**Лабораторные работы 5-6**

МАШИННОЕ ОБУЧЕНИЕ НА ПИТОНЕ. БИБЛИОТЕКА SCIKIT-LEARN.

**Базовые определения.**

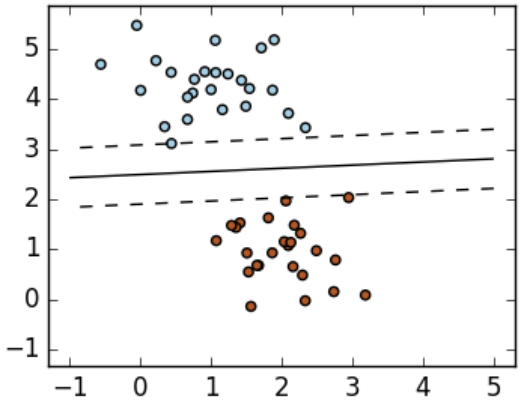
Машинное обучение можно рассматривать как область искусственного интеллекта, в которой параметры алгоритмов машинного обучения настраиваются таким образом, чтобы поведение компьютерных систем было более разумным, а не только для хранения и извлечения данных, как работают базы данных.

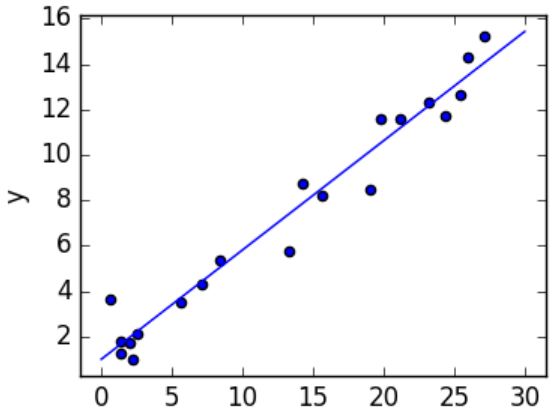
Для начала рассмотрим две базовые задачи машинного обучения. Первая – это задача классификации: на рисунке показаны два набора данных, раскрашенных различными цветовыми метками согласно их классовой принадлежности. Задача алгоритма классификации построить разделительную линию между двумя этими наборами.

Построив разделительную линию, мы обучим модель, которая можно обобщить на новые данные: когда на плоскость добавляется новая точка и алгоритм должен предсказать какого цвета она будет синего или красного.

Следующая простая задача – это регрессия: построение аппроксимирующей линии по набору данных.

Однако, основная задача – это обобщении модели на новые данные. Модель была построена на обучающей выборке и затем используется для прогнозирования результатов на тестируемой выборке, задавая значение *x*, необходимо оценить значение *y*.




**Матрица данных.**

Алгоритмы машинного обучения, реализованные в библиотеке *scikit-learn*, предполагают, что данные организованы в виде двухмерных массивов или матриц. Эти массивы могут быть *numpy* массивы или *scipy.sparse* матрицы. Размер массива *[n_samples, n_features]*.

*n_samples* – количество образцов: каждый образец –это объект для исследования (например классификации). В качестве образца может быть документ, рисунок, звук, видео, CSV файл, строка в базе данных либо что угодно, что можно описать используя фиксированный набор признаков.

*n_features* – количество признаков, которые можно использовать для описания объектов. Чаще всего признаки вещественные числа, однако могут быть так же дискретные или булевые переменные.

В дальнейшем количество признаков может быть зафиксировано, однако их может быть и несколько миллионов, преимущественно нулевые значения, что на практике заставляет использовать специальные методы для работы с разряженными матрицами, которые реализованы в модуле *scipy.sparse*, вместо стандартных *numpy* массивов.

**ПЕРВЫЙ ПРИМЕР. БАЗА ДАННЫХ ИРИСОВ.**

Как пример самого простого набора данных рассмотрим встроенную базу данных в *scikit-learn* о цветках ириса, предполагая, что необходимо определить сорт цветка. Данные содержат в себе измерения о трех разных сортах цветов.

*Scikit-learn* имеет очень строгую структуру базы данных сортов ириса:

1) Признаки: длина чашелистника (sepal *length*, см), ширина чашелистника (sepal *width*, см), длина лепестка (*petal length*, см), ширина лепестка (*petal width*, см).
2) Классы: Setosa, Versicolour, Virginica.


