Конспект по теме "Расстояние между векторами"

Скалярное произведение

Если умножить все компоненты, а затем сложить полученные результаты, то получится **скалярное произведение**. В результате этой операции над двумя векторами равного размера получается новое число, или **скаляр**.

У скалярного произведения векторов $a=[x1, x2,..., xn]$ и $b=[y1, y2,..., yn]$ такая формула:

$$(a, b) = x_1 \times y_1 + x_2 \times y_2 + ... + x_n \times y_n$$

Скалярное произведение векторов *a* и *b* обычно обозначается круглыми скобками (a, b) или точкой $a \cdot b$.

В библиотеке *NumPy* скалярное произведение можно вычислить с помощью функции **numpy.dot()**:

```
import numpy as np

dot_value = np.dot(vector1, vector2)
```

Ещё проще скалярное произведение вычисляется **оператором матричного умножения**. Он обозначается символом коммерческое эт (англ. *commercial at*), или «собака» (@):

```
import numpy as np

volume = np.array([0.1, 0.3, 0.1])
content = np.array([0.4, 0.0, 0.1])

dot_value = vector1 @ vector2
```

Другая операция — поэлементное умножение оператором *. Его результатом, в отличие от скалярного произведения, будет вектор:

```
import numpy as np

vector3 = vector1 * vector2
```

Расстояние на плоскости

Длина вектора, или его модуль, равна квадратному корню из скалярного произведения вектора на себя. Например, для вектора $a=(x, y)$ она вычисляется так:

$$|a| = \sqrt{(a, a)} = \sqrt{x^2 + y^2}$$

Чтобы измерить расстояние между двумя точками, т. е. получить квадратный корень из скалярного произведения разностей векторов, рассчитаем **евклидово расстояние**. Оно вычисляет наименьшее расстояние по теореме Пифагора: в прямоугольном треугольнике квадрат гипотенузы равен сумме квадратов катетов.

Евклидово расстояние записывается так: $d_2(a, b)$. Нижний индекс 2 означает, что координаты вектора возводятся во вторую степень.

Расстояние между точками $a(x1, y1)$ и $b(x2, y2)$ вычисляется по формуле:

$$d_2(a, b) = \sqrt{(a - b, a - b)} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Найдём евклидово расстояние между точками $a=(5, 6)$ и $b=(1, 3)$:

```
import numpy as np

a = np.array([5, 6])
b = np.array([1, 3])
d = np.dot(a-b, a-b)**0.5
print('Расстояние между точками a и b равно', d)
```

Для расчёта расстояний в *SciPy* есть библиотека **distance**. Вызовем функцию вычисления евклидова расстояния *distance.euclidean()*:

```
import numpy as np
from scipy.spatial import distance

a = np.array([5, 6])
b = np.array([1, 3])
d = distance.euclidean(a, b)
print('Расстояние между точками a и b равно', d)
```

Результаты расчётов одинаковые.

Манхэттенское расстояние

Манхэттенское расстояние, или **расстояние городских кварталов**, — сумма модулей разностей координат. Название пошло от уличной планировки Манхэттена, где не применимо евклидово расстояние (его считают по прямой).

Вычислим манхэттенское расстояние между точками $a=(x1, y1)$ и $b=(x2, y2)$ по формуле:

$$d_1(a, b) = |x_1 - x_2| + |y_1 - y_2|$$

Манхэттенское расстояние записывается так: $d_1(a, b)$. Нижний индекс 1 означает, что координаты вектора возводятся в первую степень (число не меняется).

Для расчёта манхэттенского расстояни в *SciPy* есть функция *distance.cityblock():*

```
import numpy as np
from scipy.spatial import distance

a = np.array([5, 6])
b = np.array([1, 3])
d = distance.cityblock(a, b)
print('Расстояние между точками a и b равно', d)
```

Чтобы найти минимальный индекс в массиве *NumPy*, вызовите функцию **argmin():**

```
index = np.array(distances).argmin() # индекс минимального элемента
```

Расстояния в многомерном пространстве

В машинном обучении вектор — это признаки объектов, обычно многомерные с размерностью больше двух.

