первые три примера принадлежат к первой группе, следующие четыре примера принадлежат ко второй группе и так далее:



Примеры не нужно сортировать по группам, мы сделали это в иллюстративных целях. Разбиения, вычисляемые на основе этих меток, показаны на рис. 4.

Видно, что при выполнении разбиении каждая группа полностью попадает либо в обучающий набор, либо в тестовый набор:

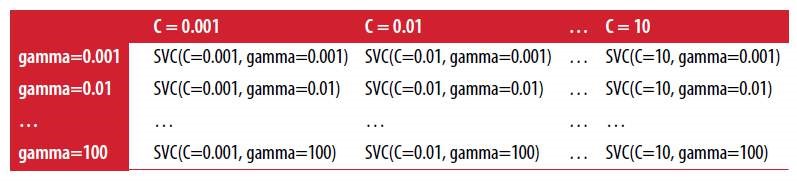
*Рис. 4 Разбиение на основе меток групп с помощью GroupKFold*

## Решетчатый поиск

Поиск оптимальных значений ключевых параметров модели (то есть значений, которые дают наилучшую обобщающую способность) является сложной задачей, но она обязательна почти для всех моделей и наборов данных. Поскольку поиск оптимальных значений параметров является общераспространенной задачей, библиотека scikit-learn предлагает стандартные методы, позволяющие решить ее.

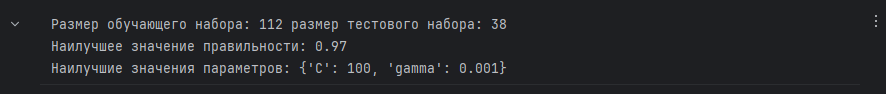
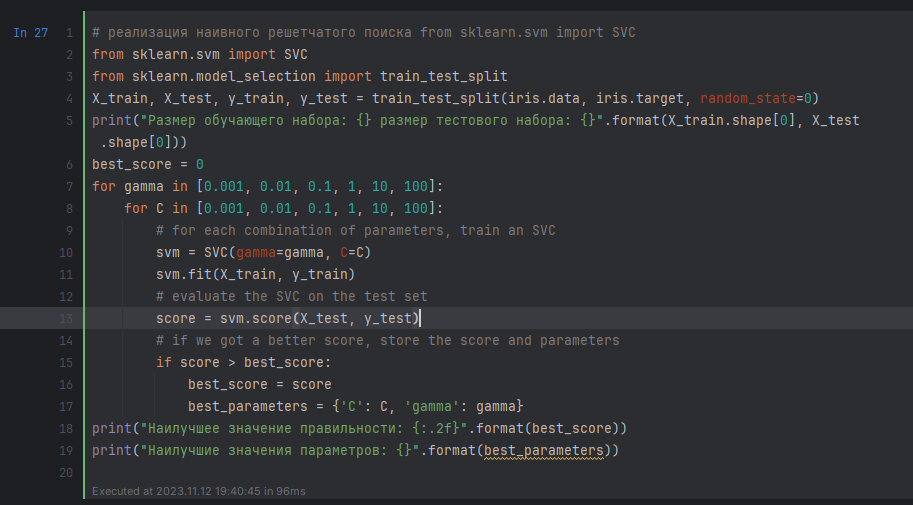
Наиболее часто используемый метод – это **решетчатый поиск (grid search),** который по сути является попыткой перебрать все возможные комбинации интересующих параметров.

Рассмотрим применение ядерного метода SVM с ядром RBF (радиальной базисной функцией), реализованного в классе SVC. В ядерном методе опорных векторов есть два важных параметра: **ширина ядра gamma** и **параметр регуляризации C**. Допустим, мы хотим попробовать значения 0.001, 0.01, 0.1, 1, 10 и 100 для параметра С и то же самое для параметра gamma. Поскольку нам нужно попробовать шесть различных настроек для C и gamma, получается 36 комбинаций параметров в целом. Все возможные комбинации формируют таблицу (которую еще называют решеткой или сеткой) настроек параметров для SVM, как показано ниже:



### Простой решетчатый поиск

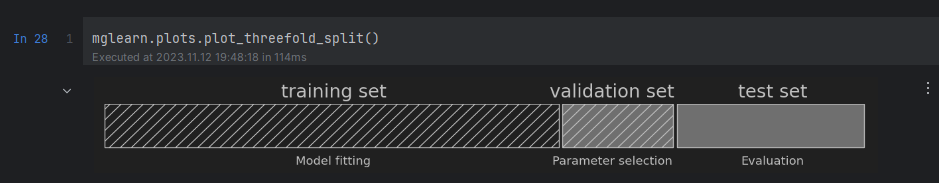
Мы можем реализовать простой решетчатый поиск с помощью вложенных циклов for по двум параметрам, обучая и оценивая классификатор для каждой комбинации:



### Опасность переобучения параметров и проверочный набор данных

Получив такой результат, мы могли бы заявить, что нашли модель, которая дает 97%-ную правильность на нашем наборе данных. Однако это заявление может быть чрезмерно оптимистичным (или просто неверным) по следующей причине: мы перебрали множество значений параметров и выбрали ту комбинацию значений, которая дает наилучшую правильность на тестовом наборе, но это вовсе не означает, что на новых данных мы получим такое же значение правильности. Поскольку мы использовали тестовый набор для настройки параметров, мы больше не можем использовать его для оценки качества модели. Это та же самая причина, по которой нам изначально нужно разбивать данные на обучающий и тестовый наборы. Теперь для оценки качества модели нам необходим независимый набор данных, то есть набор, который не использовался для построения модели и настройки ее параметров.

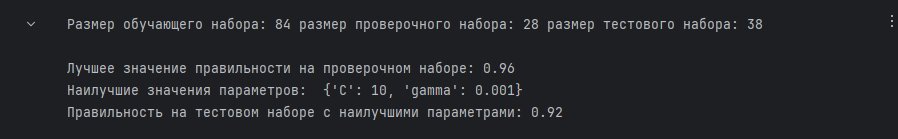
Один из способов решения этой проблемы заключается в том, чтобы разбить данные еще раз, таким образом, мы получаем три набора: обучающий набор для построения модели, проверочный (валидационный) набор для выбора параметров модели, а также тестовый набор для оценки качества работы выбранных параметров. Рис. 5 показывает, как это выглядит:



*Рис. 5 Тройное разбиение данных на обучающий набор, проверочный набор и тестовый набор*

После выбора наилучших параметров с помощью проверочного набора проверки, мы можем заново построить модель, используя найденные настройки, но теперь на основе объединенных обучающих и проверочных данных. Таким образом, можем использовать для построения модели максимально возможное количество данных. Это приводит к следующему программному коду:

from sklearn.svm import SVC   
# разбиваем данные на обучающий+проверочный набор и тестовый набор   
X\_trainval, X\_test, y\_trainval, y\_test = train\_test\_split(iris.data, iris.target, random\_state=0)   
# разбиваем обучающий+проверочный набор на обучающий и проверочный наборы   
X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X\_trainval, y\_trainval, random\_state=1)   
print("Размер обучающего набора: {} размер проверочного набора: {} размер тестового набора:" " {}\n".format(X\_train.shape[0], X\_valid.shape[0], X\_test.shape[0]))   
   
best\_score = 0   
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:   
 for C in [0.001, 0.01, 0.1, 1, 10, 100]:   
 # для каждой комбинации параметров обучаем SVC   
 svm = SVC(gamma=gamma, C=C)   
 svm.fit(X\_train, y\_train)   
 # оцениваем качество SVC на тестовом наборе   
 score = svm.score(X\_valid, y\_valid)   
 # если получаем наилучшее значение правильности, сохраняем значение и параметры   
 if score > best\_score:   
 best\_score = score   
 best\_parameters = {'C': C, 'gamma': gamma}   
# заново строим модель на наборе, полученном в результате объединения обучающих  
# и проверочных данных, оцениваем качество модели на тестовом наборе   
svm = SVC(\*\*best\_parameters)   
svm.fit(X\_trainval, y\_trainval)   
test\_score = svm.score(X\_test, y\_test)   
print("Лучшее значение правильности на проверочном наборе: {:.2f}".format(best\_score))   
print("Наилучшие значения параметров: ", best\_parameters)   
print("Правильность на тестовом наборе с наилучшими параметрами: {:.2f}".format(test\_score))



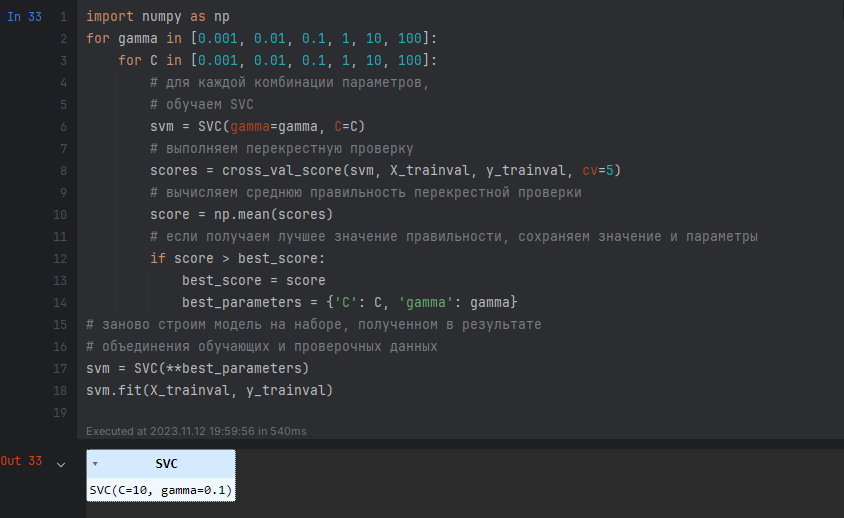
Лучшее значение правильности на проверочном наборе составляет 96%, что немного ниже значения правильности, полученного для тестового набора ранее, вероятно, из-за того, что мы использовали меньше данных для обучения модели (размер X\_train теперь стал меньше, поскольку что мы разбили наш набор данных дважды). Однако значение правильности на тестовом наборе, значение, которое показывает реальную обобщающую способность – стало еще ниже, 92%. Таким образом, мы можем утверждать, что правильность классификации новых данных составляет 92%, а не 97%, как мы думали ранее!

Наличие различий между обучающим, проверочным и тестовым наборами имеет принципиально важное значение для применения методов машинного обучения на практике. Любой выбор, сделанный, исходя из правильности на тестовом наборе, «сливает» модели информацию тестового набора. Поэтому важно иметь отдельный тестовый набор, который используется лишь для итоговой оценки. Осуществление всего разведочного анализа и отбора модели с помощью комбинации обучающего и проверочного наборов и резервирование тестового набора для итоговой оценки является хорошей практикой. Данная практика является верной даже при проведении разведочной визуализации. Строго говоря, оценка качества моделей и выбор наилучшей из них с помощью тестового набора, использующегося для отбора параметров, приведет к чрезмерно оптимистичной оценке правильности модели.

