**Ближайшие соседи**

[**sklearn.neighbors**](http://scikit-learn.org/stable/modules/classes.html#module-sklearn.neighbors) обеспечивает функциональность для неконтролируемых и контролируемых методов обучения на основе соседей. Неконтролируемые ближайшие соседи являются основой многих других методов обучения, в частности многообразного обучения и спектральной кластеризации. Контролируемое обучение соседей происходит в двух вариантах: [классификация](http://scikit-learn.org/stable/modules/neighbors.html#classification) данных с дискретными метками и [регрессия](http://scikit-learn.org/stable/modules/neighbors.html#regression) для данных с непрерывными метками.

Принцип, лежащий в основе методов ближайших соседей, заключается в том, чтобы найти предопределенное количество учебных образцов, наиболее близких к расстоянию до новой точки, и предсказать метку из них. Количество выборок может быть определяемой пользователем константой (k-ближайшая соседка) или варьироваться в зависимости от локальной плотности точек (радиус-соседнее обучение). Расстояние, вообще говоря, может быть любой метрической мерой: стандартное евклидово расстояние является наиболее распространенным выбором. Методы, основанные на соседях, известны как не обобщающие методы машинного обучения, так как они просто «запоминают» все свои учебные данные.

Несмотря на свою простоту, ближайшие соседи успешно справились с большим количеством проблем классификации и регрессии, включая рукописные цифры или сцены спутникового изображения. Будучи непараметрическим методом, он часто бывает успешным в ситуациях классификации, где граница решения очень нерегулярна.

Классы [**sklearn.neighbors**](http://scikit-learn.org/stable/modules/classes.html#module-sklearn.neighbors) могут обрабатывать либо массивы Numpy, либо scipy.sparse в качестве входных данных. Для плотных матриц поддерживается большое количество возможных метрик расстояния. Для разреженных матриц для поиска поддерживаются произвольные метрики Минковского.

Существует множество учебных процедур, которые полагаются на ближайших соседей по своей сути. Одним из примеров является [оценка плотности ядра](http://scikit-learn.org/stable/modules/density.html#kernel-density) , обсуждаемая в разделе [оценки плотности](http://scikit-learn.org/stable/modules/density.html#density-estimation) .

**Алгоритм ближайших соседи без учителя**

[**NearestNeighbors**](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors) реализует обучение без учителя ближайших соседей. Он действует как единый интерфейс для трех различных ближайших соседей алгоритмов: [**BallTree**](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html#sklearn.neighbors.BallTree), [**KDTree**](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html#sklearn.neighbors.KDTree) и brute-force алгоритм , основанной на рутинах в **[sklearn.metrics.pairwise](http://scikit-learn.org/stable/modules/classes.html" \l "module-sklearn.metrics.pairwise" \o "sklearn.metrics.pairwise)**. Выбор алгоритма поиска соседей контролируется через ключевое слово 'algorithm', которое должно быть одним из . Когда значение по умолчанию передается, алгоритм пытается определить наилучший подход из данных обучения. Для обсуждения сильных и слабых сторон каждого варианта ['auto', 'ball\_tree', 'kd\_tree', 'brute']'auto'

Поиск ближайших соседей

Для простой задачи поиска ближайших соседей между двумя наборами данных **[sklearn.neighbors](http://scikit-learn.org/stable/modules/classes.html" \l "module-sklearn.neighbors" \o "sklearn.neighbors)**можно использовать неконтролируемые алгоритмы :

**>>> from**  **sklearn.neighbors**  **import**  NearestNeighbors

**>>> import**  **numpy**  **как**  **np**

**>>>** X = np . массив [[ - 1 , - 1 ], [ - 2 , - 1 ], [ - 3 , - 2 ], [ 1 , 1 ], [ 2 , 1 ], [ 3 , 2 ]])

**>>>** nbrs знак равно NearestNeighbors ( n\_neighbors = 2 , algorithm = 'ball\_tree' ) . fit ( X )

**>>>** расстояния , индексы = nbrs . kneighbors ( X )

**>>>** индексный

массив ([[0, 1],

[1, 0],

[2, 1],

[3, 4],

[4, 3],

[5, 4]] ... )

**>>>** distances

array ([[0., 1.],

[0., 1.],

[0., 1.41421356],

[0., 1.],

[0., 1.],

[0., 1.41421356]])

Поскольку набор запросов соответствует набору тренировок, ближайшим соседом каждой точки является сама точка, на расстоянии от нуля.

