KARPOV.COURSES >>>
КОНСПЕКТ

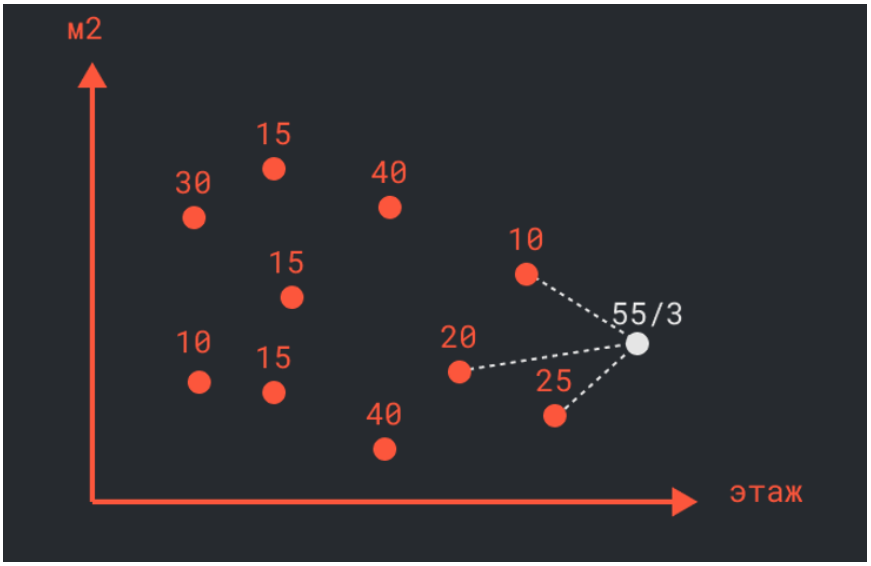
Конспект > 16 урок > Метод K ближайших соседей: обоснование нелинейности, гиперпараметры и подбор метрики близости объектов

> Алгоритм k-Nearest Neigbor

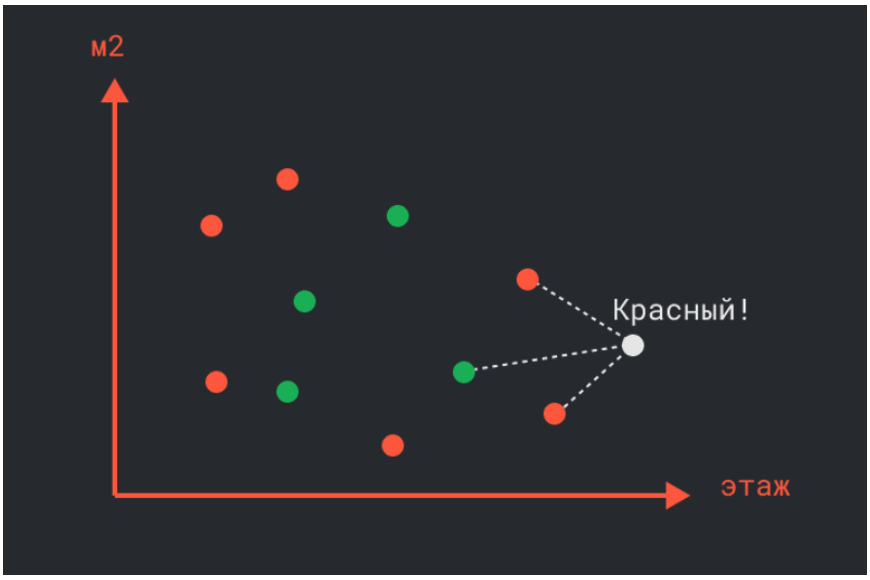
K-Nearest Neigbor или метод ближайшего соседа – метрический алгоритм, используемый при обучении с учителем. Его суть заключается в том, что таргеты похожих объектов тоже будут похожи, поэтому для предсказания нового таргета можно просто посмотреть на его соседей.