### Решетчатый поиск с перекрестной проверкой

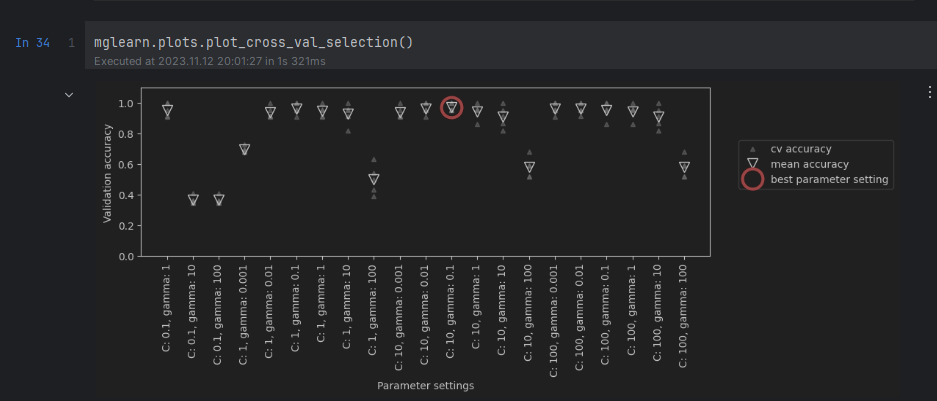
Хотя только что рассмотренный нами метод разбиения данных на обучающий, проверочный и тестовый наборы является вполне рабочим и относительно широко используемым, он весьма чувствителен к правильности разбиения данных. Взглянув на вывод, приведенный для предыдущего фрагмента программного кода, мы видим, что **GridSearchCV** выбрал в качестве лучших параметров 'C': 10, 'gamma': 0.001, тогда как вывод, приведенный для программного кода в предыдущем разделе, сообщает нам, что наилучшими параметрами являются 'C': 100, 'gamma': 0.001.

Для лучшей оценки обобщающей способности вместо одного разбиения данных на обучающий и проверочный наборы мы можем воспользоваться перекрестной проверкой. Теперь качество модели оценивается для каждой комбинации параметров по всем разбиениям перекрестной проверки. Этот метод можно реализовать с помощью следующего программного кода:



Чтобы c помощью пятиблочной перекрестной проверки оценить правильность SVM для конкретной комбинации значений C и gamma, нам необходимо обучить 36\*5=180 моделей. Как вы понимаете, основным недостатком использования перекрестной проверки является время, которое требуется для обучения всех этих моделей.

Следующая визуализация (рис. 6) показывает, как в предыдущем программном коде осуществляется выбор оптимальных параметров:

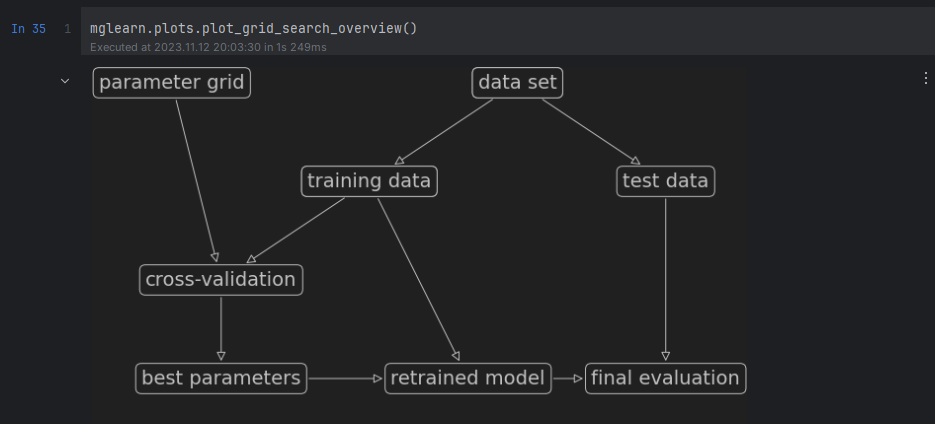


*Рис. 6 Результаты решетчатого поиска с перекрестной проверкой*

Для каждой комбинации значений С и gamma (здесь показана лишь часть комбинаций) вычисляются пять значений правильности, по одному для каждого разбиения в перекрестной проверке. Затем для каждой комбинации параметров вычисляется среднее значение правильности перекрестной проверки. В итоге выбирается комбинация с наибольшей средней правильностью перекрестной проверки и отмечается кружком.

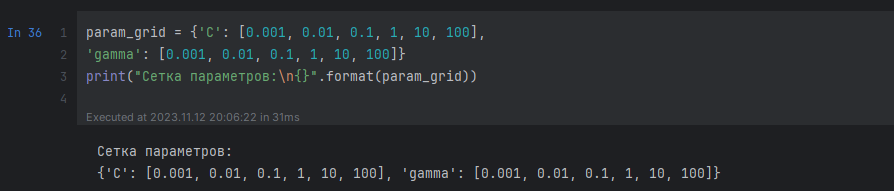
*Как мы уже говорили ранее, перекрестная проверка – это способ оценить качество работы конкретного алгоритма на определенном наборе данных. Однако она часто используется в сочетании с методами поиска параметров типа решетчатого поиска. По этой причине многие люди в разговорной речи под термином перекрестная проверка (cross-validation) подразумевают решетчатый поиск с перекрестной проверкой.*

Общий процесс разбиения данных, запуска решетчатого поиска, а также оценки итоговых параметров показан на рис. 7:

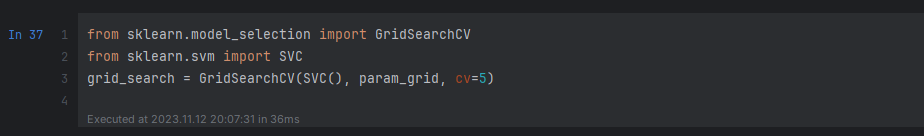


*Рис. 7 Процесс отбора параметров и оценки модели с помощью GridSearchCV*

Поскольку решетчатый поиск с перекрестной проверкой является весьма распространенным методом настройки параметров, библиотека scikit-learn предлагает класс **GridSearchCV**, в котором решетчатый поиск реализован в виде модели. Чтобы воспользоваться классом GridSearchCV, сначала необходимо указать искомые параметры с помощью словаря. GridSearchCV построит все необходимые модели. Ключами словаря являются имена настраиваемых параметров (в данном случае С и gamma), а значениями – тестируемые настройки параметров. Перебор значений 0.001, 0.01, 0.1, 1, 10 и 100 для C и gamma требует словаря следующего вида:



Теперь мы можем создать экземпляр класса GridSearchCV, передав модель (SVC), сетку искомых параметров (param\_grid), а также стратегию перекрестной проверки, которую мы хотим использовать (допустим, пятиблочную стратифицированную перекрестную проверку):



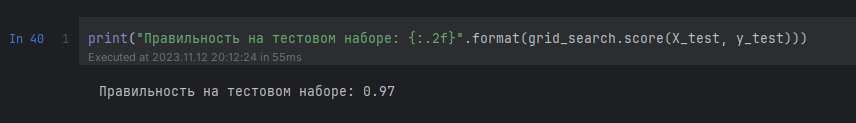
Вместо разбиения на обучающий и проверочный набор, использованного нами ранее, GridSearchCV запустит перекрестную проверку. Однако нам по-прежнему нужно разделить данные на обучающий и тестовый наборы, чтобы избежать переобучения параметров:



Созданный нами объект grid\_search аналогичен классификатору, мы можем вызвать стандартные методы fit, predict и score от его имени.[[1]](#footnote-1) Однако, когда мы вызываем fit, он запускает перекрестную проверку для каждой комбинации параметров, указанных в param\_grid:

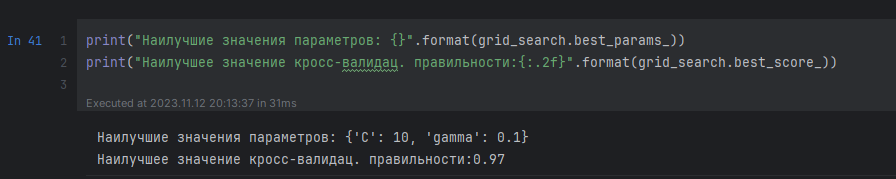


Процесс подгонки объекта GridSearchCV включает в себя не только поиск лучших параметров, но и автоматическое построение новой модели на всем обучающем наборе данных. Для ее построения используются параметры, которые дают наилучшее значение правильности перекрестной проверки. Поэтому процесс, запускаемый вызовом метода fit, эквивалентен программному коду **In[33]**. Класс GridSearchCV предлагает очень удобный интерфейс для работы с моделью, используя методы predict и score. Чтобы оценить обобщающую способность найденных наилучших параметров, мы можем вызвать метод score:



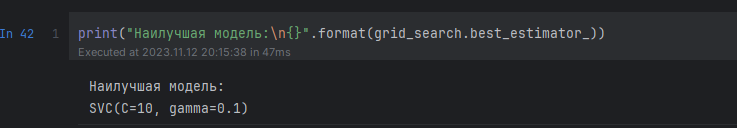
Выбрав параметры с помощью перекрестной проверки, мы фактически нашли модель, которая достигает правильности 97% на тестовом наборе. Главный момент здесь в том, что мы не использовали тестовый набор для отбора параметров.

Найденная комбинация параметров сохраняется в атрибуте best\_params\_, а наилучшее значение правильности перекрестной проверки (значение правильности, усредненное по всем разбиениям для данной комбинации параметров) – в атрибуте best\_score\_.