Евклидово расстояние между векторами $a=(x1, x2, ..., xn)$ и $b=(y1, y2,..., yn)$ — это сумма квадратов разностей координат:

$$d_2(a, b) = \sqrt{(y_1 - x_1)^2 + ... + (y_n - x_n)^2} = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

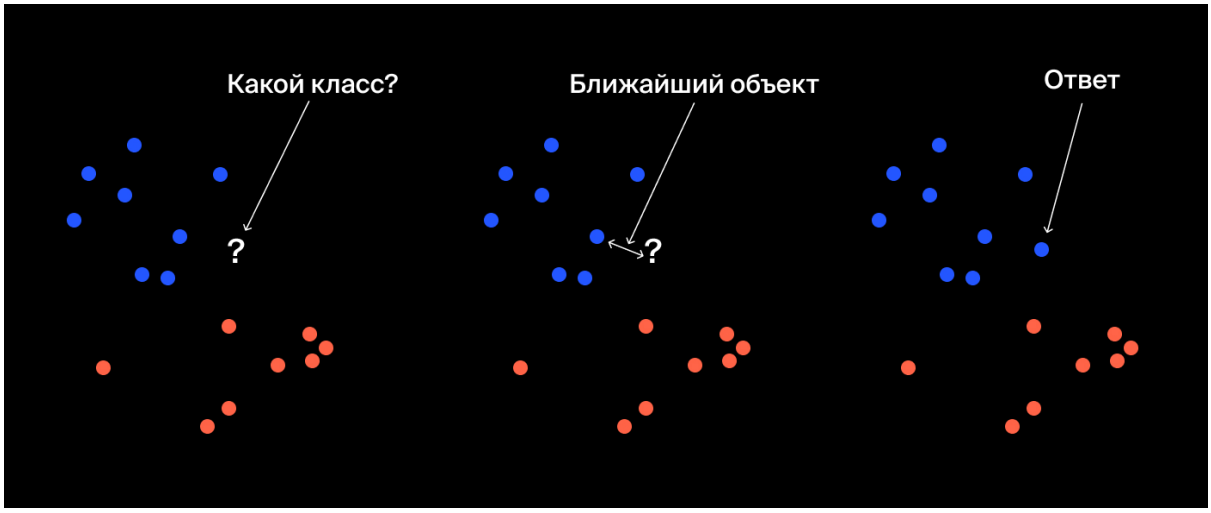
А манхэттенское расстояние — сумма модулей разностей координат:

$$d_1(a, b) = |x_1 - y_1| + |x_2 - y_2| + ... + |x_n - y_n| = \sum_{i=1}^{n} |x_i - y_i|$$

Когда координат больше двух, для вычисления расстояний в многомерном пространстве вызывают знакомые вам функции: *distance.euclidean()* и *distance.cityblock()*.

Метод ближайших соседей

Рассмотрим картинку. Как предсказать класс объекта? Найдём самый близкий объект в выборке и получим от него ответ. Именно так работает **метод ближайших соседей** (*nearest neighbors algorithm*). Ближайший объект обычно ищется в обучающей выборке.



Алгоритм работает и на плоскости, и в многомерном пространстве — только расстояния вычисляются по многомерным формулам.

Создание класса модели

Класс (англ. *class*) — это новый тип данных с собственными методами и атрибутами. Разберём константную модель для задачи регрессии. Ответы она предсказывает средним значением целевого признака по обучающей выборке.

Чтобы создать новый класс, укажите ключевое слово `class`, после которого нужно написать название класса:

```
# от англ. константная модель для задачи регрессии
class ConstantRegression:
```

```
    # содержимое класса с отступом в 4 пробела
    # ...
```

Для обучения модели создадим метод *fit()*. Это функция внутри класса, у которой первый параметр всегда **self**. *self* — это переменная, в которой хранится наша модель. Она нужна для работы с атрибутами. Два других параметра — это признаки и целевой признак обучающей выборки, как в *sklearn*.

```
class ConstantRegression:
    def fit(self, features_train, target_train):
        # содержимое функции с отступом в 4 + 4 пробела
        # ...
```

При обучении нужно сохранить среднее значение целевого признака. Чтобы создать новый атрибут `value`, добавьте `self` с точкой в начало названия переменной. Так вы покажете, что переменная находится внутри класса:

```
class ConstantRegression:
    def fit(self, features_train, target_train):
        self.value = target_train.mean()
```

Построим метод *predict()*, который предскажет ответ — сохранённое среднее:

```
class ConstantRegression:
    def fit(self, features_train, target_train):
        self.value = target_train.mean()

    def predict(self, new_features):
        answer = pd.Series(self.value, index=new_features.index)
        return answer
```