Цель лабораторных работ — получение практических навыков решения задач машинного обучения и интеллектуального анализа данных.

Содержание ЛР
ЛР 1. Знакомство с инструментами Python для анализа данных. Введение в NumPy, SciPy, Pandas, Scikit-Learn.
ЛР 2. Работа с деревьями решений.
ЛР 3. Работа с линейной регрессией.
ЛР 4. Работа с логистической регрессией.
ЛР 5. Метод опорных векторов.
ЛР 6. Кластеризация методом k-средних.

Описание процесса выполнения лабораторных работ
Каждая лабораторная работа, за исключением первой, направлена на знакомство с одним из методов интеллектуального анализа данных. Лабораторная работа делится на два этапа: знакомство с методом/инструментом и задания для самостоятельного выполнения.
В разделе знакомства с методом приводятся его теоретические основы, а также инструменты и примеры для его обучения/использования. Не все функции и их атрибуты указаны при описании метода, однако, для ознакомления с ними, можно воспользоваться документацией библиотек, ссылки на которые будут даны.
Для успешной сдачи лабораторной работы необходимо выполнить задание для самостоятельного решения (выделено синим в тексте ЛР), иметь устное объяснение выполненных работ, а также быть готовым выполнить контрольное задание. В процессе устного ответа обучающийся делает необходимые комментарии к заданию и отвечает на уточняющие и дополнительные вопросы экзаменатора. Преподавателю предоставляется право задавать обучающемуся по программе курса дополнительные вопросы.

Лабораторная работа 1. Знакомство с инструментами python для анализа данных. Введение в NumPy, SciPy, Pandas, Scikit-Learn.

Инструкции по установке основных компонент.
Предпочтительный способ среды выполнения лабораторных работ — использование Google Colaboratory. В нее можно перейти по следующей ссылке:
https://colab.research.google.com/notebooks/welcome.

Здесь и далее будет описано начало работы с Google Colab. Если исполнитель лабораторной работы предпочёл выполнять работу, используя локальные инструменты, инструкции по использованию какого-либо SDK можно найти в соответствующей документации.

Google Colab позволяет вести работу в Jupyter Notebook (https://jupyter.org). Jupyter Notebook представляет собой не статическую страницу, а интерактивную среду, которая позволяет писать и выполнять код на Python и других языках. Отличительной особенностью написания кода в блокноте Jupyter является то, что код разбивается на ячейки (cell), каждая из которых может быть выполнена отдельно. Файлы, создаваемые и редактируемые в блокноте Jupyter имеют расширении .ipynb. Для начала работы можно запустить код, представленный на первой странице лаборатории. Для этого нужно нажать на стрелку в соответствующей ячейке или выделить ячейку и нажать ctrl+enter. Для выполнения кода последовательно во всех ячейках можно воспользоваться сочетанием клавиш ctrl+F9.

Для создания нового файла вашей программы нужно перейти в меню File- >New Python 3 notebook. В новой вкладке откроется файл Untitled.ipynb, с которым вы будете работать далее.Переименовать файл можно выбрав в меню File→ Rename. Открытие ранее созданных проектов осуществляется выбором File→Open Notebook. Появится диалоговое окно, отображающее недавно открытые файлы.
Для того чтобы сохранить файл с новым именем на жесткий диск необходимо в меню выбрать File->download .ipynb. После выполнения лабораторной работы нужно будет сохранять файлы на диске.

Для добавления новой ячейки выберите Insert->Code cell. При наведении курсора мыши на ячейку кода, справа вверху появляется всплывающее меню (рис. 6). Нажатие на значок «сообщение» позволяет добавить комментарий к ячейке. Нажатие на корзину – удалить ячейку. Нажатие на «шестеренку» даст возможность редактирования настроек данной ячейки.

Для ознакомления с функциями языка Python можно воспользоваться документацией по языку Python: https://docs.python.org/3/tutorial/

Помните про отступы. В Python пробелы важны. Точнее, пробелы в начале строки важны. Это называется отступами. Передние отступы (пробелы и табуляции) в начале логической строки используются для определения уровня отступа логической строки, который, в свою очередь, используется для группировки предложений.
Это означает, что предложения, идущие вместе, должны иметь одинаковый отступ. Каждый такой набор предложений называется блоком. Вы должны запомнить, что неправильные отступы могут приводить к возникновению ошибок.

Элементарные операции с данными

Рассмотрим примеры работы с элементарными операциями с данными средствами Numpy и Pandas.

Здесь и далее будет описано только некоторые возможности рассматриваемых библиотек. Для подробного ознакомления с функциями посетите страницы с документацией Pandas:
https://pandas.pydata.org/docs/index.html
и документацией Numpy:
https://numpy.org/doc/stable/index.html.

Для импорта модуля Numpy необходимо написать следующую строку кода:

```
import numpy as np
```

В этом случае Numpy импортируется под псевдонимом np, что в дальнейшем упростит обращение к модулю.