Setosa


Versicolor


Virginica

*scikit-learn* имеет встроенную копию базы данных ирисов как CSV файл, который можно загрузить как *numpy* массив.

```
from sklearn.datasets import load_iris
iris = load_iris()
```

Признаки каждого объекта цветка сохранены как атрибуты набора данных.

```
print(iris.data.shape)
n_samples, n_features = iris.data.shape
print(n_samples)
print(n_features)
print(iris.data[0])
```

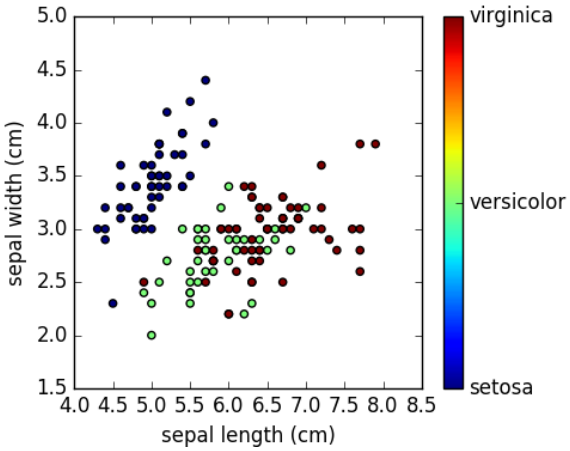
Информация об каждом классе хранится в атрибуте класса.

```
print(iris.target.shape)
print(iris.target)
```

Названия классов сохранены в последнем атрибуте *target_names*.

```
print(iris.target_names)
```

Этот набор данных четырехмерный, однако их можно визуализировать как двухмерный, используя диаграмму рассеяния.




**Задание 1**: Реализуйте код, приведенный в примере. Постройте график. Выберите два наиболее оптимальных признака для разделения объектов на классы.

**Базовые принципы машинного обучения**

Каждый алгоритм в scikit-learn реализован как объект. Например, алгоритм линейно регрессии:

```
from sklearn.linear_model import LinearRegression
```

Все параметры классификатора устанавливаются еще при его инициализации.

```
model = LinearRegression(normalize=True)
print(model.normalize)
print(model)
```

Создадим набор данных, используя numpy:

```
import numpy as np
x = np.array([0, 1, 2])
y = np.array([0, 1, 2])
X = x[:, np.newaxis]
print(X)
model.fit(X, y)
```

Когда модель обучена на данных, то ее параметры оценены по ней. Все параметры, полученные в результате оценки, выводятся с подчеркиванием.

```
print(model.coef_)
```

**Задание 2**: реализуйте код, приведенный в примере.

**Обучение с учителем.**

В машинном обучении с учителем мы работаем с набором данных, который содержит как набор признаков, так и меток. Задача сводится к тому, чтобы модель могла предсказывать метки объектов по их признакам. В качестве примера можно рассмотреть случай, когда строится предсказание о сорте ириса по измерениям его цветка. В качестве более сложных задач можно рассмотреть:

1) По изображению с телескопа определить, где находится звезда, квазар или галактика.
2) По фотографии определить личность человека
3) Определить фильм для просмотра согласно списку уже просмотренных фильмов

Общим для всех этих задач является то, что есть один или более признаков объектов, которые необходимо определить на основе других признаков.

В целом обучение с учителем делится на две задачи: классификация и регрессия. В задаче классификации, метки дискретны, в то время как в задачах регрессии метки непрерывны. Например, в астрономии, определение того, что объект является звездой, квазаром или галактикой – это задача классификации: один из трех маркеров. С другой стороны, можно попробовать оценить возраст на основе наблюдений, в этом случае, это будет задача регрессии, т.к. возраст – это непрерывная величина.

