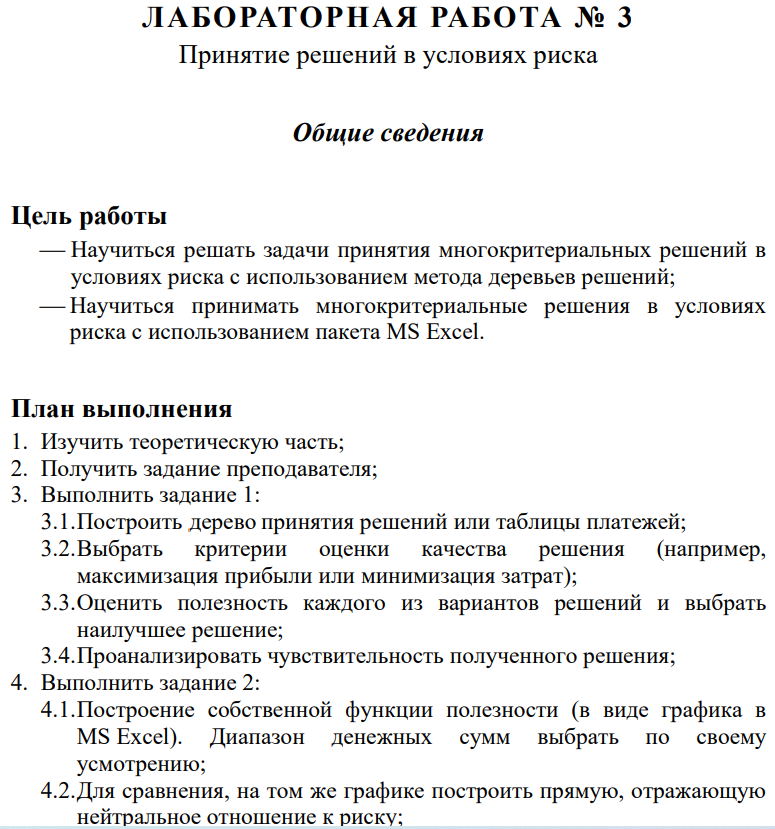
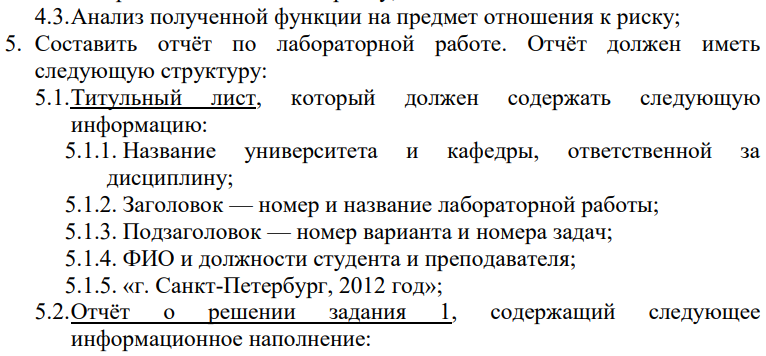
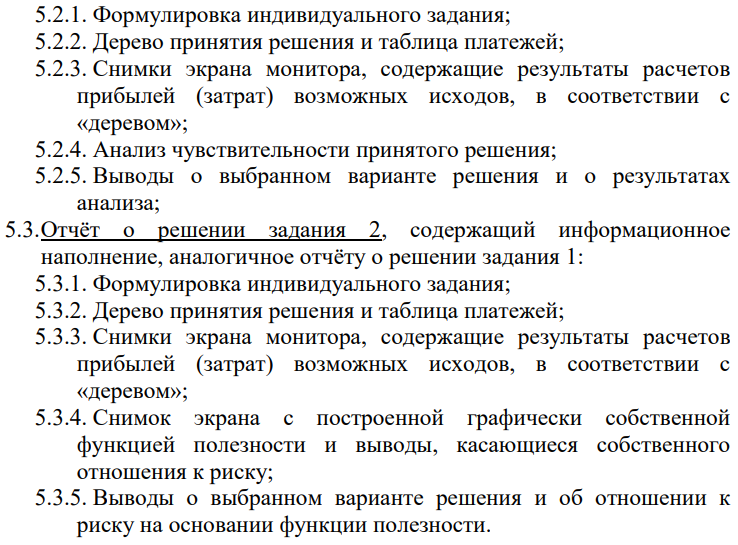
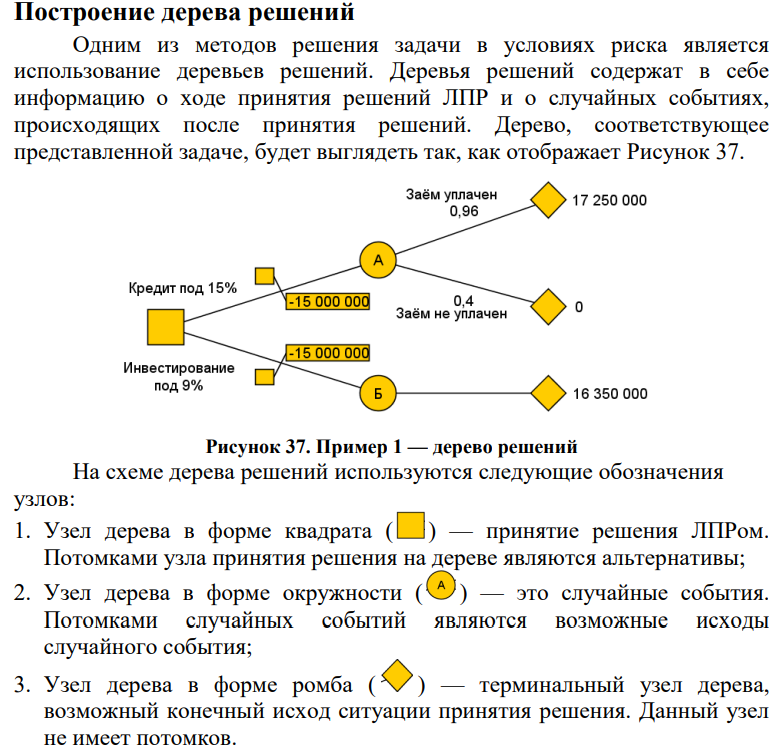
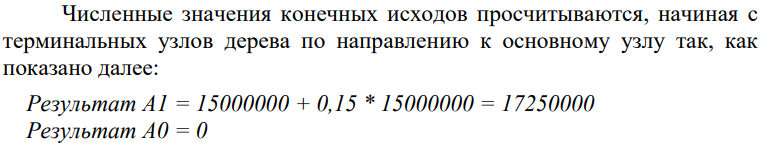
