**[sklearn.tree.DecisionTreeClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)**

tree = DecisionTreeClassifier(max\_depth=5, max\_features=10, min\_samples\_leaf=2)

**Настройка параметров**

tree = DecisionTreeClassifier()

tree\_params = {'max\_depth':range(1,20), 'max\_features':range(1,9), ‘min\_samples\_leaf’:range(1,5)}

tree\_grid = GridSearchCV(tree, tree\_params, cv=5, verbose=True)

tree\_grid.best\_params\_, tree\_grid.best\_score\_

**Основные параметры класса :**

* **max\_depth** – максимальная глубина дерева
* **max\_features** — максимальное число признаков
* **min\_samples\_leaf** – минимальное число объектов в листе. У этого параметра есть понятная интерпретация: скажем, если он равен 5, то дерево будет порождать только те классифицирующие правила, которые верны как минимум для 5 объектов

**Плюсы:**

* Порождение четких правил классификации, понятных человеку
* Деревья решений могут легко визуализироваться, то есть может "интерпретироваться"
* Быстрые процессы обучения и прогнозирования;
* Малое число параметров модели;
* Поддержка и числовых, и категориальных признаков.

**Минусы:**

* Очень чувствительны к шумам во входных данных
* Разделяющая граница, построенная деревом решений, имеет свои ограничения (состоит из гиперплоскостей, перпендикулярных какой-то из координатной оси)
* Небольшие изменения в данных могут существенно изменять построенное дерево решений.
* Проблема поиска оптимального дерева решений (минимального по размеру и способного без ошибок классифицировать выборку)
* Сложно поддерживаются пропуски в данных. около 50% кода модели
* Модель умеет только интерполировать, но не экстраполировать

**sklearn.neighbors.KneighborsClassifier**

knn = KNeighborsClassifier(n\_neighbors=10)

Найстройка параметров:

knn\_pipe = Pipeline([('scaler', StandardScaler()), ('knn', KNeighborsClassifier(n\_jobs=-1))])

knn\_params = {'knn\_\_n\_neighbors': range(1, 10)}

knn\_grid = GridSearchCV(knn\_pipe, knn\_params, cv=5, n\_jobs=-1, verbose=True)

tree\_grid.best\_params\_, tree\_grid.best\_score\_

**Основные параметры:**

* **weights:** "uniform" (все веса равны), "distance" (вес обратно пропорционален расстоянию до тестового примера) или другие
* **algorithm** (опционально): "brute", "ball\_tree", "KD\_tree", или "auto". В первом случае ближайшие соседи для каждого тестового примера считаются перебором обучающей выборки. Во втором и третьем — расстояние между примерами хранятся в дереве, что ускоряет нахождение ближайших соседей. В случае указания параметра "auto" подходящий способ нахождения соседей будет выбран автоматически на основе обучающей выборки.
* **leaf\_size** (опционально): порог переключения на полный перебор в случае выбора BallTree или KDTree для нахождения соседей
* **metric:** "minkowski", "manhattan", "euclidean", "chebyshev"

**Плюсы:**

* Простая реализация
* Как правило, метод хорош для первого решения задачи
* Можно адаптировать под нужную задачу выбором метрики или ядра
* Неплохая интерпретация, можно объяснить, почему тестовый пример был классифицирован именно так.

**Минусы:**

* в реальных задачах, как правило, число соседей, используемых для классификации, будет большим (100-150), и в таком случае алгоритм будет работать не так быстро, как дерево решений;
* Если в наборе данных много признаков, то трудно подобрать подходящие веса и определить, какие признаки не важны для классификации/регрессии;
* Зависимость от выбранной метрики расстояния между примерами. Можно отыскать хорошее решение перебором параметров, но для большого набора данных это отнимает много времени;
* Нет теоретических оснований выбора определенного числа соседей.
* Как правило, плохо работает, когда признаков много.