*Опять же, будьте осторожны, чтобы не перепутать best\_score\_ со значением обобщающей способности модели, которое вычисляется на тестовом наборе с помощью метода score. Метод score помощью метода (оценивающий качество результатов, полученных с predict) использует модель, построенную на всем обучающем наборе данных. В атрибуте best\_score\_ записывается средняя правильность перекрестной проверки. Для ее вычисления используется модель, построенная на обучающем наборе перекрестной проверки.*

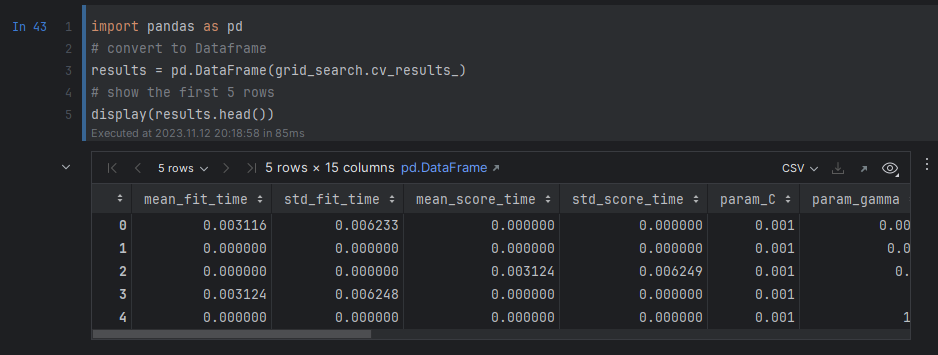
В ряде случаев вам необходимо будет ознакомиться с полученной моделью, например, взглянуть на коэффициенты или важности признаков. Посмотреть наилучшую модель, построенную на всем обучающем наборе, вы можете с помощью атрибута best\_estimator\_:



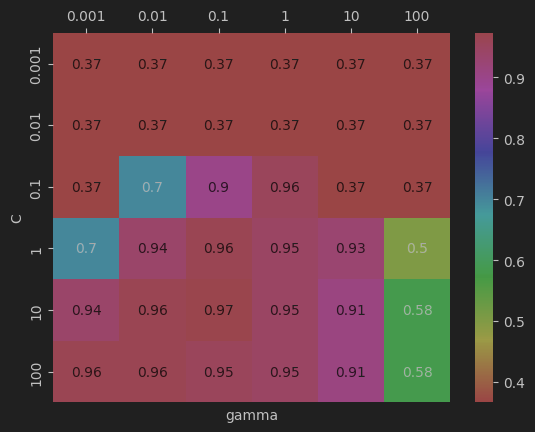
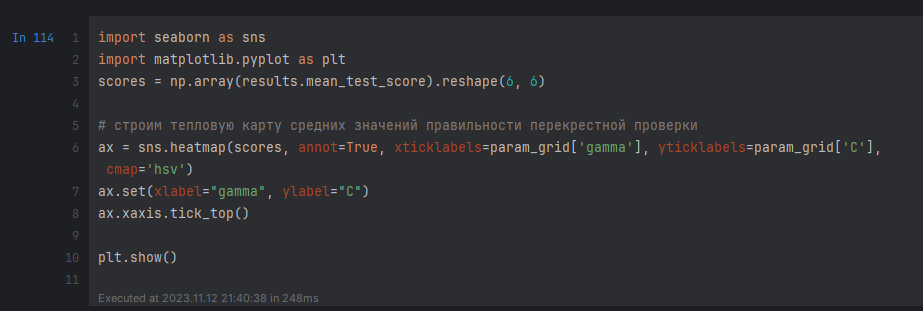
Поскольку grid\_search уже сам по себе включает методы predict и score, использование best\_estimator\_ для получения прогнозов и оценки качества модели не требуется.

### Анализ результатов перекрестной проверки

Часто бывает полезно визуализировать результаты перекрестной проверки, чтобы понять, как обобщающая способность зависит от искомых параметров. Поскольку выполнение решетчатого поиска довольно затратно с вычислительной точки зрения, целесообразно начинать поиск с простой и небольшой сетки параметров. Затем мы можем проверить результаты решетчатого поиска, использовав перекрестную проверку, и, возможно, расширить наш поиск. Результаты решетчатого поиска можно найти в атрибуте cv\_results, который является словарем, хранящим все настройки поиска. Как вы можете увидеть в выводе, приведенном ниже, словарь содержит множество деталей и принимает более привлекательный вид после преобразования в пандасовский DataFrame.



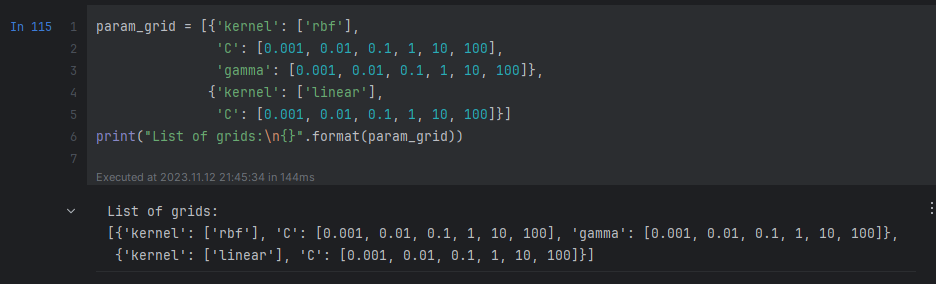
Каждая строка в results соответствует одной конкретной комбинации параметров. Для каждой комбинации записываются результаты всех разбиений перекрестной проверки, а также среднее значение и стандартное отклонение по всем разбиениям. Поскольку мы осуществляли поиск на основе двумерной сетки параметров (C и gamma), наилучший способ визуализировать этот процесс, представить его в виде тепловой карты (рис. 8). Сначала мы извлечем средние значения правильности перекрестной проверки, затем изменим форму массива со значениями так, чтобы оси соответствовали C и gamma:



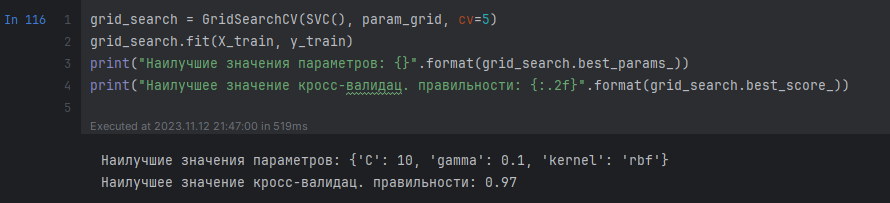
*Рис. 8 Тепловая карта для усредненной правильности перекрестной проверки, выраженной в виде функции двух параметров С и gamma*

### Экономичный решетчатый поиск

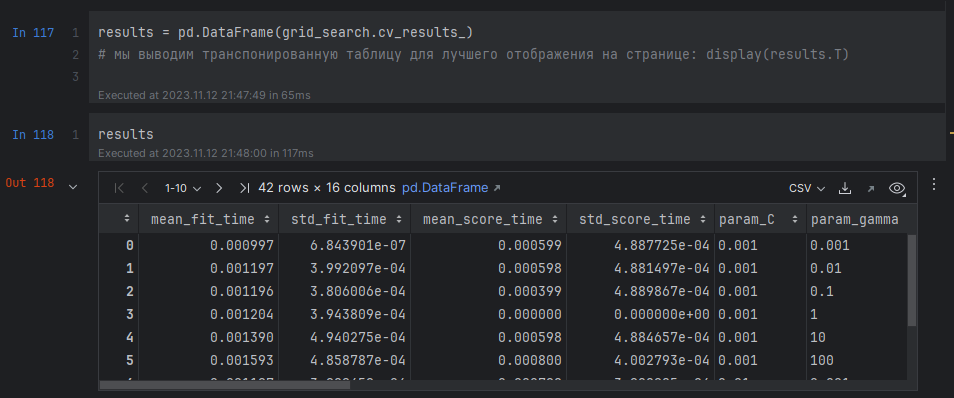
В некоторых случаях перебор всех возможных комбинаций по всем параметрам, который обычно выполняет GridSearchCV, не является хорошей идеей. Например, SVC имеет параметр kernel, и в зависимости от того, какое ядро выбрано, все остальные параметры будут иметь соответствующие этому выбору значения. Если kernel='linear', модель является линейной и используется только параметр С. Если используется kernel='rbf', используются параметры C и gamma (однако другие параметры типа degree не используются). В этом случае поиск по всем возможным комбинациям C, gamma и kernel не имеет смысла: если kernel='linear', то gamma не используется и перебор различных значений gamma – это пустая трата времени. Чтобы обработать подобные «условные» параметры, GridSearchCV позволяет превратить param\_grid в список словарей. Каждый словарь в списке выделяется в самостоятельную сетку параметров. Возможный решетчатый поиск, включающий настройки ядра и параметров, мог бы выглядеть следующим образом:



В первой сетке параметр kernel всегда принимает значение 'rbf' (обратите внимание, элемент параметра kernel представляет собой список единичной длины), изменяются значения как параметра C, так и параметра gamma. Во второй сетке параметр kernel всегда принимает значение linear и поэтому изменяется только параметр С. Теперь давайте применим этот более сложный поиск параметров:



Давайте снова посмотрим на cv\_results\_. Как и следовало ожидать, если kernel имеет значение 'linear', то меняется только параметр C:



### Применение различных стратегий перекрестной проверки с помощью решетчатого поиска

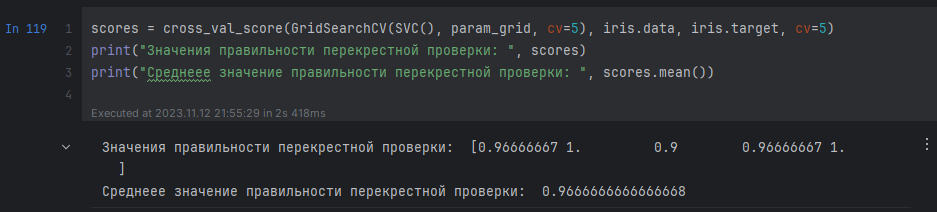
Как и cross\_val\_score, GridSearchCV использует по умолчанию kблочную перекрестную проверку для классификации и k-блочную перекрестную проверку для регрессии. Однако при использовании GridSearchCV вы можете дополнительно передать любой генератор разбиения (как было описано в разделе «Больше контроля над перекрестной проверкой») в качестве параметра cv. В частности, чтобы получить только одно разбиение на обучащий и проверочный наборы, вы можете воспользоваться ShuffleSplit или StratifiedShuffleSplit с n\_iter=1. Данная настройка может оказаться полезной для очень больших наборов данных или очень медленных моделей.

### Вложенная перекрестная проверка

В предыдущих примерах мы прошли путь от использования одного разбиения данных на обучающий, проверочный и тестовый наборы (раздел «Опасность переобучения параметров и проверочный набор данных») до разбиения данных на обучающий и тестовый наборы с проведением перекрестной проверки на обучающем наборе (раздел «Решетчатный поиск с перекрестной проверкой»). Но при использовании GridSearchCV ранее описанным способом мы все еще выполняем всего лишь одно разбиение на обучающий и тестовый наборы, что может привести к получению нестабильных результатов и ставит нас в зависимость от этого единственного разбиения данных. Мы можем пойти дальше и вместо однократного разбиения исходных данных на обучающий и тестовый наборы использовать несколько разбиений перекрестной проверки. В результате мы получим вложенную перекрестную проверкой (nested cross-validation). Во вложенной перекрестной проверке используется внешний цикл по разбиениям данных на обучающий и тестовый наборы. Для каждого из них выполняется решетчатый поиск (в результате чего для каждого разбиения внешнего цикла можно получить разные наилучшие параметры). Затем для каждого внешнего разбиения выводится правильность на тестовом наборе с использованием наилучших параметров.