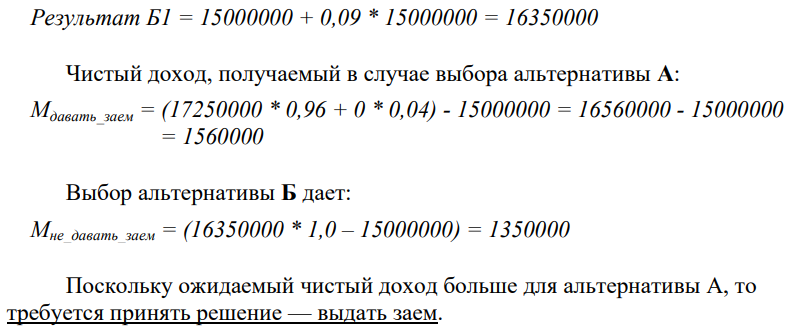
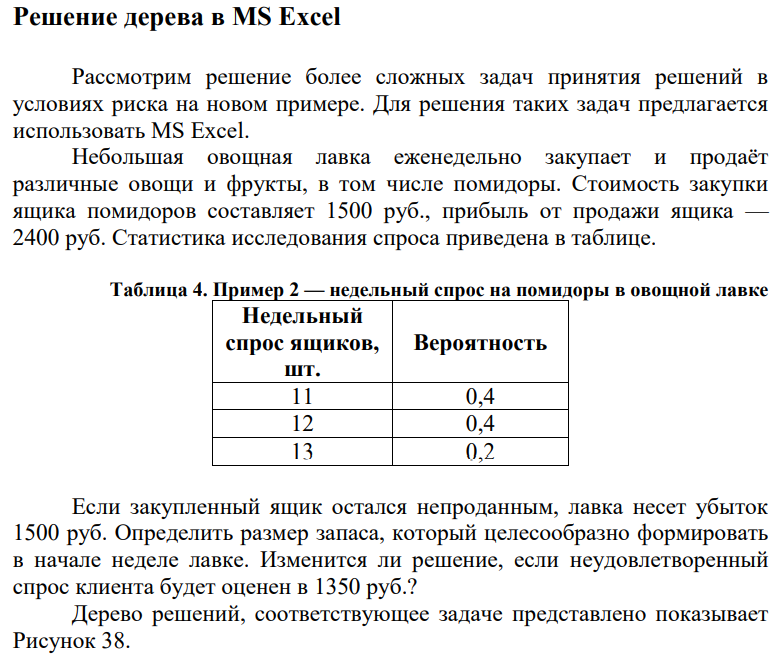
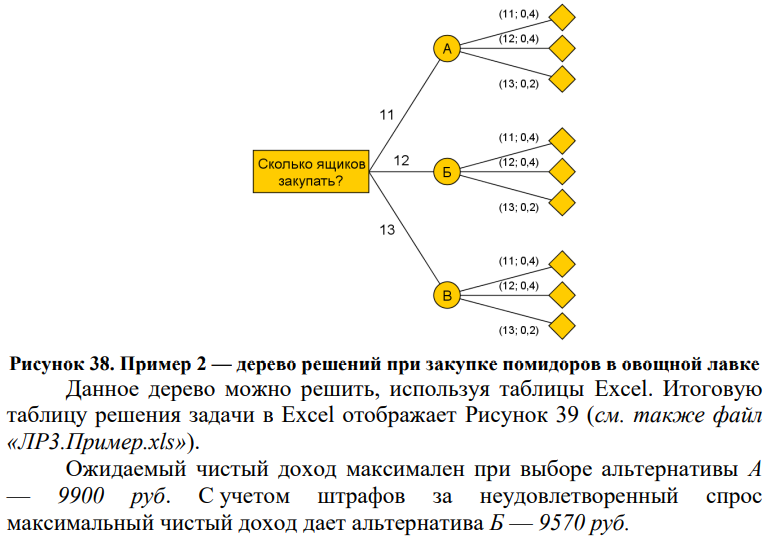
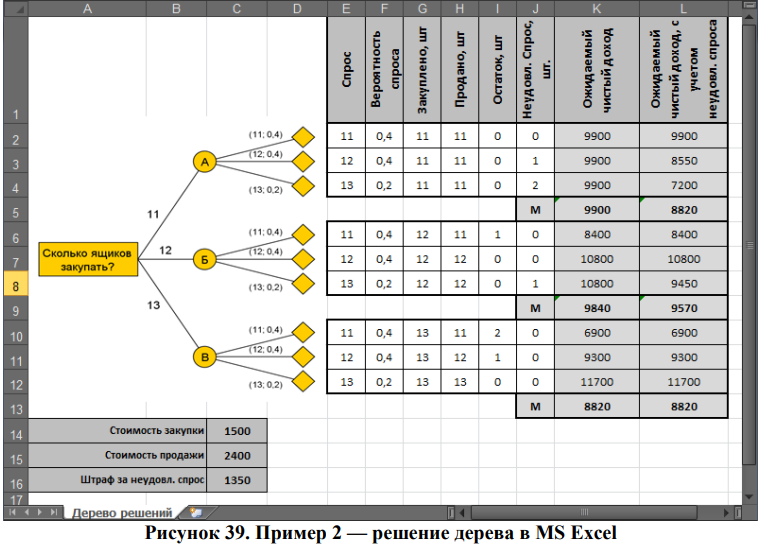
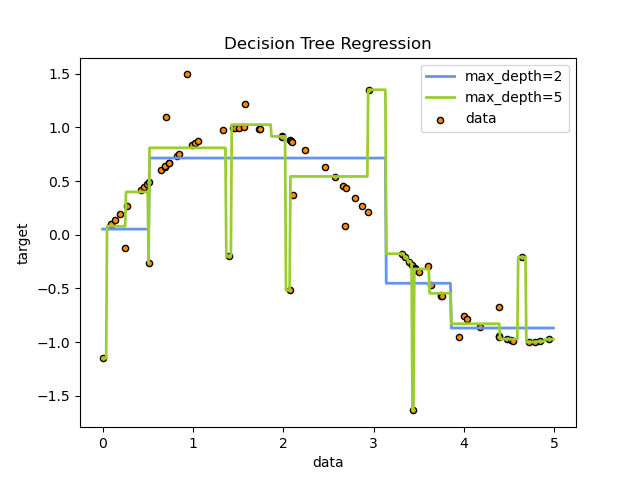
Практическая работа №6