Основные единицы данных в машинном обучении — вектора и матрицы. С точки зрения кода программы, они представляются в виде одномерных и двумерных массивов данных или специальных объектов-таблиц.
Рассмотрим основные способы получения данных: прямое объявление, случайная генерация и загрузка из csv-формата.

1. Прямое объявление:

```
X = np.array([[2,3,4], [15,30,26]])
```

Обратите внимание. В Python не требуется предварительно объявлять данные и их тип, а также не требуется предварительное выделение памяти под данные.

2. Случайная генерация:

Сгенерируем случайную, нормально распределенную матрицу, состоящую из 4 строк и 5 столбцов. Элементы ее будут являться случайными числами, имеющим нормальное распределение, со средним равным 1 и стандартным отклонением равным 5. Для этого выполним следующий код:

```
X = np.random.normal(loc = 1, scale = 5, size = (4,5))
```

Доступные генераторы псевдослучайных чисел и их распределения можно также найти в документации.

3. Загрузка данных из cvs-файла:

Как правило, в файлах такого формата столбцы разделяются запятой, а первая строка содержит их имена.
Допустим у нас есть файл `proceeds.csv`, содержащий данные о месячной выручке по отделам компании за текущий год.

Перед чтением и загрузкой файлов, расположенных локально, в Google Colab необходимо выполнить следующий код:

```
from google.colab import files
uploaded = files.upload()
```

Для загрузки данных воспользуемся средствами библиотеки pandas. Используя следующий код, вы можете импортировать pandas, загрузить файл и распечатать загруженные данные. Тут же представлен код для записи данных в csv-файл.

```
# -*- coding: utf-8 -*-
import csv
import pandas
#чтение из файла
data = pandas.read_csv('proceeds.csv', index_col='Month')
print(data)
#запись в файл данных в исходном виде
data.to_csv('proceeds_q.csv')
#запись в файл данных как массива значений
with open('proceeds_w.csv', 'w') as csvfile:
```

```
    spamwriter = csv.writer(csvfile, delimiter=';',quotechar='|',
quoting=csv.QUOTE_MINIMAL)
    spamwriter.writerow (data.values)
```

В данном коде представлены два способа записи в файл. Первый — с использованием DataFrame, второй — с использованием метода библиотеки csv.

При выполнении кода выше, данные будут загружены в виде DataFrame, с помощью которого можно удобно работать с ними. DataFrame представляет собой двумерную, изменяемую по размеру, потенциально гетерогенную структуру табличных данных с маркированными осями (строками и столбцами).

После того, как мы научились загружать и сохранять данные, перейдем к демонстрации некоторых возможностей по операциям с матрицами на конкретном примере.
Предположим, перед нами стоит следующая задача: имеются данные о месячной выручке по отделам компании за текущий год, записанные в виде таблицы:

Месяц	Отдел1	Отдел2	Отдел3	Отдел4
1	100	18	454	68
2	32	54	545	11
3	12	11	14	17
4	54	54	121	31
5	333	11	27	8
6	123	33	24	5
7	11	14	17	11
8	111	121	111	211
9	100	11	74	15
10	64	65	68	11
11	75	58	98	44
12	113	211	118	144

Данные сохранены в файл proseeds.csv.

Для того, чтобы посмотреть, что из себя представляют данные, можно воспользоваться несколькими способами:

1. Если мы хотим распечатать первые 4 строки данных, то можем воспользоваться следующим кодом:
```
import pandas
data = pandas.read_csv('proceeds.csv', index_col='Month')
print(data[:4])
```

2. Также мы можем воспользоваться методом head() DataFrame:
```
import pandas
data = pandas.read_csv('proceeds.csv', index_col='Month')
print(data.head(4))
```

3. Если нас интересует содержимое конкретного столбца, то можно использовать имя столбца в квадратных скобках:
```
import pandas
```

```
data = pandas.read_csv('proceeds.csv', index_col='Month')
print(data['Department2'])
```

4. Для подсчета некоторых статистик (среднее, максимум, минимум, среднее) можно использовать как методы объекта DataFrame, так и возможности библиотеки NumPy по операциям с матрицами.

   Например, нам необходимо вывести максимальную выручку для второго отдела, а также месяц, в которой она была достигнута. Для этого воспользуемся функциями max() и idxmax():

```
import pandas
data = pandas.read_csv('proceeds.csv', index_col='Month')
print(data['Department2'].max())
print(data['Department2'].idxmax())
```

   Значения суммы, минимума, среднего, количества и среднеквадратичного отклонения вычисляются аналогично с использованием соответствующих функций (sum, min, mean, count, std).

   Более подробно со списком методов DataFrame можно ознакомиться в документации: http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html.

5. С данными мы также можем работать, как с матрицами. Перед этим необходимо перевести данные из DataFrame в обычную матрицу и работать с ними, как с обычным двумерным массивом.