Кроме того, можно эффективно создавать разреженный график, показывающий соединения между соседними точками:

**>>>** nbrs . kneighbors\_graph ( X ) . toarray ()

([[1., 1., 0., 0., 0., 0.],

[1., 1., 0., 0., 0., 0.],

[0., 1., 1., 0., 0., 0.],

[0., 0., 0., 1., 1., 0.],

[0., 0., 0., 1., 1 ., 0.],

[0., 0., 0., 0., 1., 1.]])

Наш набор данных структурирован таким образом, что точки, расположенные рядом в порядке индекса, находятся рядом в пространстве параметров, что приводит к примерно блочно-диагональной матрице K-ближайших соседей. Такой разреженный граф является полезным в различных обстоятельствах , которые делают использование пространственных отношений между точками для неконтролируемого обучения

**Классы KDTree и BallTree**

**>>> from**  **sklearn.neighbors**  **import**  KDTree

**>>> import**  **numpy**  **как**  **np**

**>>>** X = np . массив [[ - 1 , - 1 ], [ - 2 , - 1 ], [ - 3 , - 2 ], [ 1 , 1 ], [ 2 , 1 ], [ 3 , 2 ]])

**>>>** kdt = KDTree( X , leaf\_size = 30 , metric = 'euclidean' )

**>>>** kdt . query ( X , k = 2 , return\_distance = **False** )

([[0, 1],

[1, 0],

[2, 1],

[3, 4],

[4, 3],

[5, 4] ] ...)

Для получения дополнительной информации о параметрах, доступных для поиска соседей, включая спецификацию стратегий запросов, различных показателей расстояния и т. Д., Обратитесь к документации по классам **[KDTree](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html" \l "sklearn.neighbors.KDTree" \o "sklearn.neighbors.KDTree)**и **[BallTree](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html" \l "sklearn.neighbors.BallTree" \o "sklearn.neighbors.BallTree)**классу. Список доступных показателей см. В документации к **[DistanceMetric](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html" \l "sklearn.neighbors.DistanceMetric" \o "sklearn.neighbors.DistanceMetric)**классу.

**Классификация**

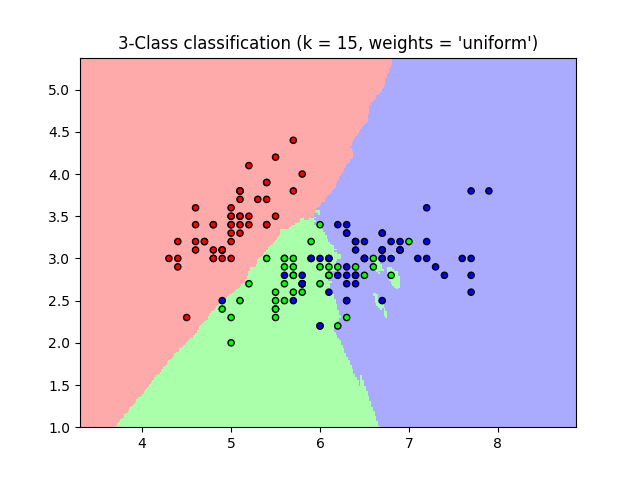
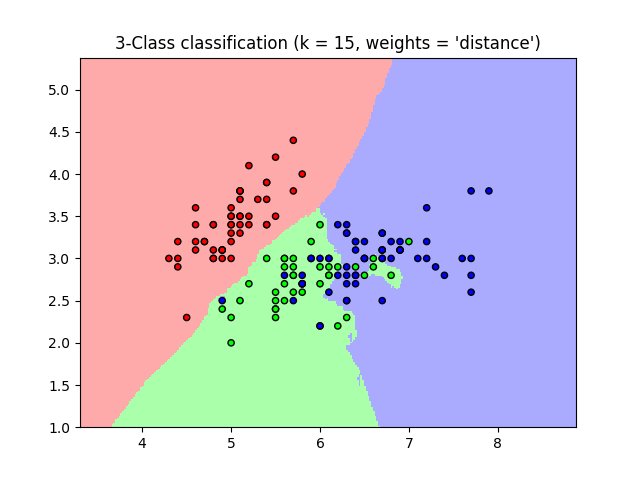
Классификация по соседству - это тип обучения на основе экземпляров или не обобщающего обучения : он не пытается построить общую внутреннюю модель, а просто хранит экземпляры данных обучения. Классификация вычисляется с простого большинства голосов ближайших соседей каждой точки: точке запроса присваивается класс данных, который имеет наибольшее количество представителей в ближайших соседях точки.