Результатом этой процедуры является не модель и не настройки параметров, а список значений правильности. Значения правильности указывают нам на обобщающую способность модели с использованием лучших параметров, найденных в ходе решетчатого поиска. Поскольку вложенная перекрестная проверка не дает модель, которую можно использовать на новых данных, ее редко используют при поиске прогнозной модели для применения к новым данным. Тем не менее, она может быть полезна для оценки работы модели на конкретном наборе данных.

Реализовать вложенную перекрестную проверку в scikit-learn довольно просто. Мы вызываем cross\_val\_score и передаем ей экземпляр GridSearchCV в качестве модели.

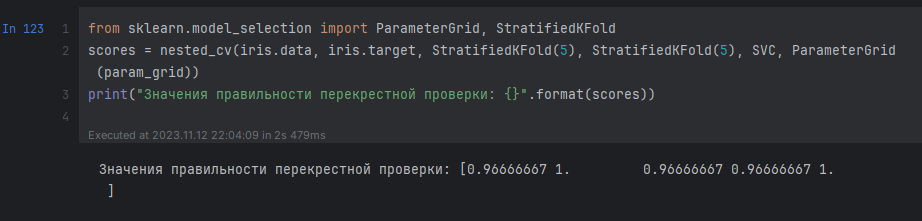


Результат нашей вложенной перекрестной проверки можно резюмировать так: «на наборе данных iris модель SVC может достигнуть средней правильности перекрестной проверки 98%» – ни больше, ни меньше.

В данном случае мы использовали стратифицированную пятиблочную перекрестную проверку как во внутреннем, так и во внешнем циклах. Поскольку наша сетка param\_grid содержит 36 комбинаций параметров, будет построено целых 36 \* 5 \* 5 = 900 моделей, что делает процедуру вложенной перекрестной проверки очень затратной с вычислительной точки зрения. В данном случае во внутреннем и внешнем циклах мы использовали один и тот же генератор разбиений, однако это не является необходимым условием и поэтому для внутреннего и внешнего циклов вы можете использовать любую комбинацию стратегий перекрестной проверки. Понимание процесса, который происходит внутри одной строки, приведенной выше, может представлять определенную сложность. Данный процесс можно визуализировать с помощью циклов for, как это сделано в следующей упрощенной реализации программного кода:

def nested\_cv(X, y, inner\_cv, outer\_cv, Classifier, parameter\_grid):  
 outer\_scores = []  
 # для каждого разбиения данных во внешней перекрестной проверке   
 # (метод split возвращает индексы)   
 for training\_samples, test\_samples in outer\_cv.split(X, y):  
 # находим наилучшие параметры с помощью внутренней перекрестной проверки   
 best\_parms = {}  
 best\_score = -np.inf  
 # итерируем по параметрам   
 for parameters in parameter\_grid:  
 # собираем значения правильности по всем внутренним разбиениям   
 cv\_scores = []  
 # итерируем по разбиениям внутренней перекрестной проверки   
 for inner\_train, inner\_test in inner\_cv.split(  
 X[training\_samples], y[training\_samples]):  
 # строим классификатор с данными параметрами на внутреннем обучающем наборе  
 clf = Classifier(\*\*parameters)  
 clf.fit(X[inner\_train], y[inner\_train])  
 # оцениваем качество на внутреннем тестовом наборе  
 score = clf.score(X[inner\_test], y[inner\_test])  
 cv\_scores.append(score)  
 # вычисляем среднее значение правильности по внутренним блокам  
 mean\_score = np.mean(cv\_scores)  
 if mean\_score > best\_score:  
 # если лучше, чем предыдущие, запоминаем параметры   
 best\_score = mean\_score  
 best\_params = parameters  
 # строим классификатор с лучшими параметрами на внешнем обучающем наборе  
 clf = Classifier(\*\*best\_params)  
 clf.fit(X[training\_samples], y[training\_samples])  
 # оцениваем качество на внешнем тестовом наборе   
 outer\_scores.append(clf.score(X[test\_samples], y[test\_samples]))  
 return np.array(outer\_scores)

Теперь давайте применим эту функцию к набору данных iris:



## Метрики качества моделей и их вычисление

До сих пор мы оценивали качество классификации, используя правильность (долю правильно классифицированных примеров), и качество регрессии, используя R2. Однако это лишь два показателя из большого количества возможных метрик, используемых для оценки качества контролируемой модели на данном наборе данных. На практике эти метрики качества могут не соответствовать вашим задачам и поэтому очень важно при отборе моделей и корректировке параметров подобрать правильную метрику.

### Метрики бинарной классификации

Бинарная классификация является, пожалуй, наиболее распространенным и концептуально простым примером практического применения машинного обучения. Однако даже при решении этой простой задачи существует целый ряд нюансов. Прежде чем мы углубимся в альтернативные метрики, давайте рассмотрим ситуации, в которых правильность измерения может ввести в заблуждение. Вспомним, что в случае бинарной классификации мы говорим о положительном (positive) классе и отрицательном (negative) классе, подразумевая под положительным классом интересующий нас класс.

#### Типы ошибок

Как правило, правильность не является адекватным показателем прогностической способности, поскольку количество совершаемых ошибок не содержит весь объем интересующей нас информации. Представьте себе скрининговое обследование для раннего обнаружения рака, построенное на основе автоматизированного теста. Если тест отрицателен, пациент будет считаться здоровым, тогда как если тест положителен, пациент будет подвергнут дополнительному обследованию. Здесь мы называем положительным тестом (наличие рака) положительный класс, а отрицательный тест соответствует отрицательному классу. Мы не можем быть уверены в отличной работе модели, она неизбежно будет совершать ошибки. Выполняя тот или иной проект, мы должны спросить себя, какими могут быть последствия этих ошибок в реальном мире.

Одна из возможных ошибок заключается в том, что здоровый пациент будет классифицирован как больной (положительный класс), что даст повод для дополнительного тестирования. Дополнительное обследование приведет к некоторым затратам и неудобствам для пациента (и, возможно, к определенному психическому дискомфорту). Пример, неправильно спрогнозированный как положительный, называется ложноположительным (false positive). Другая возможная ошибка состоит в том, что больной пациент будет классифицирован как здоровый (отрицательный класс), не пройдет дополнительные тесты и не получит лечения. Недиагностированный вовремя рак может привести к серьезным проблемам со здоровьем и может даже закончиться смертельным исходом. Пример, неправильно спрогнозированный как отрицательный, называется ложно отрицательным (false negative). В статистике ложно положительный пример также известен как ошибка I рода (type I error), а ложно отрицательный пример – как ошибка II рода (type II error). Мы будем придерживаться определений «ложно отрицательный пример» и «ложно положительный пример», поскольку они являются более явными и их легче запомнить. В примере с диагностикой рака очевидно, что мы хотим минимизировать долю ложно отрицательных примеров, тогда как ложно положительные примеры можно считать гораздо менее значительной неприятностью.

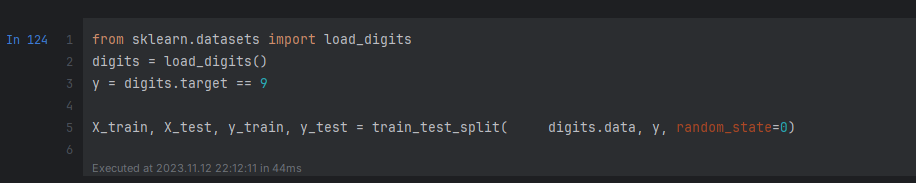
Хотя вышеприведенный пример является довольно ярким, каждый ложно положительный и ложно отрицательный прогноз редко приводит к одним и тем же последствиям. В коммерческих проектах обоим видам ошибок можно присвоить определенные стоимости, которые позволяют измерить погрешность конкретного прогноза в денежном выражении, а не с точки зрения правильности. Для процесса принятия бизнес-решений, использующего модель, данный шаг имеет гораздо большее значение.

### Несбалансированный набор ошибок

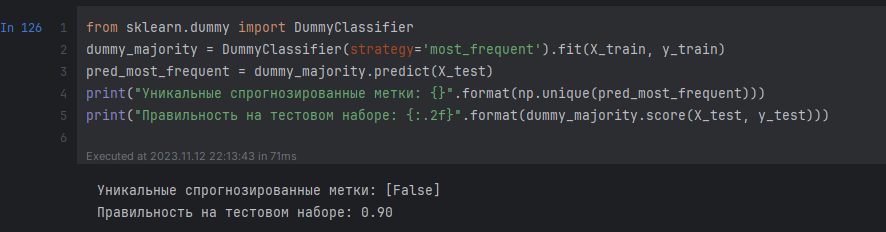
Типы ошибок играют важную роль, когда один из двух классов встречается гораздо чаще, чем другой. Это очень распространенная ситуация на практике. Хорошим примером является прогноз рейтинга кликов, где каждая точка данных представляет собой «показ» – элемент, предъявленный пользователю. Этим элементом может быть объявление, рассказ, пользователь социальной сети. Цель состоит в том, чтобы предсказать, будет ли пользователь при показе данного элемента кликать по нему (что указывает на его интерес). Большинство из того, что видит пользователь в Интернете (в частности, рекламные объявления), не вызывает у него особого интереса. Вам потребуется показать пользователю 100 объявлений или статей, прежде чем он найдет что-то достаточно интересное для себя, чтобы кликнуть. Это позволяет получить набор данных, в котором 99 точек данных соответствуют ситуации «не кликнул» и 1 точка данных – «кликнул». Другими словами, 99% примеров относятся к классу «отсутствие клика». Наборы данных, в которых один класс встречается гораздо чаще, чем остальные, часто называют несбалансированными наборами данных (imbalanced datasets) или наборами данных с несбалансированными классами (datasets with imbalanced classes). В реальности несбалансированные данные являются нормой и редко бывает, что интересующий класс встречался в данных с одинаковой или почти такой же частотой, что и остальные классы.

Теперь предположим, что вы строите классификатор, который при решении задачи прогнозирования кликов имеет правильность 99%. О чем это говорит? Правильность 99% звучит впечатляюще, но она не принимает во внимание дисбаланс классов. Вы можете достичь 99%-ной правильности и без построения модели машинного обучения, всегда прогнозируя «отсутствие клика» С другой стороны, даже для несбалансированных данных модель с 99%-ной правильностью могла бы быть вполне пригодной. Однако в данном случае правильность не позволяет нам отличить модель «постоянно прогнозируем отсутствие клика» от потенциально хорошей модели.