**Деревья принятия решений и их реализация.**

**** ****      

**ТЕОРЕТИЧЕСКАЯ ЧАСТЬ:**

**Деревья решений**



Некоторые преимущества деревьев решений:

* Просто понять и интерпретировать. Деревья можно визуализировать.
* Требуется небольшая подготовка данных. Другие методы часто требуют нормализации данных, создания фиктивных переменных и удаления пустых значений. Однако обратите внимание, что этот модуль не поддерживает отсутствующие значения.
* Стоимость использования дерева (т. Е. Прогнозирования данных) является логарифмической по количеству точек данных, используемых для обучения дерева.
* Может обрабатывать как числовые, так и категориальные данные. Однако реализация scikit-learn пока не поддерживает категориальные переменные. Другие методы обычно специализируются на анализе наборов данных, содержащих только один тип переменных. См. [Алгоритмы](https://scikit-learn.ru/1-10-decision-trees/#tree-algorithms) для получения дополнительной информации.
* Способен обрабатывать проблемы с несколькими выходами.
* Использует модель белого ящика. Если данная ситуация наблюдаема в модели, объяснение условия легко объяснить с помощью булевой логики. Напротив, в модели черного ящика (например, в искусственной нейронной сети) результаты могут быть труднее интерпретировать.
* Возможна проверка модели с помощью статистических тестов. Это позволяет учитывать надежность модели.
* Работает хорошо, даже если его предположения несколько нарушаются истинной моделью, на основе которой были сгенерированы данные.

К недостаткам деревьев решений можно отнести:

* Обучающиеся дереву решений могут создавать слишком сложные деревья, которые плохо обобщают данные. Это называется переобучением. Чтобы избежать этой проблемы, необходимы такие механизмы, как обрезка, установка минимального количества выборок, необходимых для конечного узла, или установка максимальной глубины дерева.
* Деревья решений могут быть нестабильными, поскольку небольшие изменения в данных могут привести к созданию совершенно другого дерева. Эта проблема смягчается за счет использования деревьев решений в ансамбле.
* Как видно из рисунка выше, предсказания деревьев решений не являются ни гладкими, ни непрерывными, а являются кусочно-постоянными приближениями. Следовательно, они не годятся для экстраполяции.
* Известно, что проблема обучения оптимальному дереву решений является NP-полной с точки зрения нескольких аспектов оптимальности и даже для простых концепций. Следовательно, практические алгоритмы обучения дереву решений основаны на эвристических алгоритмах, таких как жадный алгоритм, в котором локально оптимальные решения принимаются в каждом узле. Такие алгоритмы не могут гарантировать возврат глобального оптимального дерева решений. Это можно смягчить путем обучения нескольких деревьев в учащемся ансамбля, где функции и образцы выбираются случайным образом с заменой.
* Существуют концепции, которые трудно изучить, поскольку деревья решений не выражают их легко, например проблемы XOR, четности или мультиплексора.
* Ученики дерева решений создают предвзятые деревья, если некоторые классы доминируют. Поэтому рекомендуется сбалансировать набор данных перед подгонкой к дереву решений.