**sklearn.linear\_model.LogisticRegression**

logit = LogisticRegression(C=1, n\_jobs=-1)

**Настройка параметров:**  
params = {'C': np.logspace(-50, 0, 6)}

grid\_logit = GridSearchCV(logit, params, cv=5, verbose=True)

**Linear models**

**Плюсы:**

* Очень быстрые
* Могут работать на очень больших выборках
* Практически вне конкуренции, когда признаков очень много (от сотен тысяч и более), и они разреженные.
* Коэффициенты перед признаками могут интерпретироваться
* Логистическая регрессия выдает вероятности отнесения к разным классам
* Модель может строить и нелинейную границу, если на вход подать полиномиальные признаки

**Минусы:**

* Плохо работают в задачах, в которых зависимость ответов от признаков сложная, нелинейная
* На практике предположения теоремы Маркова-Гаусса почти никогда не выполняются, поэтому чаще линейные методы работают хуже, чем, например, SVM и ансамбли (по качеству решения задачи классификации/регрессии)

**Бутстрэп -** (Мешок с шариками, из которого равномерно достают шарики с возвращением.)

**Бэггинг –** ансамбль моделей с ипользованием бутстрэпа

BaggingRegressor(DecisionTreeClassifier())

Бэггинг эффективен на малых выборках

Случайный лес лучше чем бэггинг

**Out-of-Bag оценка** — это усредненная оценка базовых алгоритмов на тех ~37% данных, на которых они не обучались.

**sklearn.ensemble.RandomForestClassifier**

* n\_estimators — число деревьев в "лесу"
* criterion — критерий для разбиения выборки в вершине
* max\_features — число признаков, по которым ищется разбиение
* min\_samples\_leaf — минимальное число объектов в листе
* max\_depth — максимальная глубина дерева

**Плюсы**:  
— имеет высокую точность предсказания, на большинстве задач будет лучше линейных алгоритмов; точность сравнима с точностью бустинга  
— практически не чувствителен к выбросам в данных из-за случайного сэмлирования  
— не чувствителен к масштабированию (и вообще к любым монотонным преобразованиям)   
— не требует тщательной настройки параметров, хорошо работает «из коробки». С помощью «тюнинга» параметров можно достичь прироста от 0.5 до 3% точности в зависимости от задачи и данных  
— способен эффективно обрабатывать данные с большим числом признаков и классов  
— одинаково хорошо обрабатывет как непрерывные, так и дискретные признаки  
— редко переобучается, на практике добавление деревьев почти всегда только улучшает композицию, но на валидации, после достижения определенного количества деревьев, кривая обучения выходит на асимптоту  
— для случайного леса существуют методы оценивания значимости отдельных признаков в модели  
— хорошо работает с пропущенными данными;  
— вычисляет близость между парами объектов, которые могут использоваться при кластеризации, обнаружении выбросов или (путем масштабирования) дают интересные представления данных  
— возможности, описанные выше, могут быть расширены до неразмеченных данных, что приводит к возможности делать кластеризацию и визуализацию данных, обнаруживать выбросы

**Минусы**:  
— в отличие от одного дерева, результаты случайного леса сложнее интерпретировать  
— нет формальных выводов (p-values), доступных для оценки важности переменных  
— алгоритм работает хуже многих линейных методов, когда в выборке очень много разреженных признаков (тексты, Bag of words)  
— случайный лес не умеет экстраполировать, в отличие от той же линейной регрессии

— алгоритм склонен к переобучению на некоторых задачах, особенно на зашумленных данных  
— для данных, включающих категориальные переменные с различным количеством уровней, случайные леса предвзяты в пользу признаков с большим количеством уровней: когда у признака много уровней, дерево будет сильнее подстраиваться именно под эти признаки, так как на них можно получить более высокое значение оптимизируемого функционала (типа прироста информации)  
— если данные содержат группы коррелированных признаков, имеющих схожую значимость для меток, то предпочтение отдается небольшим группам перед большими  
— больший размер получающихся моделей. Требуется O(NK) памяти для хранения модели, где K — число деревьев.

В библиотеке **Scikit-learn** классификаторы и регрессоры, обучаемые стохастическим градиентным спуском, реализованы классами **SGDClassifier и SGDRegressor из sklearn.linear\_model.**