   Перевод осуществляется следующим образом:

```
x_test= data[['Department1', 'Department2','Department3','Department4']].values
```

   Предположим, что перед нами стоит задача: вывести имена отделов, годовая прибыль которых больше 800.
   С точки зрения работы с матрицей, для получения ответа, необходимо вычислить сумму в каждом столбце и вывести имена тех, чья сумма превысит 800. Функция для подсчета суммы np.sum. В качестве параметров она принимает матрицу, для которой необходимо посчитать сумму и измерение (строки или столбцы), которое необходимо суммировать. Измерение (axis) задается цифрой (для двумерной матрицы 0 – строки, 1 – столбцы). Если этот параметр не задать, то результат функции будет рассчитан для всей матрицы целиком. Результатом выполнения операции будет массив с соответствующими суммами. Библиотека Numpy предоставляет возможности применения к матрицам (и массивам) логических операций, причем применяемых поэлементно. Соответственно, результатом такой операции будет матрица такого же размера, в ячейках которой будет записано либо True, либо False (удовлетворяет текущий элемент условию или нет). Индексы элементов со значением True можно получить с помощью функции np.nonzero. Функция в качестве параметра принимает матрицу, в которой необходимо отыскать ненулевые элементы. Заметим, что в нашем случае мы можем сразу передать логическое выражение, составленное из массива сумм и самого условия, тогда элементы со значением False будут интерпретироваться как нулевые.

   Код решения:

```
import pandas
import numpy as np
data = pandas.read_csv('proceeds.csv', index_col='Month')
```

```
x_test= data[['Department1',
'Department2','Department3','Department4']].values
r = np.sum(x_test, axis=0)
print(data.columns.values[np.nonzero(r> 800)])
```

Еще одна библиотека, которую мы будем активно использовать называется Scikit-learn — это библиотека машинного обучения с открытым исходным кодом, которая поддерживает обучение с учителем и без. Он также предоставляет различные инструменты для подбора модели, предварительной обработки данных, выбора и оценки модели и многие другие утилиты.

Для подробного ознакомления с функциями посетите страницу с документацией:
https://scikit-learn.org/stable/user_guide.html

Scikit-learn предоставляет десятки встроенных алгоритмов и моделей машинного обучения. Примеры использования будут показаны при разборе соответствующих методов машинного обучения в следующих лабораторных работах.

Задание для самостоятельного выполнения:
1. Объявите и выведите на экран матрицу:

```
 4  3 18
22 11  2
 0 13  3
```

2. Сгенерируйте и выведите на экран случайную нормально распределенную матрицу размером 6x3, со средним равным 5 и отклонением 1.

3. Имеются данные о ежемесячной выручке по отделам компании за текущий год, записанный в файле proseeds21.csv. Выведите следующие данные: в какой месяц суммарная выручка по отделам превысила 400, имя отдела с максимальной средней выручкой за год, имя отдела с максимальной величиной стандартного отклонения прибыли.

Лабораторная работа 2. Работа с деревьями решений.

Решающие деревья относятся к классу логических методов. Их основная идея состоит в объединении определенного количества простых решающих правил, благодаря чему итоговый алгоритм является интерпретируемым. Как следует из названия, решающее дерево представляет собой бинарное дерево, в котором каждой вершине сопоставлено некоторое правило вида «j-й признак имеет значение меньше b». В листьях этого дерева записаны числа-предсказания. Чтобы получить ответ, нужно стартовать из корня и делать переходы либо в левое, либо в правое поддерево в зависимости от того, выполняется правило из текущей вершины или не выполняется.
Одна из особенностей решающих деревьев заключается в том, что они позволяют получать важности всех используемых признаков. Важность признака можно оценить на основе того, как сильно улучшился критерий качества благодаря использованию этого признака в вершинах дерева.

В библиотеке scikit-learn решающие деревья реализованы в классах sklearn.tree.DecisionTreeClassifier (для классификации) и sklearn.tree.DecisionTreeRegression (для регресии). Обучение модели производится с помощью функции fit.

См. https://scikit-learn.org/stable/modules/tree.html

Пример использования:

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
X = np.array([[1,2], [3,4], [5,6]])
y = np.array([0,1,0])
clf = DecisionTreeClassifier()
clf.fit(X,y)
```

Для нахождения важности признаков можно воспользоваться уже обученным классификатором:

```
importances = clf.feature_importances_
```

Переменная `importances` будет содержать массив «важности» признаков. Индекс в этом массиве соответствует индексу признака в данных.

Обратите внимание. Данные могут содержать пропуски. Pandas хранит такие значения в виде Nan. Для того, чтобы проверить, является ли число Nan-ом, можно воспользоваться функцией np.isnan().

Задание для самостоятельного выполнения:

1. Загрузите выборку данных abalone21.csv. Это датасет, в котором требуется предсказать возраст ракушки (число колец) по физическим измерениям.
2. Найдите все объекты, у которых есть пропущенные значения и удалите их из выборки. (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html#pandas.DataFrame.dropna)
3. Оставьте в выборке четыре признака: пол, диаметр, длина и высота.
4. Преобразуйте признак Sex в числовой: значение F должно перейти в -1, I — в 0, M — в 1.
5. Выделите целевую переменную. Она записана в последнем столбце.
6. Обучите решающее дерево с параметром random_state = 241 и остальными параметрами по умолчанию.
7. Вычислите важности признаков и найдите два признака с наибольшей важностью. В качестве ответа укажите имена признаков, а также значение важности.

Лабораторная работа 3. Работа с линейной регрессией.

В данной лабораторной работе мы познакомимся с линейной регрессией.

Для начала определим, с какими данными мы будем работать. Scikit-learn устанавливается вместе с несколькими стандартными выборками данных, например, iris и digits для классификации, и boston house prices dataset для регрессионного анализа.

см. https://scikit-learn.org/stable/datasets.html#datasets

Для примера рассмотрим набор данных Boston Housing. Задача, связанная с этим набором данных, заключается в том, чтобы спрогнозировать медианную стоимость домов в нескольких районах Бостона в 70-е годы на основе такой информации, как уровень преступности, удаленность от радиальных магистралей и т. д. Набор данных содержит 506 точек и 13 признаков.

Для загрузки данного набора используется следующий код:

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

Более подробную информацию о наборе данных вы можете получить, прочитав атрибут DESCR.
Для начала рассмотрим традиционную линейную регрессию. Искомая функция имеет вид:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \ldots + w_p x_p$$

Самый частый способ нахождения параметров линейно регрессии — метод наименьших квадратов, при использовании которого параметры *w* вычисляются таким образом, чтобы минимизировать среднеквадратическую ошибку (mean squared error) между спрогнозированными и фактическими ответами y в обучающем наборе. Среднеквадратичная ошибка равна сумме квадратов разностей между спрогнозированными и фактическими значениями. Линейная регрессия проста, что является преимуществом, но в то же время у нее нет инструментов, позволяющих контролировать сложность модели.

Более подробно см.
https://scikit-learn.org/stable/modules/linear_model.html

Ниже приводится программный код, который строит модель линейной регрессии для бостонских домов:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
...
X_train, X_test, y_train, y_test = train_test_split(data.data,
data.target, random_state=0)
lr = LinearRegression().fit(X_train, y_train)
```

Разделение на обучающую и тестовую выборки, кросс-вадидация: см.
https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation

Давайте посмотрим правильность модели на обучающем и тестовом наборах:

```
print("Score on the train set: {:.2f}".format(lr.score(X_train, y_train)))
print("Score on the test set: {:.2f}".format(lr.score(X_test, y_test)))
```

Аналогично мы можем вычислять другие метрики качества обученной модели:

```
from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score, make_scorer
import numpy as np
...
preds = lr.predict(X_test)
mse = mean_squared_error(y_test, preds)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, preds)
r2 = r2_score(y_test, preds)
print("RMSE: ", rmse)
print("MAE: ", mae)
print("R2: ", r2)
```

Подробнее о метриках: https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

Параметры «наклона» (*w*), также называемые весами или коэффициентами (*coefficients*), хранятся в атрибуте coef_, тогда как сдвиг (*offset*) или константа (*intercept*, хранится в атрибуте intercept_:

```
print("lr.coef_: {}".format(lr.coef_))
print("lr.intercept_: {}".format(lr.intercept_))
```

Значение R^2 в районе 0.64 указывает на не очень хорошее качество модели, однако можно увидеть, что результаты на обучающем и тестовом наборах очень схожи между собой. Большая разница между точностью на обучающем наборе и правильностью на тестовом наборе свидетельствовала бы о переобучении.

Подробнее о переобучении и недообучении: https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html?highlight=overfitting

Одна из наиболее часто используемых альтернатив стандартной линейной регрессии — гребневая регрессия. Гребневая регрессия также является линейной моделью регрессии, поэтому ее формула аналогична той, что используется в обычном методе наименьших квадратов. В гребневой регрессии коэффициенты (w) выбираются не только с точки зрения того, насколько хорошо они позволяют предсказывать на обучающих данных, они еще подгоняются в соответствии с дополнительным ограничением. Нам нужно, чтобы величина коэффициентов была как можно меньше. Другими словами, все элементы w должны быть близки к нулю. Это означает, что каждый признак должен иметь как можно меньшее влияние на результат (то есть каждый признак должен иметь небольшой регрессионный коэффициент) и в то же время он должен по-прежнему обладать хорошей прогнозной силой. Это ограничение является примером регуляризации (regularization). Регуляризация означает явное ограничение модели для предотвращения переобучения. Регуляризация, использующаяся в гребневой регрессии, известна как L2 регуляризация. Гребневая регрессия реализована в классе linear_model.Ridge

```
from sklearn.linear_model import Ridge
ridge = Ridge().fit(X_train, y_train)
print("Score in train set: {:.2f}".format(ridge.score(X_train, y_train)))
print("Score in test set: {:.2f}".format(ridge.score(X_test, y_test)))
```

Модель Ridge позволяет найти компромисс между простотой модели (получением коэффициентов, близких к нулю) и качеством ее работы на обучающем наборе. Компромисс между простотой модели и качеством работы на обучающем наборе может быть задан пользователем при помощи параметра alpha. В предыдущем примере мы использовали значение параметра по умолчанию alpha=1.0. Впрочем, нет никаких причин считать, что это даст нам оптимальный компромиссный вариант. Оптимальное значение alpha зависит от конкретного используемого набора данных. Увеличение alpha заставляет коэффициенты сжиматься до близких к нулю значений, что снижает качество работы модели на обучающем наборе, но может улучшить ее обобщающую способность.