Классификация

[**DecisionTreeClassifier**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier) — это класс, способный выполнять мультиклассовую классификацию набора данных.

Как и в случае с другими классификаторами, [DecisionTreeClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html" \l "sklearn.tree.DecisionTreeClassifier" \t "_blank) принимает в качестве входных данных два массива: массив **X**, разреженный или плотный, формы (n\_samples, n\_features), содержащий обучающие образцы, и массив **Y** целочисленных значений, формы (n\_samples,), содержащий метки классов для обучающих образцов:

1

>>> **from** sklearn **import** tree

2

>>> X = [[0, 0], [1, 1]]

3

>>> Y = [0, 1]

4

>>> clf = tree.DecisionTreeClassifier()

5

>>> clf = clf.fit(X, Y)

После подбора модель можно использовать для прогнозирования класса образцов:

1

>>> clf.predict([[2., 2.]])

2

array([1])

В случае, если существует несколько классов с одинаковой и самой высокой вероятностью, классификатор предскажет класс с самым низким индексом среди этих классов.

В качестве альтернативы выводу определенного класса можно предсказать вероятность каждого класса, которая представляет собой долю обучающих выборок класса в листе:

1

>>> clf.predict\_proba([[2., 2.]])

2

array([[0., 1.]])

[**DecisionTreeClassifier**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier) поддерживает как двоичную (где метки — [-1, 1]), так и мультиклассовую (где метки — [0,…, K-1]) классификацию.

Используя набор данных Iris, мы можем построить дерево следующим образом:



1

>>> **from** sklearn.datasets **import** load\_iris

2

>>> **from** sklearn **import** tree

3

>>> iris = load\_iris()

4

>>> X, y = iris.data, iris.target

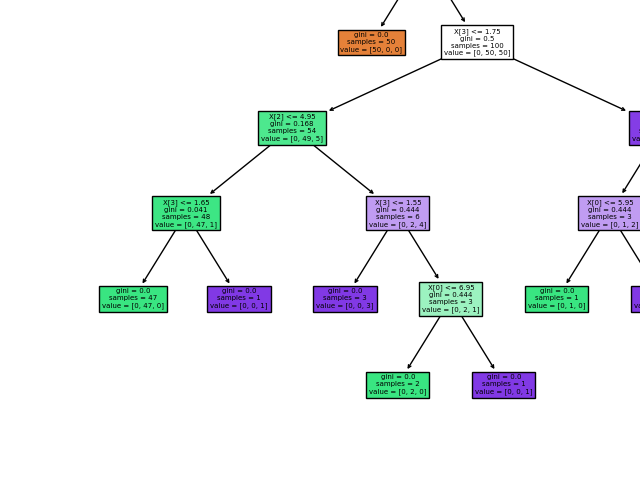
5

>>> clf = tree.DecisionTreeClassifier()

6

>>> clf = clf.fit(X, y)

После обучения вы можете построить дерево с помощью [plot\_tree](https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html" \l "sklearn.tree.plot_tree" \t "_blank) функции: >>> tree.plot\_tree(clf)



Мы также можем экспортировать дерево в формат [Graphviz](https://www.graphviz.org/" \t "_blank) с помощью [export\_graphviz](https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html" \l "sklearn.tree.export_graphviz" \t "_blank) экспортера. Если вы используете [Conda](https://conda.io/" \t "_blank) менеджер пакетов, то Graphviz бинарные файлы и пакет питон может быть установлен conda install python-graphviz.

В качестве альтернативы двоичные файлы для graphviz можно загрузить с домашней страницы проекта graphviz, а оболочку Python установить из pypi с помощью pip install graphviz.

Ниже приведен пример экспорта graphviz вышеуказанного дерева, обученного на всем наборе данных радужной оболочки глаза; результаты сохраняются в выходном файле iris.pdf:



1

>>> **import** graphviz

2

>>> dot\_data = tree.export\_graphviz(clf, out\_file=**None**)

3

>>> graph = graphviz.Source(dot\_data)

4

>>> graph.render("iris")

Экспортер **[export\_graphviz](https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html" \l "sklearn.tree.export_graphviz" \t "_blank)** также поддерживает множество эстетических вариантов, в том числе окраски узлов их класс (или значение регрессии) и используя явные имена переменных и классов , если это необходимо. Блокноты Jupyter также автоматически отображают эти графики встроенными:



1

>>> dot\_data = tree.export\_graphviz(clf, out\_file=**None**,

2

... feature\_names=iris.feature\_names,

3

... class\_names=iris.target\_names,

4

... filled=**True**, rounded=**True**,

5

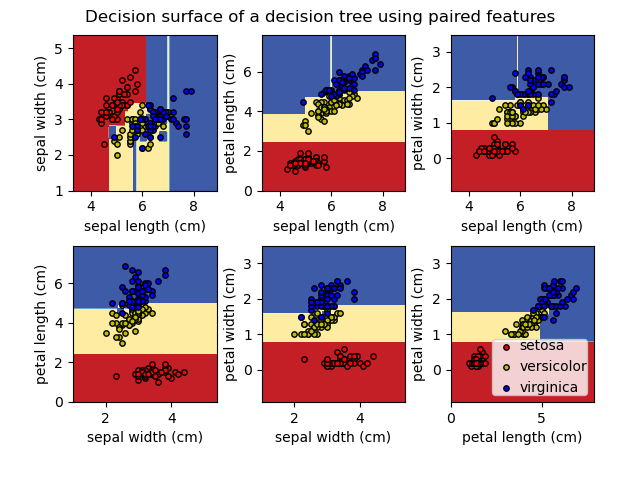
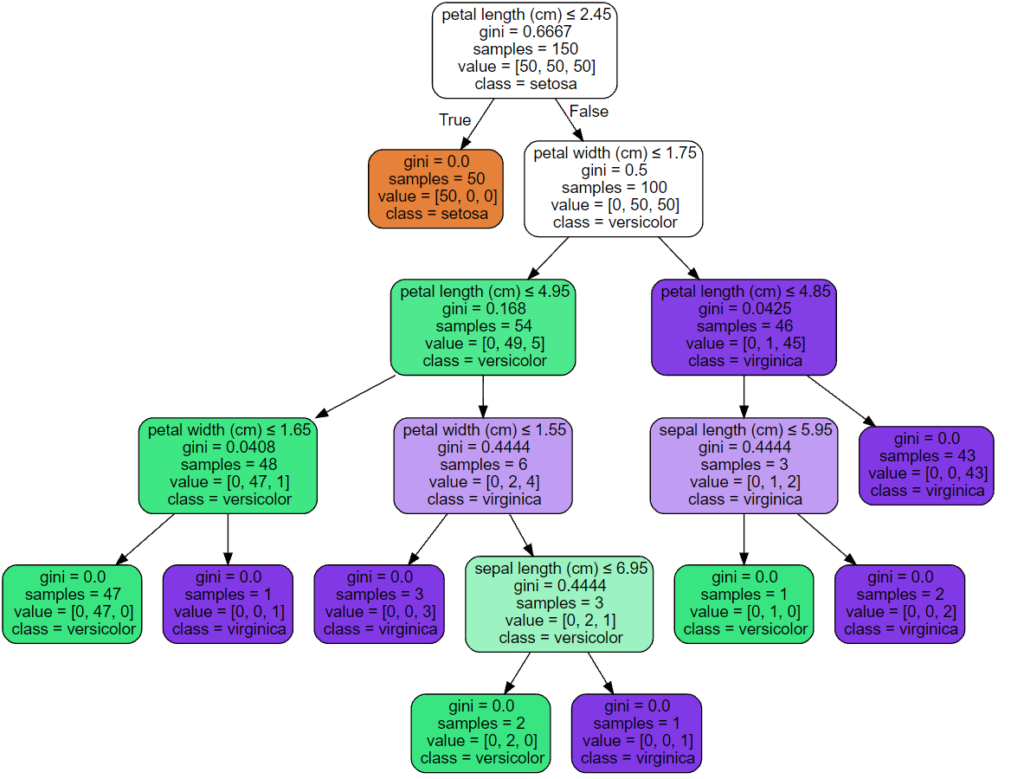
... special\_characters=**True**)

6

>>> graph = graphviz.Source(dot\_data)

7

>>> graph



В качестве альтернативы дерево можно также экспортировать в текстовый формат с помощью функции **[export\_text](https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_text.html" \l "sklearn.tree.export_text" \t "_blank)**. Этот метод не требует установки внешних библиотек и более компактен:



1

>>> **from** sklearn.datasets **import** load\_iris

2

>>> **from** sklearn.tree **import** DecisionTreeClassifier

3

>>> **from** sklearn.tree **import** export\_text

4

>>> iris = load\_iris()

5

>>> decision\_tree = DecisionTreeClassifier(random\_state=0, max\_depth=2)

6

>>> decision\_tree = decision\_tree.fit(iris.data, iris.target)

7

>>> r = export\_text(decision\_tree, feature\_names=iris['feature\_names'])

8

>>> print(r)

9

|--- petal width (cm) <= 0.80

10

| |--- **class**: 0

11

|--- petal width (cm) > 0.80

12

| |--- petal width (cm) <= 1.75

13

| | |--- **class**: 1

14

| |--- petal width (cm) > 1.75

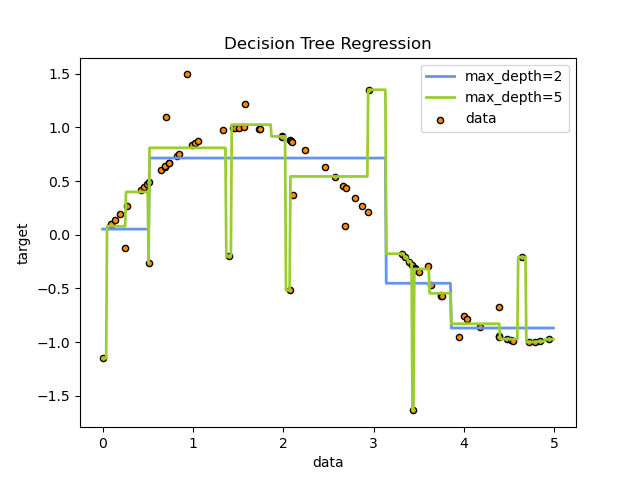
15

| | |--- **class**: 2

**Примеры**

* [Постройте поверхность принятия решений дерева решений на наборе данных радужной оболочки глаза](https://scikit-learn.ru/example/422/)
* [Понимание структуры дерева решений](https://scikit-learn.ru/example/424/)

Регрессия



Деревья решений также могут применяться к задачам регрессии с помощью класса [DecisionTreeRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html" \l "sklearn.tree.DecisionTreeRegressor" \t "_blank) .