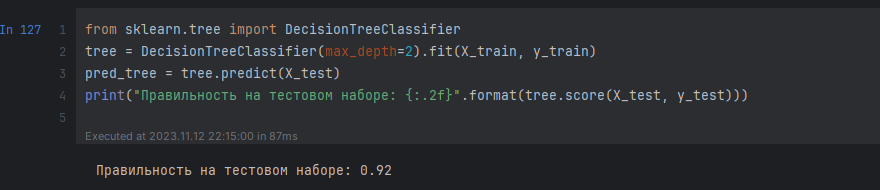
Чтобы проиллюстрировать это, мы на основе набора данных digits создадим несбалансированный набор данных с пропорциями 9:1, создав два класса «не-девятка» и «девятка»:



Мы можем воспользоваться DummyClassifier, который всегда предсказывает мажоритарный класс (в данном случае класс «недевятка»), чтобы проиллюстрировать, насколько малоинформативной может быть правильность:

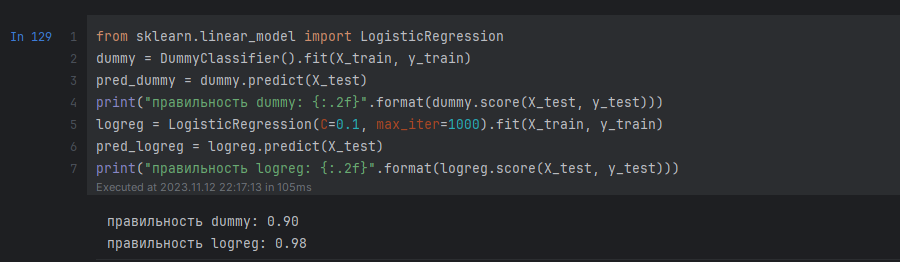


Мы получили 90%-ную правильностью без какого-либо обучения. Это может показаться поразительным, но задумайтесь об этом на минуту. Представьте себе, кто-то говорит вам, что его модель имеет 90%-ную правильность. Можно сделать вывод, что он проделал очень хорошую работу. Но это вполне возможно, лишь правильно прогнозируя один класс! Давайте сравним этот результат с результатом, полученным с помощью реальной модели:



С точки зрения правильности DecisionTreeClassifier оказался чуть лучше, чем DummyClassifier, постоянно предсказывающего мажоритарный класс. Это может означать, что либо мы неправильно использовали DecisionTreeClassifier, либо правильность на самом деле не является в данном случае адекватной метрикой.

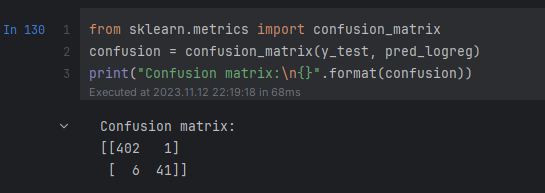
Для сравнения давайте оценим качество еще двух классификаторов, LogisticRegression и обычный DummyClassifier, который выдает случайные прогнозы:



Дамми-классификатор, который генерирует случайные прогнозы, имеет намного худшее качество (с точки зрения правильности), в то время как логистическая регрессия дает очень хорошие результаты. Однако даже случайный классификатор дает 80%-ную правильность. Поэтому очень трудно судить, какой из этих результатов является действительно полезным. Проблема здесь заключается в том, что для несбалансированных наборов данных правильность не является адекватной метрикой, позволяющей количественно оценить прогностическую способность модели. В оставшейся части этой главы мы рассмотрим альтернативные метрики, которые дают более четкие ориентиры при выборе модели. В частности, нам нужны такие метрики, которые позволяют сравнить правильность модели машинного обучения с правильностью классификатора, всегда предсказывающего «наиболее часто встречающийся класс», или случайного классификатора (в данном случае такие классификаторы были вычислены с помощью pred\_most\_frequent и pred\_dummy). Если мы используем какую-то метрику для оценки модели, она должна уметь отсекать эти бессмысленные прогнозы.

### Матрица ошибок

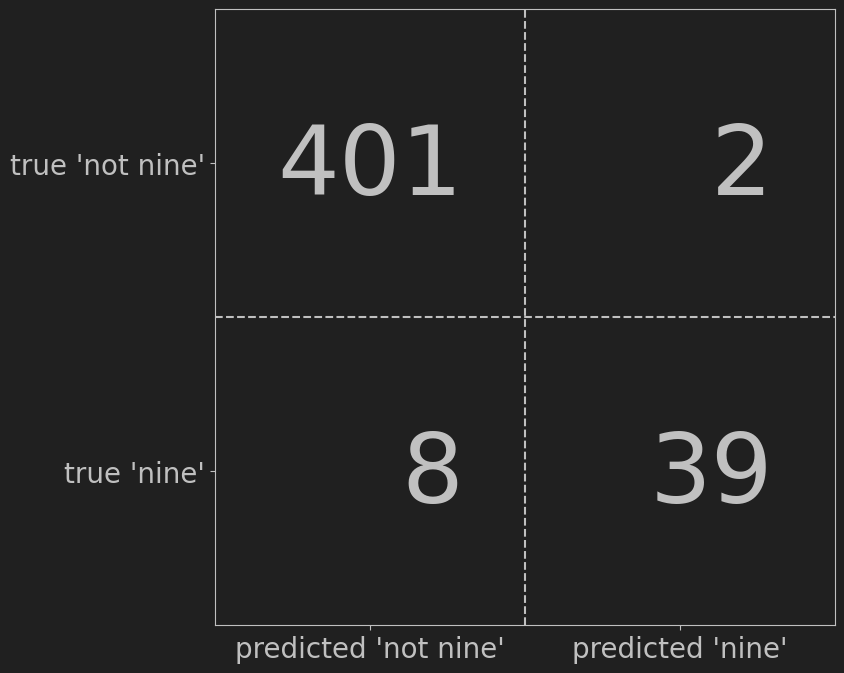
Одним из наиболее развернутых способов, позволяющих оценить качество бинарной классификации, является использование матрицы ошибок. Давайте исследуем прогнозы модели LogisticRegression, построенной в предыдущем разделе, с помощью функции confusion\_matrix. Прогнозы для тестового набора данных мы уже сохранили в pred\_logreg:



Вывод confusion\_matrix представляет собой массив размером 2x2, где строки соответствуют фактическим классам, а столбцы соответствуют спрогнозированным классам. В данном случае речь идет о классах «недевятка» и «девятка». Число в каждой ячейке показывает количество примеров, когда спрогнозированный класс, представленный столбцом, совпадает или не совпадает с фактическим классом, представленным строкой.

Следующий график (рис. 9) иллюстрирует сказанное:





Фактич класс «не-девятка»

Фактич класс «девятка»

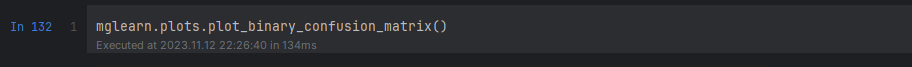
Спрогноз класс «девятка»

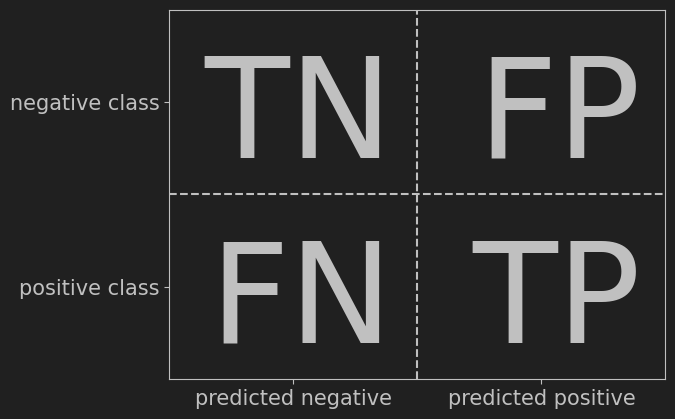
Спрогноз класс «не-девятка»

*Рис. 9 Матрица ошибок для классификационной задачи «девятка против остальных»*

Элементы главной диагонали [[2]](#footnote-2) матрицы ошибок соответствуют правильным прогнозам (результатам классификации), тогда как остальные элементы показывают, сколько примеров, относящихся к одному классу, были ошибочно классифицированы как другой класс.

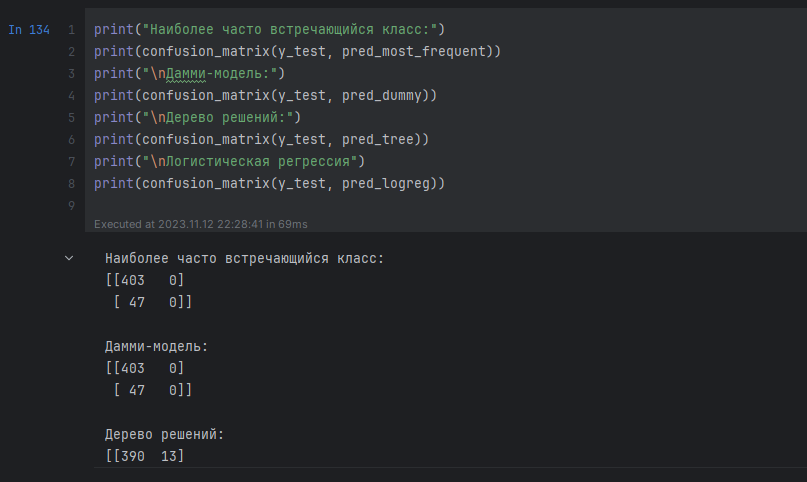
Объявив «девятку» положительным классом, мы можем рассмотреть элементы матрицы ошибок в терминах ложно положительных (false positive) и ложно отрицательных (false negative) примеров, которые мы ввели ранее. Для полноты картины мы назовем правильно классифицированные положительные примеры истинно положительными (true positive), а правильно классифицированные отрицательные примеры – истинно отрицательными (true negative). Эти термины, как правило, записывают в сокращенном виде как FP, FN, TP и TN и приводят к следующей интерпретации матрицы ошибок (рис. 10):





*Рис. 10 Матрица ошибок для бинарной классификации*

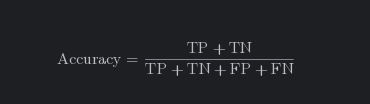
Теперь давайте воспользуемся матрицей ошибок для сравнения ранее построенных моделей (две дамми-модели, дерево решений, а также логистическая регрессия):



Взглянув на матрицу ошибок, становится совершенно ясно, что с моделью pred\_most\_frequent что-то не так, потому что она всегда предсказывает один и тот же класс. С другой стороны, модель pred\_dummy характеризуется очень маленьким количеством истинно положительных примеров (4) по сравнению с остальными примерами, при этом количество ложно положительных примеров существенно больше количества истинно положительных примеров! Прогнозы, полученные с помощью дерева решений, несут гораздо больше смысла, чем прогнозы дамми-модели, хотя правильность у этих моделей почти одинаковая. И, наконец, мы видим, что прогнозы логистической регрессии лучше прогнозов pred\_tree во всех аспектах: она имеет большее количество истинно положительных и истинно отрицательных примеров, в то время количество ложно положительных и ложно отрицательных примеров стало меньше. Из этого сравнения ясно, что лишь дерево решений и логистическая регрессия дают разумные результаты, при этом логистическая регрессия работает лучше дерева во всех отношениях. Однако интерпретация матрицы ошибок немного громоздка и хотя мы получили массу информации, анализируя все аспекты матрицы, процесс работы с матрицей ошибок был трудоемким и сложным. Есть несколько способов обобщить информацию, содержащуюся в матрице ошибок.