Классификация: Метод k-ближайших соседей – один из самых простых алгоритмов обучения: объект относится к тому классу, к которому принадлежит ближайший к нему соседний объект. Пример рассмотрен ниже:

```
from sklearn import neighbors, datasets
iris = datasets.load_iris()
X, y = iris.data, iris.target
knn = neighbors.KNeighborsClassifier(n_neighbors=1)
knn.fit(X, y)
#Какой сорт имеет 3cm x 5cm
#чашелистник и 4cm x 2cm лепесток?
print(iris.target_names[knn.predict([[3, 5, 4, 2]])])
```

Ниже приведен код для отображения результатов классификации.

```
import matplotlib.pyplot as plt
x_min, x_max = X[:, 0].min() - .1, X[:, 0].max() + .1
y_min, y_max = X[:, 1].min() - .1, X[:, 1].max() + .1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# Построение обучающих точек
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold)
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
plt.axis('tight')
```

Библиотека *Scikit-learn* старается иметь общий интерфейс для всех классификаторов, рассматривая каждый классификатор как объект (*model*), следующие методы доступны:

**model.fit()**: обучение. Для решения задач машинного обучения с учителем, принимает два аргумента: данные *X* и метки *y* (*model.fit(X, y)*). Для задач машинного обучения без учителя этот метод принимает только один аргумент (*model.fit(X)*).

Для метод машинного обучения с учителем:

**model.predict()**: на основе обученной модели предсказывает метки для нового набора данных. Этот метод принимает только один аргумент *X_new* (*model.predict(X_new)*) и возвращает метки для каждого объекта в массиве.

**model.predict_proba()**: В случае решения задачи классификации некоторые методы возвращают вероятность того к какому классу относится объект. В этом случае метод *model.predict()* возвращает ту метку, у которой максимальное значение вероятности.

**model.score()**: этот метод используется для решения задачи классификации и кластеризации, он позволят оценивать качество обучения модели используя оценку от 0 до 1. Чем выше оценка, тем лучше качество.

При обучении без учителя (кластеризация):

**model.transform()**: строит модель без учителя, преобразовывает данные в новый формат. Аналогично принимает один аргумент *X_new* и возвращает представление данные на основе модели без учителя.

**model.fit_transform()**:некоторые модели использует этот метод, т.к. он показывает более эффективный результат аппроксимации и преобразования данных.

**Задание 3:** Реализуйте код, приведенный в примере.

**Задача регуляризации.**

Регуляризация в статистике, машинном обучении, теории обратных задач — метод добавления некоторой дополнительной информации к условию с целью решить некорректно поставленную задачу или предотвратить переобучение.

Ошибка обучения. Допустим используется классификатор, который использует только один соседний объект для классификации, какие ошибки могут возникнуть при этом:

1) Набор ошибок, возникших при обучении в этом случае не является хорошим критерием качества классификации. В этом случае использовать метод перекрестной проверки.
2) В общем случае необходимо подсчитать все ошибки, произошедшие при обучении.

Регуляризацию можно рассматривать следующее упрощение, что мы считаем оптимальной ту «простую» модель, которая подходит большинству данных, несмотря на то, что она ведет к большей ошибке, чем более сложная модель

В качестве примера рассмотрен полином девятой степени с добавлением шума.

```
import numpy as np
rng = np.random.RandomState(0)
x = 2*rng.rand(100) - 1
f = lambda t: 1.2 * t**2 + .1 * t**3 - .4 * t **5 - .5 * t ** 9
y = f(x) + .4 * rng.normal(size=100)
```

Затем сгладим эти данные, используя полиномы 4-ой и 9-ой степени.

```
from sklearn import linear_model
plt.figure(figsize=(6, 4))
plt.scatter(x, y, s=4)

x_test = np.linspace(-1, 1, 100)
X = np.array([x**i for i in range(5)]).T
X_test = np.array([x_test**i for i in range(5)]).T
regr = linear_model.LinearRegression()
regr.fit(X, y)
plt.plot(x_test, regr.predict(X_test), label='4th order')

X = np.array([x**i for i in range(10)]).T
X_test = np.array([x_test**i for i in range(10)]).T
regr = linear_model.LinearRegression()
regr.fit(X, y)
plt.plot(x_test, regr.predict(X_test), label='9th order')
```

Регуляризация повсеместно используется в машинном обучении. Большинство моделей машинного обучения, реализованные в *scikit-learn*, имеют параметры для ее использования. Например, в методе k-ближайших соседей, в качестве этого параметра можно использовать количество соседей для классификации. Большее количество соседей позволяет построить более гибкие границы.

