

Практическая работа № 3

Классификация ирисов Фишера с помощью метода kNN. Использование ансамблевых моделей на основе деревьев решений

1. Используя метод k-NN, решить задачу классификации ирисов Фишера.

Имеются данные измерений для 150 экземпляров ирисов, в равных частях (по 50 штук) принадлежащих к трем видам.

1. Iris Setosa



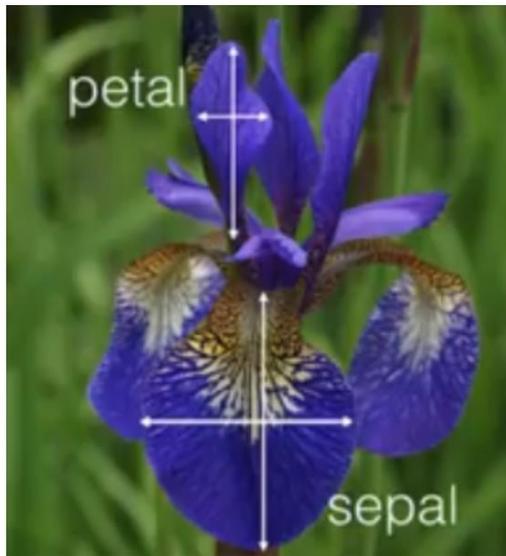
2. Iris Versicolor



3. Iris Virginica



Для каждого экземпляра ириса известны 4 величины: длина чашелистика (Sepal Length), ширина чашелистика (Sepal Width), длина лепестка (Petal Length), ширина лепестка (Petal Width).



```
5.1,3.8,1.9,0.4,Iris-setosa
4.8,3.0,1.4,0.3,Iris-setosa
5.1,3.8,1.6,0.2,Iris-setosa
4.6,3.2,1.4,0.2,Iris-setosa
5.3,3.7,1.5,0.2,Iris-setosa
5.0,3.3,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.5,2.8,4.6,1.5,Iris-versicolor
5.7,2.8,4.5,1.3,Iris-versicolor
```

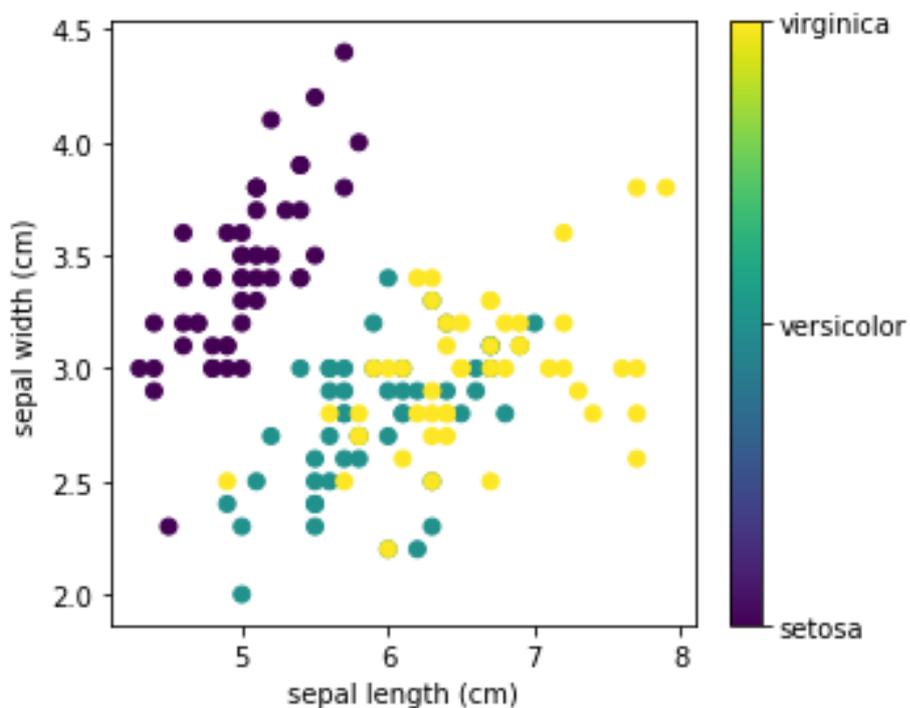
Iris flower: sepal length, sepal width, petal length and width

Исходные данные - уже есть в пакете sklearn.