### Связь с правильностью

Мы уже знакомы с одним из способов обобщить результаты матрицы – вычислением правильности, которую можно выразить в виде следующей формулы:



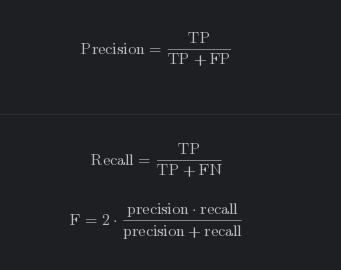
Другими словами, правильность – это количество верно классифицированных примеров (TP и TN), поделенное на общее количество примеров (суммируем все элементы матрицы ошибок).

### Точность, полнота и F-мера

Есть еще несколько способов подытожить информацию матрицы ошибок, наиболее часто используемыми из них являются точность и полнота. Точность (**precision**) показывает, сколько из предсказанных положительных примеров оказались действительно положительными. Таким образом, точность – это доля истинно положительных примеров от общего количества предсказанных положительных примеров.

Точность используется в качестве показателя качества модели, когда цель состоит в том, чтобы снизить количество ложно положительных примеров. В качестве примера представьте модель, которая должна спрогнозировать, будет ли эффективен новый лекарственный препарат при лечении болезни. Клинические испытания, как известно, дороги, и фармацевтическая компания хочет провести их лишь в том случае, когда полностью убедится, что препарат действительно работает. Поэтому важно минимизировать количество ложно положительных примеров, другими словами, необходимо увеличить точность. Точность также известна как прогностическая ценность положительного результата (positive predictive value, PPV).

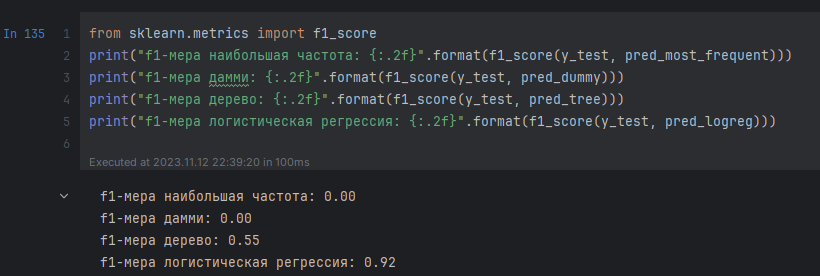
С другой стороны, полнота (**recall**) показывает, сколько от общего числа фактических положительных примеров было предсказано как положительный класс. Полнота – это доля истинно положительных примеров от общего количества фактических положительных примеров



Полнота используется в качестве показателя качества модели, когда нам необходимо определить все положительные примеры, то есть, когда важно снизить количество ложно отрицательных примеров. Пример диагностики рака, приведенный ранее в этой главе, является хорошей иллюстрацией подобной задачи: важно выявить всех больных пациентов, при этом, возможно, включив в их число здоровых пациентов. Другие названия полноты – чувствительность (sensitivity), процент результативный ответов или хит-рейт (hit rate) и доля истинно положительных примеров (true positive rate, TPR).

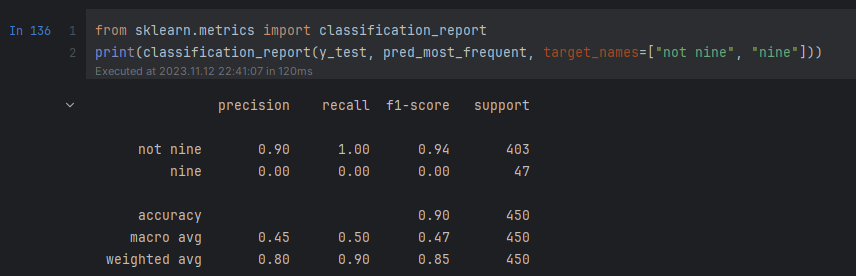
Хотя точность и полнота являются очень важными метриками, сами по себе они не дадут вам полной картины. Одним из способов подытожить их является F-мера (F-measure), которая представляет собой гармоническое среднее точности и полноты.

Этот вариант вычисления F-меры еще известен как f1-мера. Поскольку f1-мера учитывает точность и полноту, то для бинарной классификации несбалансированных данных она может быть более лучшей метрикой, чем правильность. Давайте применим ее к прогнозам для нашего набора данных «девятка против остальных», полученным нами ранее. В данном случае мы будем считать класс «девятка» положительным классом (он получает метку True, тогда как класс «не-девятка» получает метку False), таким образом, положительный класс является миноритарным классом:



Здесь мы можем отметить два момента. Во-первых, мы получаем сообщение об ошибке для прогнозов модели most\_frequent, поскольку не было получено ни одного прогноза положительного класса (таким образом, знаменатель в формуле расчета f-меры равен нулю). Кроме того, мы можем увидеть довольно сильное различие между прогнозами даммимодели и прогнозами дерева, которое не так явно бросается в глаза, когда мы анализируем только правильность. Использовав f-меру для оценки качества, мы снова подытоживаем прогностическую способность с помощью одного числа. Однако, похоже, что f-мера действительно дает более лучшее представление о качестве модели, чем правильность. Вместе с тем недостаток f-меры заключается в том, что в отличие от правильности ее труднее интерпретировать и объяснить.

Если мы хотим получить более развернутый отчет о точности, полноте и f1-мере, можно воспользоваться удобной функцией classification\_report, чтобы вычислить все три метрики сразу и распечатать их в привлекательном виде:

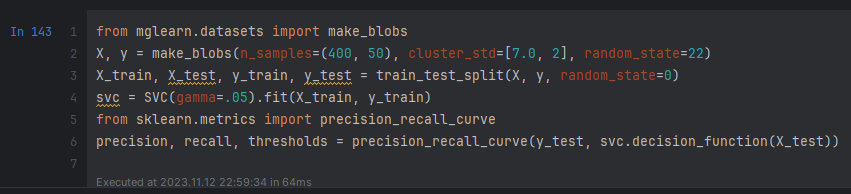


### Кривые точности - полноты и ROC-кривые

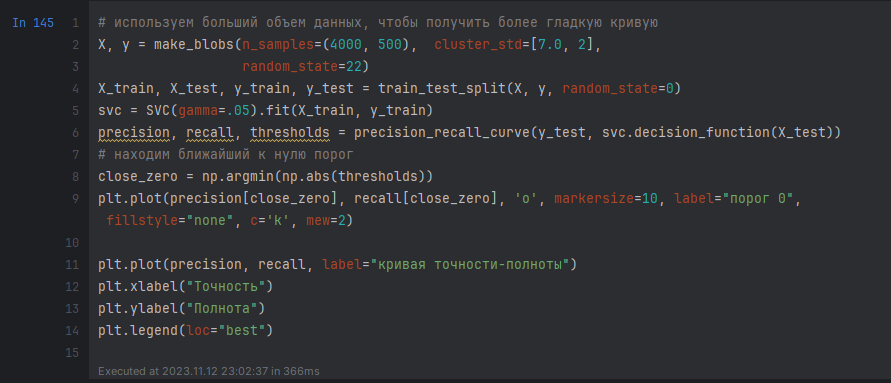
Как мы уже сказали, изменение порога, используемого для классификации решений модели – это способ, позволяющий найти компромисс между точностью и полнотой для данного классификатора. Возможно, вы хотите пропустить менее 10% положительных примеров, таким образом, желаемое значение полноты составит 90%. Решение зависит от конкретного примера, и оно должно определяться бизнес-целями. Как только поставлена конкретная цель, скажем, задано конкретное значение полноты или точности для класса, можно установить соответствующий порог. Всегда можно задать то или иное пороговое значение для реализации конкретной цели (например, достижения значения полноты 90%). Трудность состоит в разработке такой модели, которая при этом пороге еще и будет иметь приемлемое значение точности, ведь классифицировав все примеры как положительные, вы получите значение полноты, равное 100%, но при этом ваша модель будет бесполезной.

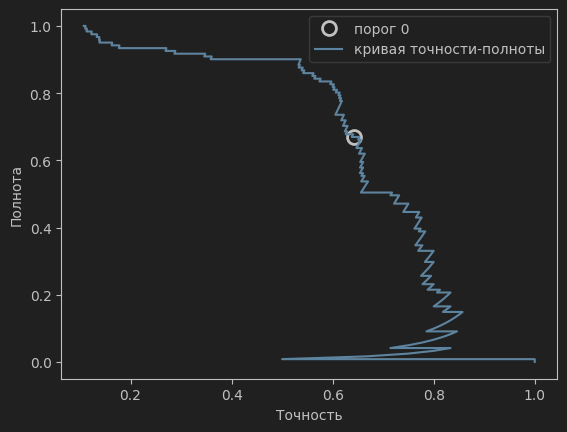
Требование, выдвигаемое к качеству модели (например, значение полноты должно быть 90%), часто называют рабочей точкой (operating point). Фиксирование рабочей точки часто бывает полезно в контексте бизнеса, чтобы гарантировать определенный уровень качества клиентам или другим группам лиц внутри организации.