**Задание 4:** Реализуйте код, приведенный в примере.

**Обучение с учителем: классификация рукописных цифр.**

Рассмотрим использование библиотеки *scikit-learn* для решения задачи классификации рукописных цифр, учитывая некоторые метрики, которые позволяют влиять на качество классификации.

```
from sklearn.datasets import load_digits
digits = load_digits()
```

Выполним визуализацию загруженных данных

```
from matplotlib import pyplot as plt
fig = plt.figure(figsize=(6, 6))  # figure size in inches
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)
for i in range(64):
    ax = fig.add_subplot(8, 8, i + 1, xticks=[], yticks=[])
    ax.imshow(digits.images[i], cmap=plt.cm.binary, interpolation='nearest')
```

Визуализация данных методом главных компонент. Хорошим первым шагом для решения многих задач прикладного анализа является визуализация данных используя методы снижения размерности. Одним из самых распространенных среди них является метод главных компонент PCA (Principal Component Analysis). Метод главных компонент ищет ортогональную линейную комбинацию признаков, которая имеет максимальную дисперсию, что позволяет получить представление об общей структуре данных.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
proj = pca.fit_transform(digits.data)
plt.scatter(proj[:, 0], proj[:, 1], c=digits.target)

plt.colorbar()
```

**Вопрос:** по отображению на первые две главные компоненты, предположите какие цифры будет отличить друг от друга сложнее всего, а какие проще всего.

**Наивный байесовский классификатор**

Для решения большинства задач классификации было бы удобно использовать простой и быстрый метод классификации, который позволял бы проводить примитивную классификацию. Если этот метод оказывается эффективным, то нет смысла использовать более сложные методы, загружая тем самым ЦПУ. Если нет, то его результаты можно использовать как подсказки о структуре данных. Одним из таких методов является Наивный Байесовский классификатор.

Наивный Гауссовский классификатор строит нормальное распределение для каждого класса независимо и использует их затем для «жесткой» классификации. На практике этот метод не подходит для работы с реальными данными, но он может оказаться на удивление успешным при работе с текстовыми данными.

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

# разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target)

# обучение модели
clf = GaussianNB()
clf.fit(X_train, y_train)


# прогнозирование с помощью обученной модели
predicted = clf.predict(X_test)
expected = y_test
print(predicted)

print(expected)
```

**Задание 5:** Реализуйте код, приведенный в примерах. Постройте изображение с результатами классификации по примеру, показанному на рисунке.

**Численная оценка качества классификации**

Необходимо оценить качество классификации без визуального представления, в этом случае можно использовать количество совпадений.

```
matches = (predicted == expected)
print(matches.sum())
print(len(matches))
k = matches.sum() / float(len(matches))
print(k)
```

Однако есть более интересные методы, реализованные в библиотеки *sklearn.metrics*, например, *classification_report*, который комбинирует несколько измерений и строит результаты.

```
from sklearn import metrics
print(metrics.classification_report(expected, predicted))
```

Вторым методом для отображения качества обучения является построение матрицы ошибок, которая позволяет оценить те метки, которые чаще всего принимаются за другие.

```
print(metrics.confusion_matrix(expected, predicted))
```

**Задание 6:** Реализуйте код, приведенный в примерах. Какие цифры чаще всего были приняты за цифру 8 при классификации?

**Обучение с учителем: задача регрессии.**

Для рассмотрения задачи регрессии будем использовать данные о продаже жилья в окрестности Бостона, которые доступны в библиотеке *scikit-learn*. Каждое жилье описывается 13 признаками, всего имеется 500 объектов.

```
from sklearn.datasets import load_boston
data = load_boston()
print(data.data.shape)

print(data.target.shape)
print(data.DESCR)
```

Целевым значением для исследования будет медианная стоимость. Хорошим шагом для начала работы будет построение ее гистограммы используя модуль *pyplot* библиотеки *matplotlib*.

```
plt.hist(data.target)
```

Для того, чтобы оценить какие параметры оказывают наибольшее влияние на параметр медианной стоимости необходимо построить диаграммы рассеяния по признакам.

```
for index, feature_name in enumerate(data.feature_names):
    plt.figure()
    plt.scatter(data.data[:, index], data.target)
```

Используя такой способ можно вручную отобрать значимые признаки.