Как и в настройке классификации, метод fit будет принимать в качестве аргументов массивы X и y, только в этом случае ожидается, что y будет иметь значения с плавающей запятой вместо целочисленных значений:



1

>>> **from** sklearn **import** tree

2

>>> X = [[0, 0], [2, 2]]

3

>>> y = [0.5, 2.5]

4

>>> clf = tree.DecisionTreeRegressor()

5

>>> clf = clf.fit(X, y)

6

>>> clf.predict([[1, 1]])

7

array([0.5])

**Пример:**

* [Регрессия дерева решений](https://scikit-learn.ru/example/420/)

Проблемы с несколькими выходами

Задача с несколькими выходами — это проблема контролируемого обучения с несколькими выходами для прогнозирования, то есть когда Y — это 2-й массив формы (n\_samples, n\_outputs).

Когда нет корреляции между выходами, очень простой способ решить эту проблему — построить n независимых моделей, то есть по одной для каждого выхода, а затем использовать эти модели для независимого прогнозирования каждого из n выходов. Однако, поскольку вполне вероятно, что выходные значения, относящиеся к одному и тому же входу, сами коррелированы, часто лучшим способом является построение единой модели, способной прогнозировать одновременно все n выходов. Во-первых, это требует меньшего времени на обучение, поскольку строится только один оценщик. Во-вторых, часто можно повысить точность обобщения итоговой оценки.

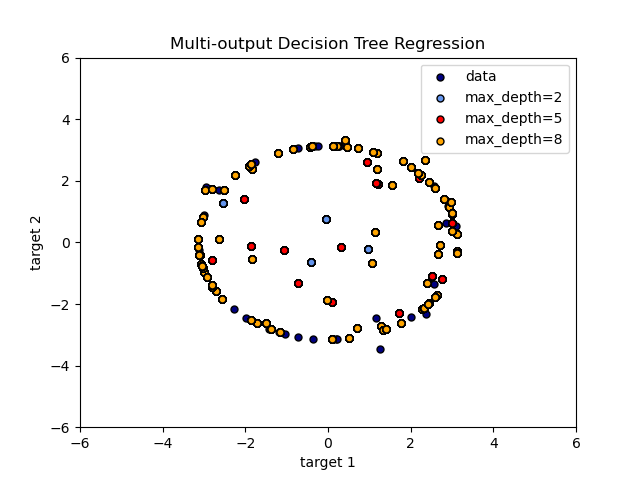
Что касается деревьев решений, эту стратегию можно легко использовать для поддержки задач с несколькими выходами. Для этого требуются следующие изменения:

* Сохранять n выходных значений в листьях вместо 1;
* Используйте критерии разделения, которые вычисляют среднее сокращение для всех n выходов.

Этот модуль предлагает поддержку для задач с несколькими выходами, реализуя эту стратегию как в, так [DecisionTreeClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html" \l "sklearn.tree.DecisionTreeClassifier" \t "_blank) и в **[DecisionTreeRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html" \l "sklearn.tree.DecisionTreeRegressor" \t "_blank)**. Если дерево решений соответствует выходному массиву Y формы (n\_samples, n\_outputs), то итоговая оценка будет:

* Вывести значения n\_output при predict;
* Выведите список массивов n\_output вероятностей классов на predict\_proba.

Использование деревьев с несколькими выходами для регрессии продемонстрировано в [разделе «Регрессия дерева решений с несколькими выходами»](https://scikit-learn.ru/example/421/) . В этом примере вход X — это одно действительное значение, а выходы Y — синус и косинус X.



Использование деревьев с несколькими выходами для классификации демонстрируется в [разделе «Завершение лица с оценками с несколькими выходами»](https://scikit-learn.ru/example/face-completion-with-a-multi-output-estimators/) . В этом примере входы X — это пиксели верхней половины граней, а выходы Y — пиксели нижней половины этих граней.



**Примеры:**

* [Регрессия дерева решений с несколькими выходами](https://scikit-learn.ru/example/421/)
* [Завершение лица с помощью многовыходных оценщиков](https://scikit-learn.ru/example/face-completion-with-a-multi-output-estimators/)

**Рекомендации:**

* М. Дюмон и др., [Быстрая мультиклассовая аннотация изображений со случайными подокнами и множественными выходными рандомизированными деревьями](http://www.montefiore.ulg.ac.be/services/stochastic/pubs/2009/DMWG09/dumont-visapp09-shortpaper.pdf) , Международная конференция по теории и приложениям компьютерного зрения, 2009 г.

Сложность

В общем, время выполнения для построения сбалансированного двоичного дерева составляет O(nsamplesnfeatureslog⁡(nsamples)) и время запроса O(log⁡(nsamples)). Хотя алгоритм построения дерева пытается генерировать сбалансированные деревья, они не всегда будут сбалансированными. Предполагая, что поддеревья остаются примерно сбалансированными, стоимость на каждом узле состоит из перебора O(nfeatures) найти функцию, обеспечивающую наибольшее снижение энтропии. Это стоит O(nfeaturesnsampleslog⁡(nsamples)) на каждом узле, что приводит к общей стоимости по всем деревьям (суммируя стоимость на каждом узле) O(nfeaturesnsamples2log⁡(nsamples))