Более подробно см.

https://scikit-learn.org/stable/modules/linear_model.html#ridge-regression-and-classification

Альтернативой Ridge как метода регуляризации линейной регрессии является Lasso. Как и гребневая регрессия, лассо также сжимает коэффициенты до близких к нулю значений, но несколько иным способом, называемым L1 регуляризацией. Результат L1 регуляризации заключается в том, что при использовании лассо некоторые коэффициенты становятся равны точно нулю. Получается, что некоторые признаки полностью исключаются из модели. Это можно рассматривать как один из видов автоматического отбора признаков. Получение нулевых значений для некоторых коэффициентов часто упрощает интерпретацию модели и может выявить наиболее важные признаки вашей модели.

Применим метод Лассо к набору данных бостонских домов:

```
from sklearn.linear_model import Lasso
lasso = Lasso().fit(X_train, y_train)
print("Score in train set: {:.2f}".format(lasso.score(X_train, y_train)))
print("Score in test set: {:.2f}".format(lasso.score(X_test, y_test)))
```

Как и Ridge, Lasso также имеет параметр регуляризации alpha, который определяет степень сжатия коэффициентов до нулевых значений.

Более подробно см.
https://scikit-learn.org/stable/modules/linear_model.html#lasso

На практике, когда стоит выбор между гребневой регрессией и лассо, предпочтение, как правило, отдается гребневой регрессии. Однако, если у вас есть большое количество признаков и есть основания считать, что лишь некоторые из них важны, Lasso может быть оптимальным выбором. Аналогично, если вам нужна легко интерпретируемая модель, Lasso поможет получить такую модель, так как она выберет лишь подмножество входных признаков.

Задание для самостоятельного выполнения:

1. Загрузите данные из стандартной выборки sklearn diabetes(). Разделите их на обучающую и тестовую выборки в соотношении 7 к 3. Random_state установите равным 241.
2. Обучите линейную регрессию с регуляризаторами Lasso и Ridge с с параметрами по умолчанию. Посмотрите, какое число коэффициентов близко к 0. Постройте график зависимости числа ненулевых коэффициентов от коэффициента регуляризации. (Подробнее о визуализации см. https://matplotlib.org/stable/tutorials/introductory/pyplot.html#sphx-glr-tutorials-introductory-pyplot-py)
3. Посчитайте для Ridge регрессии следующие метрики: RMSE, MAE, R^2.
4. Подберите на обучающей выборке для Ridge-регрессии коэффициент регуляризации для каждой из метрик. Для этого воспользуйтесь GridSearchCV и KFold из sklearn (число частей, на которые разбивается выборка равна 5, https://scikit-learn.org/stable/modules/grid_search.html#grid-search). Постройте графики зависимости функции потерь от коэффициента регуляризации. Посчитайте те же метрики снова для лучшего коэффициента регуляризации.
5. Как известно, MSE сильно штрафует за большие ошибки на объектах-выбросах. С помощью cross_val_predict (https://scikit-

learn.org/stable/modules/generated/sklearn.model_selection.cross_val_predict.html#sklearn.model_selection.cross_val_predict) сделайте Out-of-Fold предсказания для обучающей выборки. Посчитайте ошибки и посмотрите на их распределение (plt.hist).

6. Попробуйте удалить объекты-выбросы из обучающей выборки (что считать или не считать выбросами на ваше усмотрение). Обучите модель заново и посмотрите на качество на отложенной выборке (учитывайте, что там тоже могут быть выбросы, с которыми вы ничего не можете сделать). Стало ли лучше?

Лабораторная работа 4. Работа с логистической регрессией.

В предыдущей работе мы рассмотрели применение регрессии для решения задачи предсказания, теперь перейдем к задаче классификации и познакомимся с еще одной моделью — логистической регрессией. Ее особенностью является то, что результатом является вероятность принадлежности объекта к определенному классу, тогда как большинство линейных классификаторов могут выдавать только номера классов.
Логистическая регрессия для оценки качества использует функцию, которую не допускает записи решения в явном виде, тем не менее логистическую регрессию можно настраивать с помощью градиентного спуска.