Тогда для каждого нового вещественного предсказания необходимо будет посмотреть на k его соседей и усредниться по их таргету.

Допустим, есть датафрейм с двумя признаками ($м^2$ и этаж) и вещественным таргетом (например, ценой), а также один объект с неизвестным таргетом. Усреднившись по трём ближайшим соседям ($k = 3$), получим значение предсказываемого таргета равное $\frac{55}{3}$.



Для решения задачи классификации можно посмотреть на самый популярный класс среди ближайших соседей, а если количество объектов в обоих классах одинаково, то взять самый ближайший.



Общий алгоритм kNN

1. Поступление алгоритму kNN нового объекта x_i;

2. Измерение попарных расстояний между x_i и всеми остальными объектами из тренировочной выборки;

3. Выбор k ближайших соседей;

4. Формирование таргетной переменной через усреднение или голосование;

Сравнение алгоритма kNN с линейными моделями

Линейные модели являются параметрическими, то есть мы можем сначала вычислить коэффициенты β, позволяющие построить модель, а затем использовать их для всех остальных предсказаний. Более того, можно интерпретировать коэффициенты β, например, оценив важность влияния их на таргет или принцип их влияния (положительно или отрицательно влияют). Немаловажным является тот факт, что модели умеют экстраполировать зависимости и улавливать общий тренд, поэтому они могут предсказывать таргеты объектов, непохожих на предыдущие.

Метод kNN в свою очередь является неинтерпретируемым, так как он смотрит на ближайшие объекты, не строя общую зависимость. С другой стороны, если зависимость переменных и таргета нелинейна, более предпочтительно применение kNN (но не всегда). Наконец из-за того, что для каждого нового объекта метод kNN считает попарные расстояния со всеми имеющимися объектами, с большими данными этот метод работает довольно медленно.