Советы по практическому использованию

* Деревья решений имеют тенденцию чрезмерно соответствовать данным с большим количеством функций. Получение правильного соотношения образцов к количеству функций важно, поскольку дерево с небольшим количеством образцов в многомерном пространстве, скорее всего, переоборудуется.
* Предварительно рассмотрите возможность уменьшения размерности ([PCA](https://scikit-learn.org/stable/modules/decomposition.html#pca), [ICA](https://scikit-learn.org/stable/modules/decomposition.html#ica), или [Feature selection](https://scikit-learn.org/stable/modules/feature_selection.html" \l "feature-selection" \t "_blank)), чтобы дать вашему дереву больше шансов найти отличительные признаки.
* [Понимание структуры дерева решений](https://scikit-learn.ru/example/424/) поможет лучше понять, как дерево решений делает прогнозы, что важно для понимания важных функций данных.
* Визуализируйте свое дерево во время тренировки с помощью export функции. Используйте max\_depth=3 в качестве начальной глубины дерева, чтобы понять, насколько дерево соответствует вашим данным, а затем увеличьте глубину.
* Помните, что количество образцов, необходимых для заполнения дерева, удваивается для каждого дополнительного уровня, до которого дерево растет. Используйте max\_depth для управления размером дерева во избежание переобучения.
* Используйте min\_samples\_split или, min\_samples\_leaf чтобы гарантировать, что несколько выборок информируют каждое решение в дереве, контролируя, какие разделения будут учитываться. Очень маленькое число обычно означает, что дерево переоборудуется, тогда как большое число не позволяет дереву изучать данные. Попробуйте min\_samples\_leaf=5 в качестве начального значения. Если размер выборки сильно различается, в этих двух параметрах можно использовать число с плавающей запятой в процентах. В то время как min\_samples\_split может создавать произвольно маленькие листья, min\_samples\_leaf гарантирует, что каждый лист имеет минимальный размер, избегая малодисперсных, чрезмерно подходящих листовых узлов в задачах регрессии. Для классификации с несколькими классами min\_samples\_leaf=1 это часто лучший выбор.  
    
  Обратите внимание, что min\_samples\_split выборки рассматриваются напрямую и независимо от них sample\_weight, если они предусмотрены (например, узел с m взвешенными выборками по-прежнему обрабатывается как имеющий ровно m выборок). Рассмотрим min\_weight\_fraction\_leaf или min\_impurity\_decrease если учет образцов весов требуются при расколах.
* Перед обучением сбалансируйте набор данных, чтобы дерево не смещалось в сторону доминирующих классов. Балансировка классов может быть выполнена путем выборки равного количества выборок из каждого класса или, предпочтительно, путем нормализации суммы весов выборок (sample\_weight) для каждого класса к одному и тому же значению. Также обратите внимание, что критерии предварительного отсечения на основе веса, такие как min\_weight\_fraction\_leaf, будут менее смещены в сторону доминирующих классов, чем критерии, которые не знают весов выборки, например min\_samples\_leaf.
* Если выборки взвешены, будет проще оптимизировать древовидную структуру, используя основанный на весе критерий предварительной отсечения, например min\_weight\_fraction\_leaf, который гарантирует, что конечные узлы содержат по крайней мере часть общей суммы весов выборки.
* Все деревья решений np.float32 внутренне используют массивы. Если данные обучения не в этом формате, будет сделана копия набора данных.
* Если входная матрица X очень разреженная, рекомендуется преобразовать ее в разреженную csc\_matrix перед вызовом соответствия и разреженную csr\_matrix перед вызовом предсказания. Время обучения может быть на порядки меньше для входной разреженной матрицы по сравнению с плотной матрицей, когда функции имеют нулевые значения в большинстве выборок.

Алгоритмы дерева: ID3, C4.5, C5.0 и CART

Что представляют собой различные алгоритмы дерева решений и чем они отличаются друг от друга? Какой из них реализован в scikit-learn?

[ID3](https://en.wikipedia.org/wiki/ID3_algorithm) (Iterative Dichotomiser 3) был разработан Россом Куинланом в 1986 году. Алгоритм создает многостороннее дерево, находя для каждого узла (т. Е. Жадным образом) категориальный признак, который даст наибольший информационный выигрыш для категориальных целей. Деревья вырастают до максимального размера, а затем обычно применяется этап обрезки, чтобы улучшить способность дерева обобщать невидимые данные.

C4.5 является преемником ID3 и снял ограничение, что функции должны быть категориальными, путем динамического определения дискретного атрибута (на основе числовых переменных), который разбивает непрерывное значение атрибута на дискретный набор интервалов. C4.5 преобразует обученные деревья (т. Е. Результат алгоритма ID3) в наборы правил «если-то». Затем оценивается точность каждого правила, чтобы определить порядок, в котором они должны применяться. Удаление выполняется путем удаления предусловия правила, если без него точность правила улучшается.

C5.0 — это последняя версия Quinlan под частной лицензией. Он использует меньше памяти и создает меньшие наборы правил, чем C4.5, но при этом является более точным.

[CART](https://en.wikipedia.org/wiki/Predictive_analytics#Classification_and_regression_trees_.28CART.29) (Classification and Regression Trees — деревья классификации и регрессии) очень похож на C4.5, но отличается тем, что поддерживает числовые целевые переменные (регрессию) и не вычисляет наборы правил. CART строит двоичные деревья, используя функцию и порог, которые дают наибольший прирост информации в каждом узле.

scikit-learn использует оптимизированную версию алгоритма CART; однако реализация scikit-learn пока не поддерживает категориальные переменные.