Более подробно см. https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

Для демонстрации работы с логистической регрессией сгенерируем некоторую случайную выборку из 1000 элементов:

```
n_samples = 10000
np.random.seed(0)
X = np.random.normal(size=n_samples)
```

Для формирования множества откликов Y определим функцию, реализующую правило: Y=1 для всех X>0 и Y = 0 – в остальных случаях:

```
y = (X>0).astype(np.float)
```

Выполним преобразование исходного массива X путем добавления небольшого разброса и смещения, чтобы избавиться от прямой функциональной зависимости. Также преобразуем его в двумерный массив:

```
X[X>0] *=4
X+= 0.3*np.random.normal(size=n_samples)
X = X[:, np.newaxis]
```

Разделим выборку на обучающую и тестовую:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
```

Теперь мы можем объявить и обучить модель логистической регрессии:

```
lg = LogisticRegression()
lg.fit(X_train, y_train)
```

Следующим шагом нам необходимо выполнить предсказание. Оно будет выполняться методом predict. Он возвращает бинарные метки классов (0,1). Кроме этого, существует еще метод predict_proba. Он возвращает вероятность принадлежности объекта к классу 0 или 1 соответственно. Поскольку у нас два класса, то в результате predict_proba получается двумерный массив (один столбец отражает принадлежность к классу 0, а другой к классу 1).

```
preds = lg.predict_proba(X_test)[:,1]
```

Следующим шагом вычислим ROC-AUC и PR-AUC и построим ROC-кривую для оценки точности полученной модели:

```
roc = roc_auc_score(y_test, preds)
pr = average_precision_score(y_test, preds)
print("ROC-AUC: ", roc)
print("PR-AUC: ", pr)

fpr, tpr, threshold = roc_curve(y_test, preds)
plt.plot(fpr,tpr, color='r', label='Log Res')
plt.title('ROC curve for Log Res')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.grid()
plt.show()
plt.gcf().clear()
```

Поскольку выборка очень простая, то значения по умолчанию обеспечивают неплохое качество работы модели, однако на реальных данных, необходимо осуществлять подбор коэффициентов для его улучшения.

Задание для самостоятельного выполнения:

1. Загрузите набор данных по раку молочной железы. Используйте breast_cancer из стандартного набора sklearn.
2. Разделите на обучающую и тестовую.
3. Ортонормируйте признаки. Для этого используйте StandardScaler (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler).
4. Обучите логистическую регрессию со своими параметрами. Сделайте предсказание на тестовой части выборки.
5. Постройте ROC-кривую и Precision-Recall-кривую, вычислите ROC-AUC и PR-AUC. Какие выводы можно сделать?
6. Найдите порог бинаризации вероятностей, который позволяет обеспечить полноту (recall) не менее 0,9. Постройте матрицу ошибок для данного порога. Какие выводы вы можете сделать?
7. Проанализируйте влияние изменения значения параметра в методе логистической регрессии (анализируемый параметр выбирается самостоятельно) на качество классификации. Сделайте выводы.

Лабораторная работа 5. Метод опорных векторов

В ходе обучения SVM вычисляет важность каждой точки обучающих данных с точки зрения определения решающей границы между двумя классами. Обычно лишь часть точек

обучающего набора важна для определения границы принятия решений: точки, которые лежат на границе между классами. Они называются опорными векторами (support vectors) и дали свое название машине опорных векторов. Чтобы получить прогноз для новой точки, измеряется расстояние до каждого опорного вектора. Классификационное решение принимается, исходя из расстояний до опорных векторов, а также важности опорных векторов, полученных в процессе обучения (хранятся в атрибуте dual_coef_ класса SVC).

Подробнее см.
https://scikit-learn.org/stable/modules/svm.html#svm

Для демонстрации работы с машиной опорных векторов воспользуемся данными о пассажирах Титаника. Гибель Титаника является одним из самых печально известных кораблекрушений в истории. 15 апреля 1912 года во время своего первого плавания, Титаник затонул после столкновения с айсбергом, при этом погибло 1502 из 2224 пассажиров и членов экипажа. Некоторые группы людей имели больше шансов выжить, по сравнению с другими. Наша задача заключается в том, чтобы предсказать, какие пассажиры могли выжить в этой трагедии.

Загрузим данные из файла titanic.csv:

```
dataSrc = pd.read_csv('titanic.csv', index_col='PassengerId')
```

Набор содержит признак Survived для каждого пассажира, обозначающий, выжил данный пассажир или нет (0 для умерших, 1 для выживших).
Каждая строчка наборов данных содержит следующие поля:

- Pclass — класс пассажира (1 — высший, 2 — средний, 3 — низший);
- Name — имя;
- Sex — пол;
- Age — возраст;
- SibSp — количество братьев, сестер, сводных братьев, сводных сестер, супругов на борту титаника;
- Parch — количество родителей, детей (в том числе приемных) на борту титаника;
- Ticket — номер билета;
- Fare — плата за проезд;
- Cabin — каюта;
- Embarked — порт посадки (C — Шербур; Q — Квинстаун; S — Саутгемптон).

Выберем некоторые из них, а именно класс, стоимость билета, возраст и пол.