scikit-learn реализует два разных классификатора ближайших соседей: **[KNeighborsClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html" \l "sklearn.neighbors.KNeighborsClassifier" \o "sklearn.neighbors.KNeighborsClassifier)**реализует обучение на основе К ближайших соседей каждой точки запроса, где К- целочисленное значение, заданное пользователем.[**RadiusNeighborsClassifier**](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.RadiusNeighborsClassifier.html#sklearn.neighbors.RadiusNeighborsClassifier)реализует обучение, основанное на количестве соседей в пределах фиксированного радиуса ркаждой точки тренировки, где р- значение с плавающей запятой, указанное пользователем.

ККлассификация « соседей» в России **[KNeighborsClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html" \l "sklearn.neighbors.KNeighborsClassifier" \o "sklearn.neighbors.KNeighborsClassifier)** чаще используется в двух методах. Оптимальный выбор значения Ксильно зависит от данных: в целом, более сильный Кподавляет эффекты шума, но делает границы классификации менее отчетливыми.

В тех случаях, когда данные не подвергаются равномерной выборке, классификация соседей по радиусу **[RadiusNeighborsClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.RadiusNeighborsClassifier.html" \l "sklearn.neighbors.RadiusNeighborsClassifier" \o "sklearn.neighbors.RadiusNeighborsClassifier)**может быть лучшим выбором. Пользователь задает фиксированный радиус р, так что точки в более редких окрестностях используют меньшее количество ближайших соседей для классификации. Для пространств с большими размерами этот метод становится менее эффективным из-за так называемого «проклятия размерности».

Основная классификация ближайших соседей использует однородные веса: то есть значение, присвоенное точке запроса, вычисляется с простого большинства голосов ближайших соседей. В некоторых случаях лучше убирать соседей таким образом, чтобы приближенные соседи вносили больший вклад в подгонку. Это можно выполнить с помощью weightsключевого слова. Значение по умолчанию присваивает однородным весам каждому соседу.присваивает весам, пропорциональным обратному расстоянию от точки запроса. В качестве альтернативы может быть предоставлена ​​пользовательская функция расстояния, которая используется для вычисления весов.weights = 'uniform'weights = 'distance'

** **

*class* sklearn.neighbors.**KNeighborsClassifier**(*n\_neighbors=5*, *weights=’uniform’*, *algorithm=’auto’*, *leaf\_size=30*, *p=2*, *metric=’minkowski’*, *metric\_params=None*, *n\_jobs=1*, *\*\*kwargs*)

Классификатор реализующий выбор к-ближайших соседей:

Параметры:

**n\_neighbors**(n-соседей): целое число, опциональный параметр (по умолчанию значение = 5)

Количество соседей используется по умолчанию для к-соседних запросов

**weights**(веса) - вызываемый, опциональный параметр (по умолчанию ‘uniform’ - равнозначный)

Весовая функция используется для прогнозирования. Возможные значения:

uniform - все точки из каждого класса весят одинаково

distance(расстояние) - взвешивает точки путем инвертирования их дистанций. Таким образом ближайший сосед запрашиваемой точки оказывает большее влияние чем сосед который расположен дальше.

callable(вызываемый) - пользовательски определенная функция которая принимает массив расстояний и возвращает массив такого же типа который уже содержит веса.

**algorithm:** {‘auto’, ‘ball\_tree’, ‘kd\_tree’, ‘brute’}, опциональный параметр

“авто” - подбирает наиболее подходящий алгоритм исходя из входящих параметров , метрическое дерево” , “k - мерное дерево” - алгоритм который позволяет разбивать к- мерное метрическое пространство на меньшие части путем его рассечения гиперплоскостями, ‘brute” - брут - форс перебирает все точки метрического пространства.