В библиотеке *scikit-learn* имеется несколько различных методов регрессии, в нашем случае мы будем использовать самый простой – линейная регрессия на основе метода наименьших квадратов.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target)
from sklearn.linear_model import LinearRegression
clf = LinearRegression()
clf.fit(X_train, y_train)

predicted = clf.predict(X_test)
expected = y_test
print("RMS: %s" % np.sqrt(np.mean((predicted - expected) ** 2)))
plt.scatter(expected, predicted)
```

**Задание 7**: реализуйте код, приведенный в примере. Дополнительно решите задачу регрессии, используя *gradient boosted tree*. Сравните результаты.

**Оценка качества предсказания**

Простой тест для метода k-средних.

```
# Скачивание данных
from sklearn.datasets import load_digits
digits = load_digits()
X = digits.data
y = digits.target
# Инициализация и обучение
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=1)
clf.fit(X, y)
# Проверка результатов
from sklearn import metrics
y_pred = clf.predict(X)
print(metrics.confusion_matrix(y_pred, y))
```

Согласно полученным результатам может показаться, что был найден идеальный классификатор. Однако, это не так недостатком этой модели является, что она запоминает все объекты, которые использовались для обучения. Для того, чтобы на самом деле оценить насколько качественно обучен алгоритм необходимо использовать те данные, которые не использовались для обучения. Аналогичную проблему можно встретить и при решении задачи регрессии.

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.tree import DecisionTreeRegressor
data = load_boston()
clf = DecisionTreeRegressor().fit(data.data, data.target)
predicted = clf.predict(data.data)
expected = data.target
plt.scatter(expected, predicted)
plt.plot([0, 50], [0, 50], '--k')
```

Использовать для обучения и тестирования один и тот же набор данных – это методологическая ошибка. Модель будет только воспроизводить метки образцов, которые она видела, тем самым получая превосходную оценку на них, однако будет не способна работать на новом наборе данных.

Для того чтобы избежать переобучения существует два различных набора:

1) Обучающая выборка X_train, y_train, которая используется для обучения модели
2) Тестовая выборка, которая используется для оценки качества обученной модели.

В библиотеке *scikit-learn* это разбиении данных может быть сделано с помощью функции *train_test_split()*:

```
from sklearn import model_selection
from sklearn.datasets import load_digits
digits = load_digits()
X = digits.data
y = digits.target
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
                                    test_size=0.25, random_state=0)
print("%r, %r, %r" % (X.shape, X_train.shape, X_test.shape))
```

Теперь необходимо обучить модель на обучающей выборке, оценить предсказания на модели с метками в тестовой выборке. Среднее значение единичной меры (f1-score) часто используется для

оценки качества классификации алгоритмом. Ее можно найти в нижней строке отчета, а также вызвать напрямую. В случае переобучения она будет принимать значение равное 1.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
clf = KNeighborsClassifier(n_neighbors=1).fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(metrics.confusion_matrix(y_test, y_pred))
print(metrics.classification_report(y_test, y_pred))
print(metrics.f1_score(y_test, y_pred, average="macro"))
print(metrics.f1_score(y_train, clf.predict(X_train), average="macro"))
```

**Задание 8**. Реализуйте код, приведенный в примере.

Выбор оптимальной модели

После того как последовательно были применены три различные модели для классификации, возникает вопрос о том, какая из них лучше всего работает на нашем наборе данных. Устанавливая различные параметры алгоритмов классификации, например, *n_neighbors* в *clf = KNeighborsClassifier(n_neighbors=1),* будут получены различные значения для меры *f1*.

```
from sklearn import model_selection, datasets, metrics
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier

digits = datasets.load_digits()
X = digits.data
y = digits.target
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
                            test_size=0.25, random_state=0)

for Model in [LinearSVC, GaussianNB, KNeighborsClassifier]:
    clf = Model().fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print('%s: %s' %
          (Model.__name__, metrics.f1_score(y_test, y_pred, average="macro")))

print('-------------------')

# test SVC loss
for loss in ['l1', 'l2']:
    clf = LinearSVC(loss=loss).fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print("LinearSVC(loss='{0}'): {1}".format(loss,
          metrics.f1_score(y_test, y_pred, average="macro")))

print('-------------------')