```
dataP = dataSrc[['Pclass', 'Fare', 'Age', 'Sex', 'Survived']]
```

Визуальное изучение данных показало, что поля Age и Fare могут принимать пустые значения. Удалим данные для пассажиров, значения которых не указаны:

```
dataP = dataP.dropna()
```

Поле Sex является категориальным. Мы можем значения male заменить на 0, значения female — на 1. Но для демонстрации работы с категориальными данными, которых может быть больше, чем два, воспользуемся типичным подходом для работы с ними: one-hot кодирование. Его идея заключается в том, что мы преобразуем категориальный признак при помощи

бинарного кода: каждой категории ставим в соответствие набор из нулей и единиц.

Для начала выберем столбцы с категориальными данными и преобразуем в dict:

```
categorical_columns = [c for c in data.columns if
data[c].dtype.name == 'object']
data2 = data2.astype(str)
data2 = data2.T.to_dict().values()
```

Теперь преобразуем данные таким образом, что вместо одного столбца пола появится два столбца, первый из которых характеризует метку отнесения пассажира к мужскому полу, а второй — к женскому. Для этого воспользуемся следующим кодом:

```
from sklearn.feature_extraction import DictVectorizer
...
transformer = DictVectorizer(sparse=False)
data2 = transformer.fit_transform(data2)
```

см. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html#sklearn.feature_extraction.DictVectorizer

Теперь удалим столбец пола из исходных данных и добавим к ним два новых столбца:

```
data1 = data.drop(['Sex'], axis=1)
X = np.hstack((data1, data2))
```

После того, как мы подготовили данные обучим классификатор, основанный на машине опорных векторов (SVM). SVM является одним из самых известных алгоритмов машинного обучения, применяемых в основном для задачи классификации. Также как и логистическая регрессия, SVM допускает многоклассовую классификацию методом one-vs-all.

```
#обучение
clf = SVC(probability=True, kernel='rbf', C = 1)
clf.fit(X_train, y_train)

#предсказание
preds = clf.predict_proba(X_test)[:,1]
roc = roc_auc_score(y_test, preds)
print("ROC-AUC: ", roc)
```

В результате мы получили классификатор, результат работы которого характеризуется значением площади под ROC-кривой равным примерно 0,75.

Задание для самостоятельного выполнения:

1. Загрузите данные из файла salary_part.csv. Решаем следующую задачу: по социальным признакам человека необходимо определить зарабатывает ли он более 50К или нет. Целевая переменная указана в последнем столбце. Если заработок более 50К, то целевая переменная — 1, 0 — в противном случае.
2. Разделите признаки на числовые и категориальные.
3. Ортонормируйте числовые признаки признаки (StandartScaler).
4. Выполните One-hot-кодирование категориальных признаков с помощью DictVectorizer.
5. Объедините измененные числовые и категориальные признаки.

6. Разделите данные на обучающую и тестовую выборки.
7. Обучите классификатор на этих данных методом опорных векторов с параметрами по умолчанию.
8. Постройте ROC-кривую и Precision-Recall-кривую, посчитайте ROC-AUC и PR-AUC. Какие наблюдения и выводы по ним можно сделать?
9. Подберите лучшие значения типа ядра и параметра регуляризации C для метода опорных векторов с помощью кросс-валидации по 5 блокам (GridSearchCV). В качестве метрики качества используйте AUC-ROC. По итогам кросс-валидации укажите лучшие значения параметров. Обучите классификатор с этим параметром на всей обучающей выборке и найдите качество (AUC-ROC) на тестовой выборке. Что вы можете сказать о полученных результатах? Как повлияло использование лучших значений параметра на результат?

Лабораторная работа 6. Кластеризация методом k-средних.

Кластеризация k-средних – один из самых простых и наиболее часто используемых алгоритмов кластеризации.

Подробнее см. https://scikit-learn.org/stable/modules/clustering.html#k-means

Для демонстрации работы алгоритма k-средних применим его к синтетическим данным.

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# генерируем синтетические двумерные данные
X, y = make_blobs(random_state=1)
```

После генерации данных создадим экземпляр класса Kmeans и зададим число выделяемых кластеров. Затем вызываем метод fit и передаем ему в качестве аргумента данные:

```
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
```

Обратите внимание. Если вы не зададите количество выделяемых кластеров, то значение n_clusters по умолчанию будет равно 8. При этом нет никаких конкретных причин, в силу которых вы должны использовать именно это значение.

Во время работы алгоритма каждой точке обучающих данных X присваивается метка кластера. Вы можете найти эти метки в атрибуте kmeans.labels_:

```
print("kmeans.labels_:\n{}".format(kmeans.labels_))
```

Поскольку мы задали три кластера, кластеры пронумерованы от 0 до 2. Кроме того, вы можете присвоить метки кластеров новым точкам с помощью метода predict. В ходе прогнозирования каждая новая точка назначается ближайшему центру кластера, но существующая модель не меняется. Запуск метода predict на обучающем наборе возвращает тот же самый результат, что содержится в атрибуте labels_:

```
print(kmeans.predict(X))
```

Вы можете увидеть, что кластеризация немного похожа на классификацию в том плане, что

каждый элемент получает метку. Однако нет никаких оснований утверждать, что данная метка является истинной и поэтому сами по себе метки не несут никакого априорного смысла.