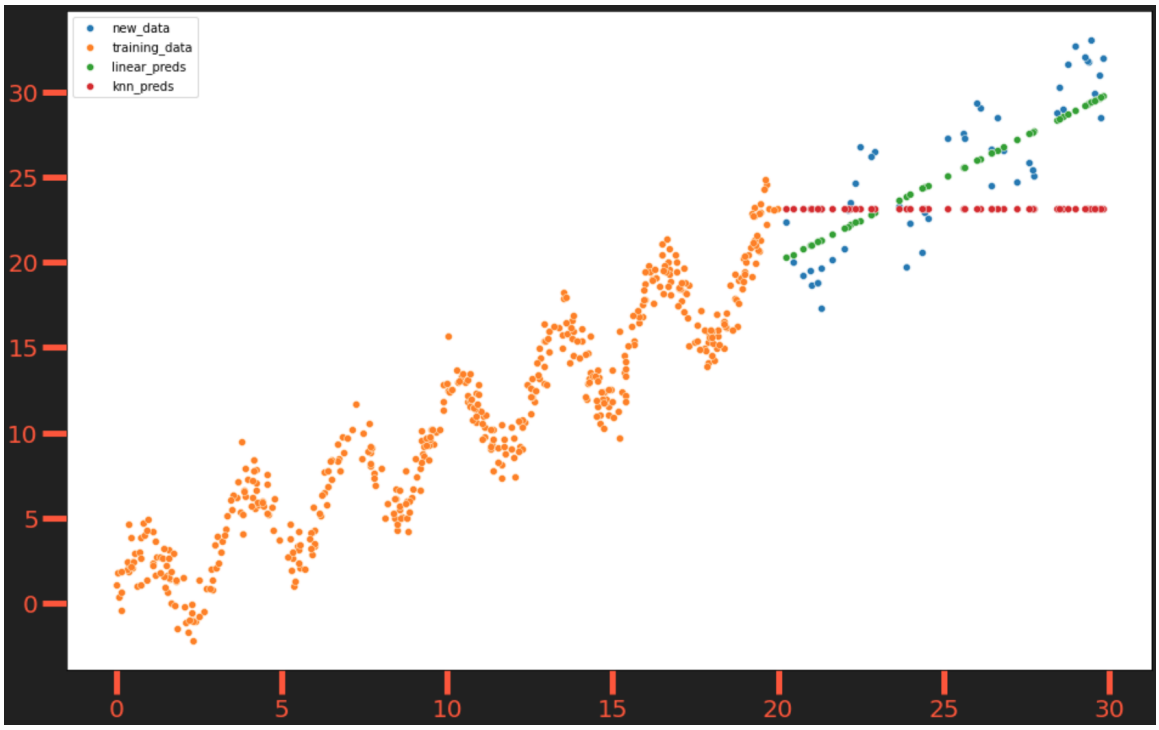
> Практика. Класс KNeighborsRegressor

Метод kNN реализован в классе `KNeighborsRegressor` библиотеки `sklearn`.

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_validate


splitter = KFold(n_splits=5, shuffle=True, random_state=33)
knn = KNeighborsRegressor(n_neighbors=3)
knn_cv = cross_validate(knn, X, Y,
                        cv=splitter, scoring='neg_mean_squared_error',
                        return_train_score=True)
```

Если предсказываемые объекты похожи на объекты тренировочной выборки, на нелинейных данных метод kNN справляется гораздо лучше линейной регрессии. Однако если предсказываемые объекты не похожи на объекты тренировочной выборки, качество модели kNN сильно падает.



> Способы вычисления расстояний между объектами

Каким способом можно улучшить метод kNN?

Во-первых, можно использовать разные методы вычисления расстояния между объектами. Общая формула для вычисления расстояния между объектами — это формула расстояния Минковского:

$$\rho(x, z) = (\sum_j^M (|x_j - z_j|^p))^{\frac{1}{p}}$$

Из формулы видно, что Евклидово расстояние является частным случаем расстояния Минковского при $p = 2$:

$$\rho(x, z) = \sqrt{\sum_{j=1}^M (|x_j - z_j|^2)}$$

В свою очередь, расстояние Манхэттена также является частным случаем расстояния Минковского при $p = 1$.

$$\rho(x, z) = \sum_{j=1}^M |x_j - z_j|$$

Как параметр p из формулы расстояния влияет на алгоритм kNN?

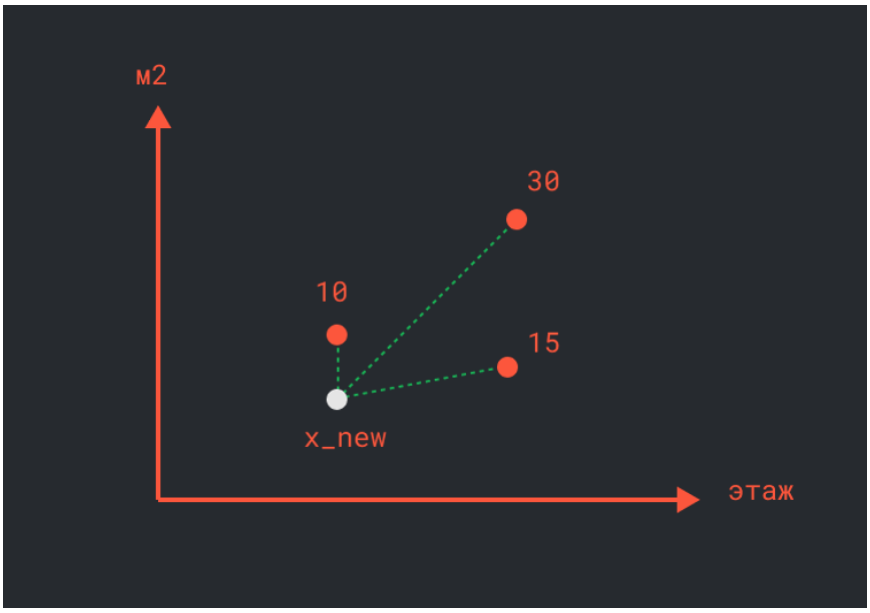
Использование параметра p позволяет указать модели, какие объекты называть схожими и близкими друг другу, а какие нет. Другими словами, на сколько важно расстояние (разница) между объектами для классификации объекта как схожего.

$p < 1 \rightarrow$ с ростом разницы между объектами по конкретному признаку вклад этой разницы уменьшается;

$p > 1 \rightarrow$ с ростом разницы между объектами по конкретному признаку вклад разницы увеличивается.

> Перевзвешивание соседей

До этого момента при расчете таргета мы с одинаковой важностью оценивали каждого из соседей, не обращая внимания на их близость к предсказываемому объекту. Однако кажется логичным всё же принимать во внимание тот факт, что чем дальше сосед, тем меньше его вклад.



Как можно перевзвесить таргеты соседей таким образом, чтобы ближайшие давали больший вклад, чем самые далекие?

Зная веса каждого объекта, можно вычислять таргет следующим образом:

$a(x) = \frac{\sum_j^K w_i y_i}{\sum_j^K w_i}$, где w — вес каждого объекта.

Как же находить эти веса w? Рассмотрим некоторые подходы.

1. Базовая концепция подбора весов — возведение произвольного параметра `q` в степень, равную номеру объекта: $w_i = q^i$

2. Другая базовая концепция — $w_i = \frac{k+1-i}{k}$, где k — произвольный параметр.

3. Гауссовское ядро, учитывающее расстояния между объектами — $w_i = \frac{1}{\sqrt{2 \cdot \pi}} \cdot e^{-\frac{1}{2} \cdot z^2}$, где

z соответствует расстоянию между двумя точками, разделенному на гиперпараметр h: $z = \frac{\rho(x, x_j)}{h}$.

> Практика. Гауссовское ядро

Для начала реализуем Гауссовское ядро по формуле $w_i = \frac{1}{\sqrt{2 \cdot \pi}} \cdot e^{(-\frac{1}{2} \cdot \frac{\rho^2(x, x_i)}{h^2})}$