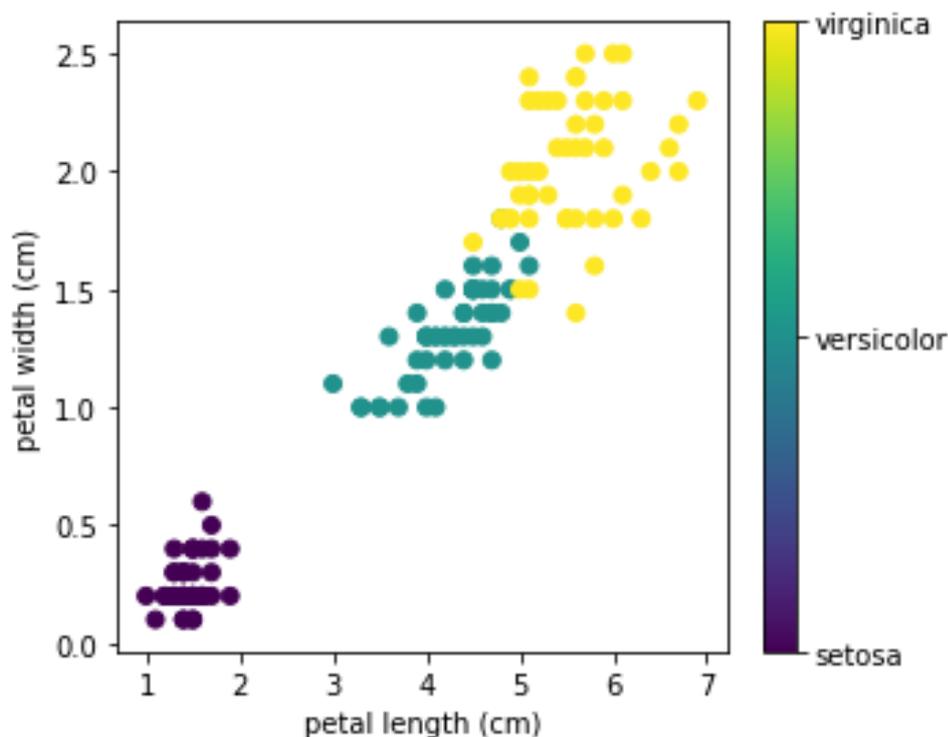
```
from sklearn.datasets import load_iris
iris = load_iris()
```

Загрузите и изучите данные.

Попробуем их визуализировать. Поскольку признаков 4, возьмем из них, например, длину и ширину чашелистика:



Теперь посмотрим на распределение классов в зависимости от длины и ширины лепестка.



Один из классов (*Iris setosa*) линейно-разделим от двух остальных. Два других класса являются уже трудно разделимыми, что и делает эту задачу интересной. Задача классификации ирисов Фишера является классической задачей машинного обучения, как и задача MNIST например, и на ней тестируются многие алгоритмы классификации и кластеризации.

Перейдем к ее решению.

Создайте массивы X и y входных данных и меток классов, соответственно.

Для решения данной задачи методом ближайших соседей, нам потребуется несколько вспомогательных функций.

Для проверки написания этих функций давайте создадим небольшой тестовый набор данных из 10 экземпляров "цветов" разных классов.

```
dataset = X[:150:15]  
output = y[:150:15]
```

Реализуйте функцию, вычисляющую евклидово расстояние между двумя векторами euclidean_distance(row1, row2).

Проверьте ее работу на тестовом наборе данных, вычислив расстояние между всеми векторами тестового набора и dataset[5]. У Вас должны получиться следующие значения:

```
3.59722114972099  
3.4899856733230297  
3.539774004085572
```

```
3.66742416417845
2.128379665379276
0.0
1.1874342087037915
2.5159491250818244
1.6217274740226855
2.2158519806160335
```

Далее нам потребуется функция **get_neighbors**(train_set, labels, test_row, num_neighbors), которая находит в train_set выборке k = num_neighbors ближайших соседей (в смысле близости евклидова расстояния) к данному (test_row).