**leaf\_size(размер листа)**: int, optional (default = 30) (целочисленный, опциональный ( по умолчанию - 30)

Размер листа относится к параметру algorithm , а конкретно к “ метрическому дереву” и “к мерное метрическое дерево” . Данный параметр может ускорить данный алгоритм - структуру данных, все зависит от памяти которая требуется для хранения этой самой структуры данных.Оптимально значение зависит от происхождения задачи.

**p** : integer, optional (default = 2) (целочисленный , опциональный (по умолчанию 2)

Параметр изменение которого приведет к изменению метрики Минковского

l1 - когда p = 1, l1 = норме пространства l1 то есть сумма модуля разности координат 2-х точек.

l2 - когда p = 2, l2 - Эвклидово пространство

и lp когда p > 2

…

**metric**: метрика, строка либо вызываемый параметр, по умолчанию “минковский”

Метрика расстояния используется для дерева.

По умолчанию метрика Минковского при p = 2 - l2.

**n\_jobs** : int, optional (default = 1) (целочисленный , опциональный параметр по умолчанию = 1)

Число параллельных вычислений для поиска соседей. Если данный параметр равен -1 то число вычислений устанавливается по количеству ядер процессора.

**Пример:**

**>>> X = [[0], [1], [2], [3]]  
 >>> y = [0, 0, 1, 1]  
 >>> from sklearn.neighbors import KNeighborsClassifier  
 >>> neigh = KNeighborsClassifier(n\_neighbors=3)  
 >>> neigh.fit(X, y)   
 KNeighborsClassifier(...)  
 >>> print(neigh.predict([[1.1]]))  
 [0]  
 >>> print(neigh.predict\_proba([[0.9]]))  
 [[ 0.66666667 0.33333333]]**

**Методы:**

[**fit**](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier.fit)**(X, y)** - подстраивает модель, использую Х как обучающую выборку , а y как целевые значения.

**get\_params([deep])** - считывает параметры для оценки.

**kneighbors([X, n\_neighbors, return\_distance])** - осущ. поиск к - соседей запрашиваемой точки.

**kneighbors\_graph([X, n\_neighbors, mode])** - просчитывает взвешенный граф для К - соседей точки Х.

**predict(X) -** прогнозирует для входящих данных принадлежность к классам.

[**predict\_proba**](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier.predict_proba)**(X)** - возвращает возможность оценить тестовую выборку Х.

**score(X, y[, sample\_weight])** - возвращает среднюю точность определения меток принадлежности к классам для тестовой выборки.

**set\_params(\*\*params)** - устанавливает параметры для оценки.

**\_\_init\_\_(*n\_neighbors=5*, *weights=’uniform’*, *algorithm=’auto’*, *leaf\_size=30*, *p=2*, *metric=’minkowski’*, *metric\_params=None*, *n\_jobs=1*, *\*\*kwargs*)**

**fit(*X*, *y*)**

Подстраивает модель используя Х как обучающую выборку и у как целевые значения.

Параметры: **X : {array-like, sparse matrix, BallTree, KDTree}**

**Training data. If array or matrix, shape [n\_samples, n\_features], or [n\_samples, n\_samples] if metric=’precomputed’.**

**y : {array-like, sparse matrix}**

**Target values of shape = [n\_samples] or [n\_samples, n\_outputs]**

**get\_params(*deep=True*)**

Параметры: deep: булевый , опциональный. Если истина то вернет параметры для этой оценки и подобъекты…

Тип возвращаемого значеия: params: сопоставление строки - параметры сопоставляются со своими значениями

**kneighbors(*X=None*, *n\_neighbors=None*, *return\_distance=True*)**

Ищет к - соседей точки , Возвращает индексы и расстояния до соседей каждой точки.

**kneighbors\_graph(X=None, n\_neighbors=None, mode=’connectivity’)**

Просчитывает взвешенный граф к-соседей до нужной точки.

Параметры: **X : array-like, shape (n\_query, n\_features), or (n\_query, n\_indexed) if metric == ‘precomputed’**

Запрашиваемая точка или точки. Если отсутствует то находятся соседи для кажлй проиндексированной точки. В этом случае точка запроса не считается своим соседом.

**n\_neighbors : int**