```
def kernel(distances, h):
    const = 1 / (np.sqrt(2 * np.pi))
    power = (-1/2) * ((distances)**2) / h**2
    return const * np.exp(power)
```

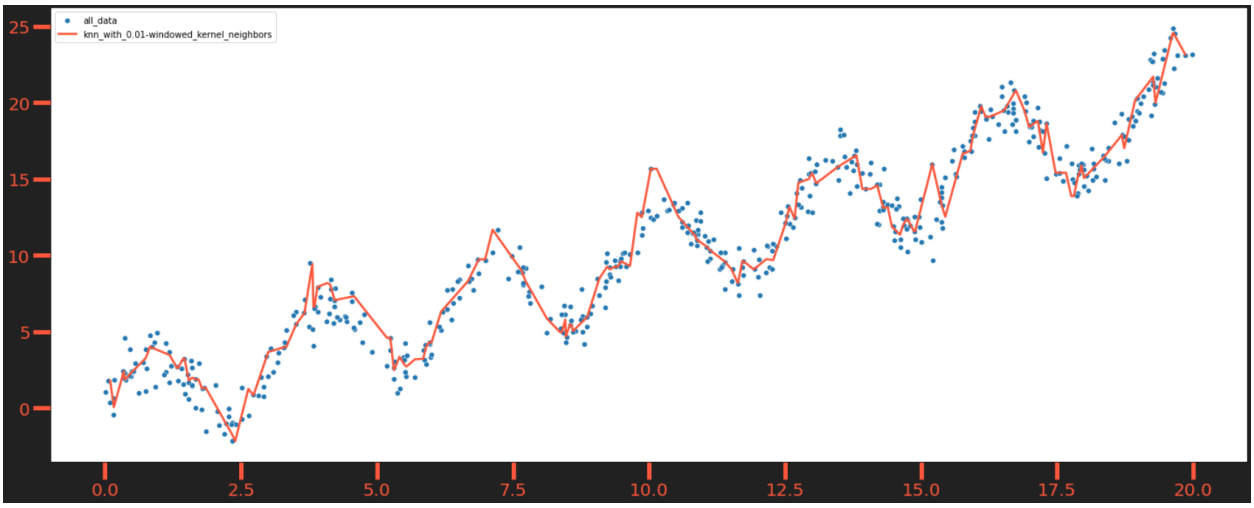
А затем запустим алгоритм на нескольких h:

```
for h in [0.01, 0.01, 10, 100, 500]:
    k = kernel(distances, h=h):
    knn = KNeighborsRegressor(n_neighbors=8, weights=k)
    knn.fit(X_train, y_train)
    preds_test = knn.predict(X_test)
```

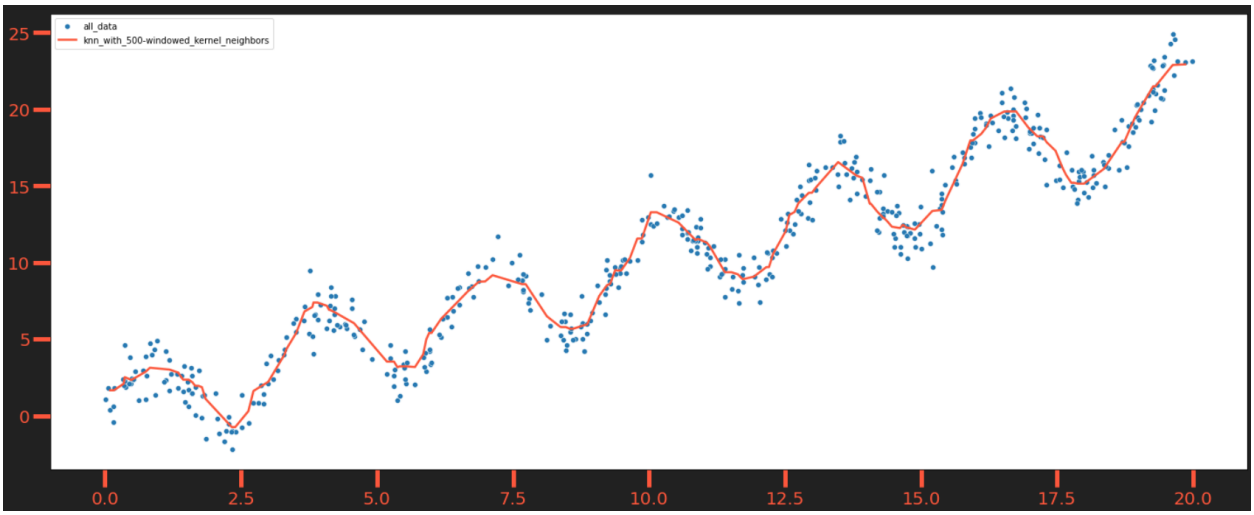
Если сравнить результаты при $h = 0.01$ и $h = 500$, то можно увидеть, что при малых значениях h модель переобучена, однако с ростом значения h график

кривой сглаживается.

$h = 0.01$



$h = 500$



> Масштабирование данных

Если признаки обладают разными масштабами, то именно масштаб, а не само расстояние между объектами, будет вносить вклад в разницу между объектами. Поэтому важно масштабировать данные перед запуском алгоритма kNN.

Для масштабирования данных можно использовать класс `StandardScaler` из библиотеки `sklearn`.

```
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split


X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                    random_state=0,
                                                    test_size=0.2)

pipe = Pipeline([('scaler', StandardScaler()),
                 ('KNN', KNeighborsClassifier(weights=kernel))])

pipe.fit(X_train, Y_train)
```