Как правило, при разработке новой модели нет четкого представления о том, что будет рабочей точкой. По этой причине, а также для того, чтобы получить более полное представление о решаемой задаче, полезно сразу взглянуть на все возможные пороговые значения или все возможные соотношения точности и полноты для этих пороговых значений. Данную процедуру можно осуществить с помощью инструмента, называемого кривой точности-полноты (precision-recall curve). Функцию для вычисления кривой точности-полноты можно найти в модуле sklearn.metrics. Ей необходимо передать фактические метки классов и спрогнозированные вероятности, вычисленные с помощью decision\_function или predict\_proba:



Функция precision\_recall\_curve возвращает список значений точности и полноты для всех возможных пороговых значений (всех значений решающей функции) в отсортированном виде, поэтому мы можем построить кривую, как показано на рис. 11:





*Рис. 11 Кривая точности-полноты для SVC (gamma=0.05)*

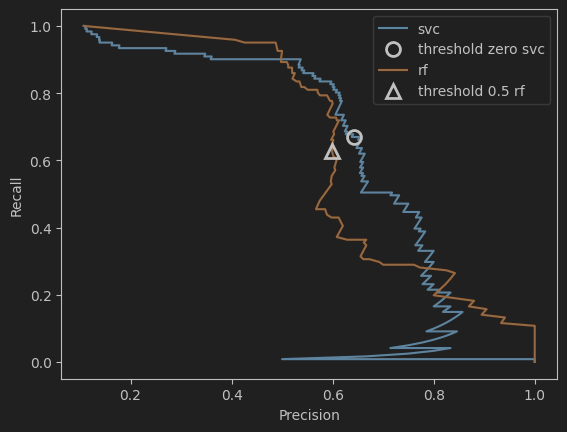
Каждая точка на кривой (рис. 11) соответствует возможному пороговому значению решающей функции. Например, видно, что мы можем достичь полноты 0.4 при точности около 0.75. Черный кружок отмечает точку, соответствующую порогу 0, пороговому значению по умолчанию для решающей функции. Данная точка является компромиссом, который выбирается при вызове метода predict.

Чем ближе кривая подходит к верхнему правом углу, тем лучше классификатор. Точка в верхнем правом углу означает высокое значение точности и высокое значение полноты для соответствующего порога. Кривая начинается в верхнем левом углу, что соответствует очень низкому порогу, все примеры классифицируются как положительный класс. Повышение порога перемещает кривую в сторону более высоких значений точности и в то же время более низких значений полноты. При дальнейшем повышении порога мы получаем ситуацию, в которой большинство точек, классифицированных как положительные, являются истинно положительными, что приводит к очень высокой точности, но более низкому значению полноты. Чем больше модель сохраняет высокое значение полноты при одновременном увеличении точности, тем лучше.

Взглянув на эту кривую чуть более пристально, можно увидеть, что с помощью построенной модели можно добиться точности в районе 0.5 при очень высоком значении полноты. Если мы хотим получить гораздо более высокое значение точности, мы должны в значительной степени пожертвовать полнотой. Другими словами, слева наша кривая выглядит относительно плоской, это означает, что при увеличении точности полнота падает незначительно. Однако, чтобы получить значение точности более 0.5, нам придется пожертвовать значительным снижением полноты.

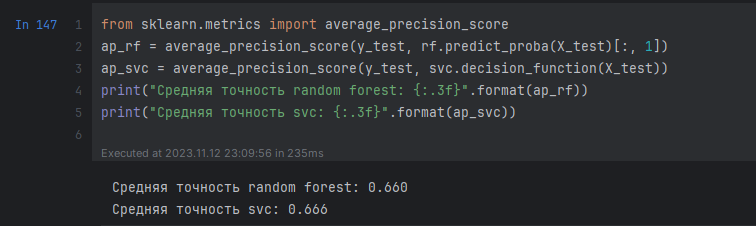
Различные классификаторы могут давать хорошее качество на различных участках кривой, то есть в разных рабочих точках. Давайте сравним модель SVM с моделью случайного леса, построенной на том же наборе данных. RandomForestClassifier вместо decision\_function использует метод predict\_proba. Функция precision\_recall\_curve ожидает, что в качестве второго аргумента ей будет передана вероятность положительного класса (класса 1), то есть rf.predict\_proba(X\_test)[:, 1]. В бинарной классификации пороговое значение по умолчанию для predict\_proba равно 0.5, поэтому мы отметили эту точку на кривой (см. рис. 12):

from sklearn.ensemble import RandomForestClassifier  
  
rf = RandomForestClassifier(n\_estimators=100, random\_state=0, max\_features=2)  
rf.fit(X\_train, y\_train)  
  
# RandomForestClassifier has predict\_proba, but not decision\_function  
precision\_rf, recall\_rf, thresholds\_rf = precision\_recall\_curve(  
 y\_test, rf.predict\_proba(X\_test)[:, 1])  
  
plt.plot(precision, recall, label="svc")  
  
plt.plot(precision[close\_zero], recall[close\_zero], 'o', markersize=10,  
 label="threshold zero svc", fillstyle="none", c='k', mew=2)  
  
plt.plot(precision\_rf, recall\_rf, label="rf")  
  
close\_default\_rf = np.argmin(np.abs(thresholds\_rf - 0.5))  
plt.plot(precision\_rf[close\_default\_rf], recall\_rf[close\_default\_rf], '^', c='k',  
 markersize=10, label="threshold 0.5 rf", fillstyle="none", mew=2)  
plt.xlabel("Precision")  
plt.ylabel("Recall")  
plt.legend(loc="best")



*Рис. 12 Сравнение кривых точности-полноты для SVM и случайного леса*

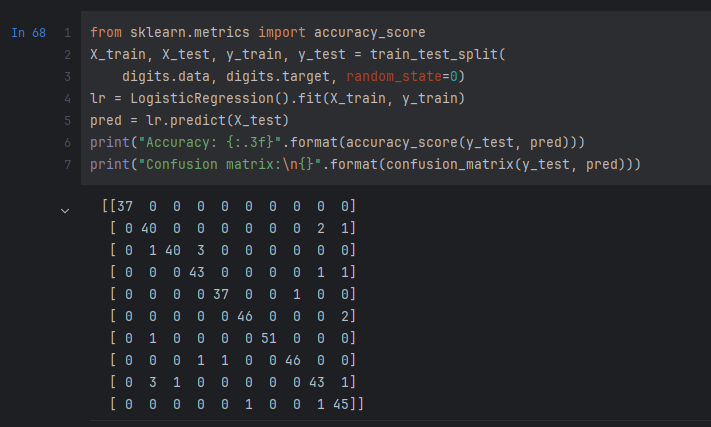
Сравнение двух кривых точности-полноты дает много детальной информации, но представляет собой довольно трудоемкий процесс. Чтобы выполнить автоматическое сравнение моделей мы могли бы обобщить информацию, содержащуюся в кривой, не ограничиваясь конкретным пороговым значением или рабочей точкой. Один из способов подытожить информацию кривой заключается в вычислении интеграла или площади под кривой точности-полноты, он также известен как метод средней точности (average precision). [[3]](#footnote-3) Для вычисления средней точности вы можете воспользоваться функцией average\_precision\_score. Поскольку нам нужно вычислить ROC-кривую и рассмотреть несколько пороговых значений, функции average\_precision\_score вместо результата predict нужно передать результат decision\_function или predict\_proba:



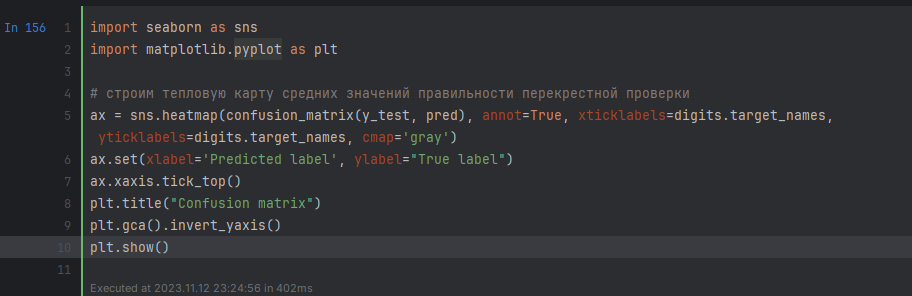
При усреднении по всем возможным пороговым значением мы видим, что случайный лес и SVC дают примерно одинаковое качество модели, при этом случайный даже чуть-чуть вырывается вперед. Это в значительном мере отличаются от результата, полученного нами ранее с помощью f1\_score. Поскольку средняя точность равна площади под кривой, которая принимает значения от 0 до 1, средняя точность всегда возвращает значение от 0 (худшее значение) до 1 (лучшее значение). Средняя точность случайного классификатора равна доле положительных примеров в наборе данных.

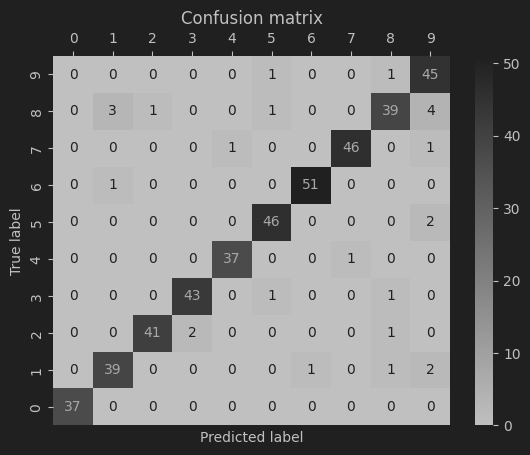
## Метрики для мультиклассовой классификации

Все метрики для мультиклассовой классификации являются производными от метрик бинарной классификации, но при этом усредняются по всем классам. В мультиклассовой классификации правильность вновь определяется как доля правильно классифицированных примеров. И опять же, когда классы не сбалансированы, правильность перестает быть адекватной метрикой оценки качества. Представьте себе задачу трехклассовой классификации, когда 85% точек данных принадлежат к классу А, 10% – к классу В и 5% – к классу C. Что означает среднее значение правильности 85% применительно к этому набору данных? В целом результаты мультиклассовой классификации труднее интерпретировать, чем результаты бинарной классификации. Помимо правильности часто используемыми инструментами являются матрица ошибок и отчет о результатах классификации, которые мы рассматривали, разбирая случай бинарной классификации в предыдущем разделе. Давайте применим эти два метода оценки для классификации 10 различных рукописных цифр в наборе данных digits:



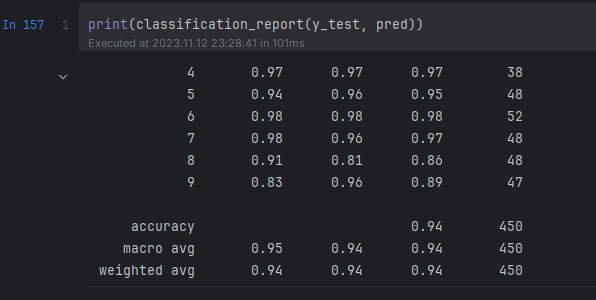
Модель имеет точность 95.3%, что уже говорит нам об очень хорошем качестве модели. Матрица ошибок дает нам несколько более подробную информацию. Как и в случае бинарной классификации, каждая строка соответствует фактической метке класса, а каждый столбец соответствует спрогнозированной метке класса. Вы можете построить более наглядный график, приведенный на рис. 13:

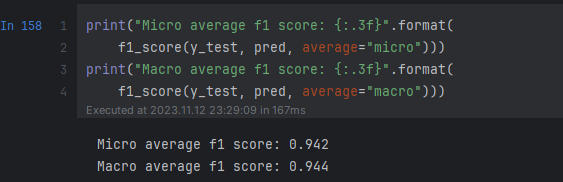




*Рис. 13 Матрица ошибок для десятиклассовой задачи распознавания рукописных цифр*

Фактическое количество примеров, относящихся к первому классу (цифре 0), равно 37 и все эти примеры были классифицированы как класс 0 (то есть ложно отрицательные примеры для класса 0 отсутствуют). Об этом говорит тот факт, что все остальные элементы первой строки матрицы ошибок имеют нулевые значения. Кроме того, ни одна из остальных цифр не была ошибочно классифицирована как 0, поскольку все остальные элементы первого столбца имеют нулевые значения (то есть ложно положительные примеры для класса 0 отсутствуют). Однако некоторые цифры были спутаны с остальными, например, цифра 2 (третья строка), три примера, являющиеся цифрой 2, были классифицированы как цифра 3 (четвертый столбец). Кроме того, у нас есть одна цифра 3, классифицированная как 2 (третий столбец, четвертая строка), и одна цифра 8, классифицированная как 2 (третий столбец, девятая строка).





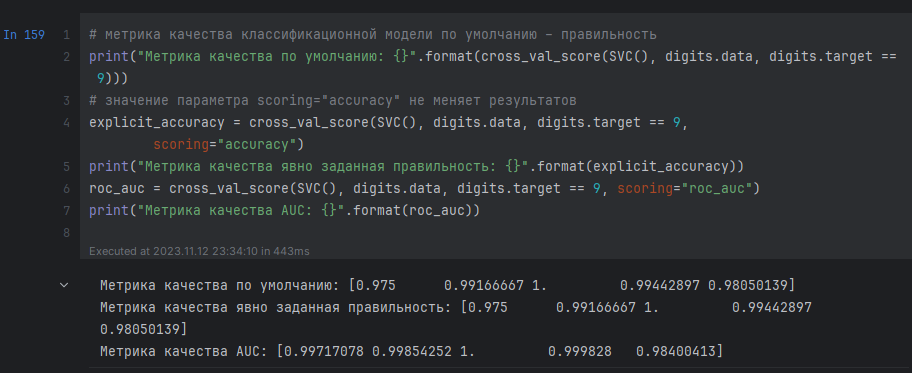
## Метрики регрессии

Оценить качество регрессии можно таким же способом, которой мы использовали для классификации, например, сравнив количество завышенных и заниженных расчетных значений зависимой переменной. Однако в большинстве рассмотренных примеров будет достаточно применения R2, который в методе score используется по умолчанию для всех моделей регрессии. Иногда бизнес-решения принимаются на основе среднеквадратической ошибки или средней абсолютной ошибки, что является стимулом для использования этих метрик при настройке моделей. Однако в целом мы пришли к выводу, что с точки зрения оценки качества регрессионных моделей R2 является более понятной метрикой.

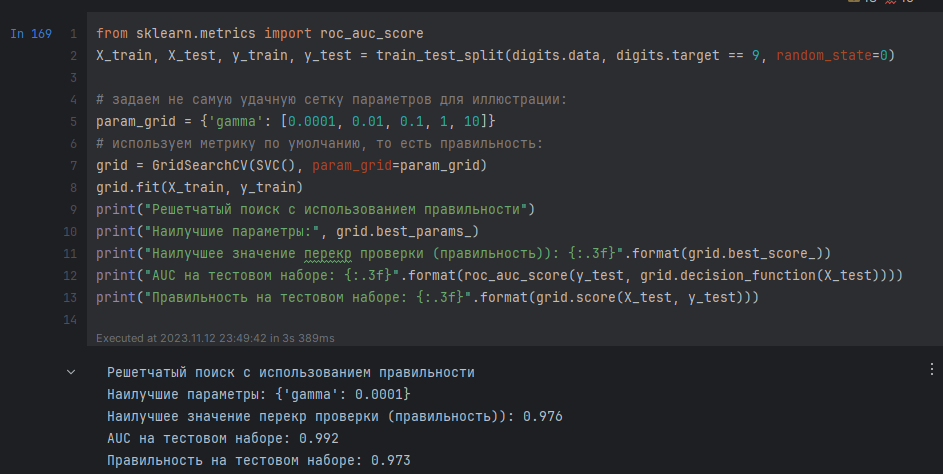
### Использование метрик оценки для отбора модели

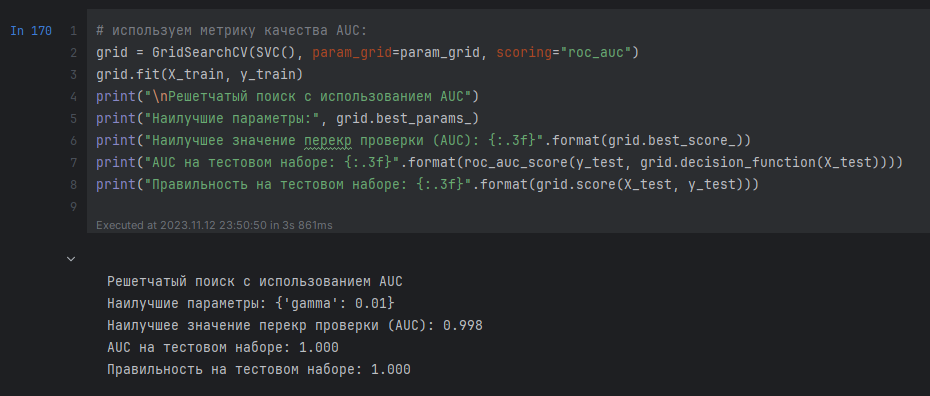
Мы подробно рассмотрели множество методов оценки и обсудили их применение с учетом фактических и спрогнозированных результатов.

Однако часто нам нужно воспользоваться метриками типа AUC для отбора модели, выполняемого на основе GridSearchCV или cross\_val\_score. К счастью, scikit-learn предлагает очень простой способ решения этой задачи с помощью аргумента scoring, который можно использовать как в GridSearchCV, так и в cross\_val\_score. Вы можете просто задать строку с описанием необходимой метрики оценки. Допустим, мы хотим оценить качество классификатора SVM при решении задачи «девять против остальных» для набора данных digits, используя значение AUC. Чтобы поменять метрику оценки с правильности, установленной по умолчанию, на AUC, достаточно указать "roc\_auc" в качестве параметра scoring:



Точно так же мы можем изменить метрику, используемую для отбора наилучших параметров в Grid-SearchCV:





Когда использовалась правильность, был выбран параметр gamma=0.0001, тогда как при использовании AUC был выбран gamma= 0.01. В обоих случаях правильность перекрестной проверки соответствует правильности на тестовом наборе. Однако использование AUC позволила найти настройку параметра, оптимальную с точки зрения AUC и даже с точки зрения правильности.[[4]](#footnote-4)

Наиболее важными значениями параметра scoring для классификации являются accuracy (по умолчанию), roc\_auc для площади под ROC-кривой, average\_precision (площадь под кривой точности-полноты), f1, f1\_macro, f1\_micro и f1\_weighted для бинарной f1-меры и различных стратегий усреднения. Для регрессии, наиболее часто используемыми значениями являются r2 для R2, mean\_squared\_error для среднеквадратической ошибки и mean\_absolute\_error для средней абсолютной ошибки. Полный список поддерживаемых аргументов вы можете найти, ознакомившись с [документацией](http://scikit-learn.org/stable/modules/model_evaluation.html#the-scoring-parameter-defining-model-evaluation-rules):

<https://scikit-learn.ru/3-3-metrics-and-scoring-quantifying-the-quality-of-predictions/>

# ЗАДАНИЕ:

1. Изучить материал и примеры, описанные в ходе работы[[5]](#footnote-5) (датасет iris).
2. Используя один из подходящих игрушечных датасетов (breast\_canser, digits, diabetes …) применить алгоритмы улучшения качества и оценки моделей (перекрестная проверка, решетчатый поиск, метрики модели)
3. Создать модель классификатор, которая по определенному обучаемому набору (salary, city, age, vacation\_prefer, transport\_prefer) будет определять предпочтения человека (target), в каком городе провести отпуск.
   1. Создать свой датасет с категориальными данными (1000 строк), которые содержат следующие поля: (salary, city, age, vacation\_prefer, transport\_prefer, target). Применить import random (random. choice, random.randint)
   2. **Salary** – установить числовой тип (например 50000), **city** – город проживания (например, Bishkek), **age** – возраст (например, от 30 до 65), **vacation\_prefer** – тип отдыха (например, Shopping или Beach holiday), **transport\_prefer** – тип транспорта (например, auto, plane), **target** – город, в котором проведем отпуск (например, London, Moscow)
   3. Преобразовать категориальные данные в числовые, используя panadas.get\_dummies.
   4. Выделить обучающую выборку и тестовую выборку (X\_train, y\_train, X-test, y\_test)
   5. Выбрать модель классификатор (например, from sklearn.ensemble import RandomForestClassifier)
   6. Проверить оценку модели.
   7. Сделать предсказание на случайных данных.
   8. Улучшить модель согласно примерам лекций 9-10 и материала лабораторной работы

1. Модель scikit-learn, которая создается с помощью другой модели называется метамоделью (metaestimator). GridSearchCV является наиболее часто используемой метамоделью, но об этом мы поговорим позже. [↑](#footnote-ref-1)
2. Главная диагональ двумерного массива или матрицы A имеет вид A[i, i].

   [↑](#footnote-ref-2)
3. С технической точки зрения существует некоторые незначительные различия между площадью под кривой точности-полноты и средней точностью. Однако приведенное объяснение передает общую идею. [↑](#footnote-ref-3)
4. Вероятно, решение с более высоким значением правильности, полученное с помощью AUC – это следствие того, что правильность не является адекватной метрикой качества модели при несбалансированных данных. [↑](#footnote-ref-4)
5. Источник: Andreas C. Mueller and Sarah Guido. Introduction to Machine Learning with Python. O’Reilly 2016 p.268-325 [↑](#footnote-ref-5)