Математическая постановка

Данные обучающие векторы xi∈Rn, i = 1,…, l и вектор-метка y∈Rl дерево решений рекурсивно разбивает пространство признаков таким образом, что образцы с одинаковыми метками или аналогичными целевыми значениями группируются вместе.

Пусть данные в узле m быть представлен Qm с участием Nm образцы. Для каждого раскола кандидатов θ=(j,tm) состоящий из функции j и порог tm, разделите данные на Qmleft(θ) а также Qmright(θ) подмножества  
Qmleft(θ)=(x,y)|xj<=tm  
Qmright(θ)=Qm∖Qmleft(θ)

Качество кандидата разделения узла m затем вычисляется с использованием функции примеси или функции потерь H(), выбор которых зависит от решаемой задачи (классификация или регрессия)  
G(Qm,θ)=NmleftNmH(Qmleft(θ))+NmrightNmH(Qmright(θ))

Выберите параметры, которые минимизируют примеси  
θ∗=argminθ⁡G(Qm,θ)

Рекурсия для подмножеств Qmleft(θ∗) а также Qmright(θ∗) пока не будет достигнута максимально допустимая глубина, Nm<minsamples или же Nm=1.

1.10.7.1. Критерии классификации

Если целью является результат классификации, принимающий значения 0,1,…, K-1, для узла m, позволять  
pmk=1/Nm∑y∈QmI(y=k)

быть пропорцией наблюдений класса k в узле m. Еслиmявляется конечным узлом, predict\_proba для этого региона установлено значение pmk. Общие меры примеси следующие.

Джини:  
H(Qm)=∑kpmk(1—pmk)

Энтропия:  
H(Qm)=—∑kpmklog⁡(pmk)

Неверная классификация:  
H(Qm)=1—max(pmk)

1.10.7.2. Критерии регрессии

Если целью является непрерывное значение, то для узла m, общими критериями, которые необходимо минимизировать для определения местоположений будущих разделений, являются среднеквадратичная ошибка (ошибка MSE или L2), отклонение Пуассона, а также средняя абсолютная ошибка (ошибка MAE или L1). MSE и отклонение Пуассона устанавливают прогнозируемое значение терминальных узлов равным изученному среднему значению y¯m узла, тогда как MAE устанавливает прогнозируемое значение терминальных узлов равным медиане median(y)m.

Среднеквадратичная ошибка:  
y¯m=1Nm∑y∈Qmy  
H(Qm)=1Nm∑y∈Qm(y—y¯m)2

Половинное отклонение Пуассона:  
H(Qm)=1Nm∑y∈Qm(ylog⁡yy¯m—y+y¯m)

Настройка criterion="poisson" может быть хорошим выбором, если ваша цель — счетчик или частота (количество на какую-то единицу). В любом случае, y>=0 является необходимым условием для использования этого критерия. Обратите внимание, что он подходит намного медленнее, чем критерий MSE.

Средняя абсолютная ошибка:  
median(y)m=mediany∈Qm(y)  
H(Qm)=1Nm∑y∈Qm|y—median(y)m|

Обратите внимание, что он подходит намного медленнее, чем критерий MSE.

Обрезка с минимальными затратами и сложностью

Сокращение с минимальными затратами и сложностью — это алгоритм, используемый для сокращения дерева во избежание чрезмерной подгонки, описанный в главе 3 [[BRE]](https://scikit-learn.ru/1-10-decision-trees/#bre) . Этот алгоритм параметризован α≥0 известный как параметр сложности. Параметр сложности используется для определения меры затрат и сложности, Rα(T)  данного дерева T:  
Rα(T)=R(T)+α|T~|

где |T~| количество конечных узлов в T а также R(T) традиционно определяется как общий коэффициент ошибочной классификации конечных узлов. В качестве альтернативы scikit-learn использует взвешенную общую примесь конечных узлов для R(T). Как показано выше, примесь узла зависит от критерия. Обрезка с минимальными затратами и сложностью находит поддеревоT что сводит к минимуму Rα(T).

Оценка сложности стоимости одного узла составляет Rα(t)=R(t)+α. Ответвление Tt, определяется как дерево, в котором узел t это его корень. В общем, примесь узла больше, чем сумма примесей его конечных узлов, R(Tt)<R(t). Однако мера стоимости и сложности узла, t, и его ветвь, Tt, может быть равным в зависимости от α. Определяем эффективныйα узла быть значением, где они равны, Rα(Tt)=Rα(t) или же αeff(t)=R(t)−R(Tt)|T|−1. Нетерминальный узел с наименьшим значением αeff является самым слабым звеном и будет удалено. Этот процесс останавливается, когда обрезанное дерево минимально αeff больше ccp\_alpha параметра.

**Примеры:**

* [Публикация деревьев решений об обрезке с сокращением сложности затрат](https://scikit-learn.ru/example/423/)

**Рекомендации:**

* [BRE](https://scikit-learn.ru/1-10-decision-trees/#bre) Л. Брейман, Дж. Фридман, Р. Олшен и К. Стоун. Деревья классификации и регрессии. Уодсворт, Белмонт, Калифорния, 1984.
* <https://en.wikipedia.org/wiki/Decision_tree_learning>
* <https://en.wikipedia.org/wiki/Predictive_analytics>
* JR Quinlan. C4. 5: программы для машинного обучения. Морган Кауфманн, 1993.
* Т. Хасти, Р. Тибширани и Дж. Фридман. Элементы статистического обучения, Springer, 2009.