Такая функция, возможно, сложна для написания, поэтому Вы можете воспользоваться уже готовой реализацией. На нашем тестовом наборе

```
neighbors = get_neighbors(dataset, output, dataset[5], 3)
for neighbor in neighbors:
    print(neighbor)
```

она дает результат (3 ближайших соседа)

```
(array([6.6, 3. , 4.4, 1.4]), 0.0, 1)
(array([5.5, 2.6, 4.4, 1.2]), 1.1874342087037915, 1)
(array([6.9, 3.2, 5.7, 2.3]), 1.6217274740226855, 2)
```

Функция возвращает список из k элементов, каждый из которых в свою очередь – кортеж из 3 элементов (со значениями координат вектора, расстояния до test_row и метки класса).

Переходим к следующей функции **predict_classification**(train_set, labels, test_row, num_neighbors), возвращающей метку класса, которому вероятно принадлежит объект test_row.

predict_classification будет вызывать предыдущую функцию **get_neighbors**, определять, какое из значений меток классов (0, 1, 2) встречается среди ближайших соседей наибольшее число раз и возвращать это значение.

Реализуйте функцию predict_classification(train_set, labels, test_row, num_neighbors) и проверьте ее работу на тестовом наборе данных.

Вызываем функцию:

```
prediction = predict_classification(dataset, output, dataset[5], 3)
print('Expected %d, Got %d.' % (output[5], prediction))
```

У Вас должно получиться (ожидаемый ответ, полученный ответ):

```
Expected 1, Got 1.
```

Наконец, реализуем функцию **k_nearest_neighbors**(train_set, labels, test, num_neighbors), которая получает на вход уже не один тестовый элемент, как предыдущие функции, а

набор таких элементов (test), для каждого из них определяет класс и возвращает список предсказанных ответов.

Теперь мы готовы вернуться к нашей задаче.

Разобьем выборку на обучающую и тестовую.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Для этого мы используем метод **train_test_split** из sklearn, который случайным образом разбивает выборку в соотношении 80%-20% (test_size=0.2) для обучения и тестирования соответственно.

Зададим теперь произвольно num_neighbors и найдем предсказания predictions меток классов для тестовой выборки.

```
predictions = k_nearest_neighbors(X_train, y_train, X_test, num_neighbors)
```

Правильные ответы хранятся в переменной y_test.

Найдите долю правильных ответов.

Снова произведите разбиение **train_test_split** выборки на тестовую и обучающую. Элементы выборок будут уже другие. Оцените точность классификатора.

Для выбора оптимального значения k постройте график зависимости точности классификатора от числа k=1,..., 60. Посмотрите на этот график при различных разбиениях на тестовую и обучающую выборку и сделайте вывод о наилучших значениях k в Вашей задаче.

В заключении работы посмотрим на встроенный в sklearn алгоритм kNN.

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

(Вы можете задавать число соседей, используемую метрику (расстояние не обязательно должно быть евклидовым, можно написать даже свою функцию "расстояния"). Также можно каждому соседу дать вес в соответствии с его расстоянием до нашего объекта.)

В приложенном файле приведена также реализация алгоритма логистической регрессии из sklearn для нашей задачи.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Сравните точность работы этих двух классификаторов.

2. Использование деревьев решений и их ансамблей для задачи классификации ирисов Фишера

Мы по-прежнему рассматриваем тот же набор данных (вам нужна тестовая и обучающая выборка). В этой части работы Вы используете методы, реализованные в `sklearn`.

Обучите следующие классификаторы на основе деревьев решений:

1) одно дерево,

Покажите пример переобученного дерева и опишите, как Вы боретесь с этим явлением.

2) случайный лес,

Какие параметры есть у этого метода? Продемонстрируйте их влияние на эффективность модели.

3) градиентный бустинг.

Какие параметры есть у этого метода? Продемонстрируйте их влияние на эффективность модели.