# test the number of neighbors
for n_neighbors in range(1, 11):
    clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print("KNeighbors(n_neighbors={0}): {1}".format(n_neighbors,
        metrics.f1_score(y_test, y_pred, average="macro")))
```

**Задание 9.** Реализуйте код, приведенный в примере.

**Перекрестная проверка**

Перекрестная проверка заключается в чередующемся повторении разбиения данных на группы обучающая и тестовая, называемые *‘folds’*. В библиотеке *Scikit-learn* существует специальная функция, которая автоматически рассчитывает *‘folds’*. Далее рассмотрен пример, где $k = 5$. При этом поддерживаются различные концепции, например, случайное разбиение:

```
clf = KNeighborsClassifier()
from sklearn.model_selection import cross_val_score
cross_val_score(clf, X, y, cv=5)
from sklearn.model_selection import ShuffleSplit
cv = ShuffleSplit(n_splits=5)
cross_val_score(clf, X, y, cv=cv)
```

Перекрестную проверку можно использовать для нахождения параметров модели.

Рассмотрим две линейные модели: Метод регуляризации Тихонова, у которой 12 параметров. и Лассо регрессии, где 11 признаков. Значения этих параметров сильно влияют на качество сглаживания. Для исследования будем рассматривать базу данных о диабетических больных, которая содержит 10 переменных (возраст, пол, вес, кровяное давление) измеренных на 442 пациентах, наблюдаемых во времени в течении года. С параметрами, установленными по умолчанию, рассчитаем значение для метода перекрестной проверки.

```
from sklearn.datasets import load_diabetes
data = load_diabetes()
X, y = data.data, data.target
print(X.shape)
from sklearn.linear_model import Ridge, Lasso
for Model in [Ridge, Lasso]:
    model = Model()
    print('%s: %s' % (Model.__name__, cross_val_score(model, X, y).mean()))
```

Рассмотрим изменения значения перекрёстной проверки по методу Тихонова и методу Лассо как зависимость от параметра α, где возьмем двадцать значений на интервале от 0.0001 до 1.

```
alphas = np.logspace(-3, -1, 30)
for Model in [Lasso, Ridge]:
    scores = [cross_val_score(Model(alpha), X, y, cv=3).mean()
              for alpha in alphas]
    plt.plot(alphas, scores, label=Model.__name__)
```

Одним из методов подбора параметров является сеточный метод, реализованные в функции *sklearn.grid_search.GridSearchCV*.

```
from sklearn.grid_search import GridSearchCV
for Model in [Ridge, Lasso]:
    gscv = GridSearchCV(Model(), dict(alpha=alphas), cv=3).fit(X, y)
    print('%s: %s' % (Model.__name__, gscv.best_params_))
```

Для больших наборов данных реализованы отдельные методы для поиска оптимальных параметров для модели Тихонова и Лассо. Тогда поиск параметров может быть реализован как:

```
from sklearn.linear_model import RidgeCV, LassoCV
for Model in [RidgeCV, LassoCV]:
    model = Model(alphas=alphas, cv=3).fit(X, y)
    print('%s: %s' % (Model.__name__, model.alpha_))
```

**Задание 10.** Реализуйте код, приведенный в примере.

**Машинное обучение без учителя**

В случае обучения без учителя отсутствует вектор y, содержащий метки классов. Методы снижения размерности строят новый набор искусственных признаков, которых может быть меньше, чем в исходном наборе. В этом примере будут использовать метод главных компонент для набора цветов ириса. Метод главных компонент строит линейную комбинацию, используя сингулярное разложение матрицы *X*, реализованный в функции *sklearn.decomposition.PCA*:

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target

from sklearn.decomposition import PCA
pca = PCA(n_components=2, whiten=True)
pca.fit(X)
print(pca.components_)
print(pca.explained_variance_ratio_)
X_pca = pca.transform(X)
print(X_pca.shape)

from matplotlib import pyplot as plt
target_ids = range(len(iris.target_names))
for i, c, label in zip(target_ids, 'rgbcmykw', iris.target_names):
    plt.scatter(X_pca[y == i, 0], X_pca[y == i, 1],
                c=c, label=label)
```

**Задание 11.** Реализуйте код, приведенный в примере.