Задание для самостоятельного выполнения:

1. Загрузите набор данных изобржений цифр. Используйте load_digits из стандартного набора sklearn. Сами изображения чисел в виде векторов хранятся в поле *data*, а истинные метки — в поле *target*. Сохраните изображения в переменную *X*, а истинные метки — *y*.
2. Задача кластеризации состоит в следующем: нужно построить разбиение всех объектов на *K* кластеров, где *K* задано заранее. В данном случае, в датасете присутствуют цифры от 0 до 9, поэтому *K* будет равно 10.
3. Для того, чтобы в дальнейшем оценивать качество получившейся кластеризации, будем использовать следующие метрики: Homogenity (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.homogeneity_score.html) — эта метрика основывается на использовании информации об истинных метках объектов. Она позволяет оценить, все ли объекты в кластере имеют одну и ту же метку. В качестве первого параметра ей передаются истинные метки класса, второй параметр — предсказанные. Обратите внимание, что данная метрика не является симметричной. Данная метрика принимает значения от 0 до 1, где 1 соответствует наилучшей кластеризации. Silhouette Coefficient (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html) — эта метрика, в отличии от предыдущей, оперирует только с исходной матрицей "объект-признак" и предсказанными метками. Она позволяет получить информацию насколько "хорошей" получилась кластеризация с точки зрения расположения объектов в кластере, оценивая, насколько далеко они находятся друг от друга. Данная метрика принимает значения от -1 до 1, где 1 соответствует лучшей кластеризации, 0 говорит о том, что есть перекрывающиеся кластера.
4. Создайте объект *KMeans*, имеющий следующие интересные для нас параметры: количество кластеров и *random_state* (его следует зафиксировать для воспроизводимости результата).
5. Вызовите метод *fit_predict*, передав на вход переменную *X* и сохранив результаты в переменную.
6. Теперь вычислите значения указанных выше метрик для полученной кластеризации. Хорошее ли качество имеет данная кластеризация, исходя из интерпретации метрик?
7. Иногда полезно бывает посмотреть на результаты кластеризации. Поэтому воспользуемся методами уменьшения размерности, чтобы визуализировать результаты кластеризации. Для этих целей будем использовать два метода PCA (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html), который основан на получении новых признаков с помощью линейной комбинации старых, а так же tSNE (https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html), который преобразует пространство более сложным образом, пытаясь оставить рядом объекты, которые были близки в исходном пространстве. Эти методы имеют схожий интерфейс, поэтому в обоих случаях последовательность действий будет заключаться в следующем: создать соответствующий объект, указав количество компонент *n_components* равным 2, вызвать метод *fit_transform*, передав переменную *X* и сохранив результат в переменные *X_pca* и *X_tsne* соответственно.
8. Теперь выполните 2 кластеризации, но в качестве матрицы "объекты-признаки" передавайте полученные ранее *X_pca* и *X_tsne*. Посчитайте качество каждой кластеризации с помощью указанных ранее метрик качества.

9. Сильно ли отличается качество новых кластеризаций от исходной для каждого способа сжатия размерности? Нашелся ли способ уменьшения размерности, который позволяет достичь наилучшего качества по обеим метрикам?
10. С помощью функции *plot_embedding* ниже можно посмотреть на визуализацию кластеризации. На вход ей передается двумерная матрица "обекты-признаки", истинные метки и предсказанные метки. Она строит на плоскости объекты, при этом числом обозначена истинная метка объекта, а одинаковым цветом — объекты одного кластера.

```
def plot_embedding(X, y, predicted_labels, title=None):
    x_min, x_max = np.min(X, 0), np.max(X, 0)
    X = (X - x_min) / (x_max - x_min)

    plt.figure(figsize=(10, 10))
    ax = plt.subplot(111)
    for i in range(X.shape[0]):
        plt.text(X[i, 0], X[i, 1], str(y[i]),
                 color=plt.cm.Set1(predicted_labels[i] / 10.),
                 fontdict={'weight': 'bold', 'size': 9})

    plt.xticks([]), plt.yticks([])
    if title is not None:
        plt.title(title)
    plt.show()
```

11. Для каждой из матриц *X_pca* и *X_tsne* и результатов соответствующих кластеризаций вызовите функцию *plot_embedding*.
12. Проанализируйте полученные результаты: какой из трех методов кластеризации: на исходных признаках, на признаках, полученных с помощью PCA или tSNE дал наилучший результат по обеим метрикам? Какой из методов сжатия размерности дает наилучшую визуализацию на плоскости? Можно ли сказать, что результаты визуализации согласуются с метриками: если кластеры кажутся отделимыми визуально, то и значения метрик выше? Как вы думаете: если увеличить число кластеров (то есть взять не 10, а 12, 15 и пр.) может ли кластеризация быть более качественной? Почему?