Graphviz

```
pip install graphviz
```

Загрузить (взять у преподавателя) файл ram_price.csv

(содержит исторические данные о ценах на компьютерную память)

3.6 Линейные модели регрессии в задачах дискретной классификации

3.6.1 Бинарная классификация

Уравнения классификатора для расчета оценки (классификации) (n признаков):

$$\{y_{est} = w_1 x_1 + w_2 x_2 + \mathrm{L} + w_n x_n + b\} \geq 0$$

Рассчитываемая оценка является непрерывной величиной. К ней применяется пороговая функция (например, пороговое значение =0).

Т.о., бинарный линейный классификатор – это классификатор, который разделяет два класса с помощью линии (плоскости, гиперплоскости).

Существует множество алгоритмов обучения линейных моделей. Характеризуются в основном параметрами:

-какая метрика;

-используется ли регуляризации, какая регуляризация;

3.6.1 Бинарная классификация

Наиболее распространенные:

-логистическая регрессия (logistic regression), реализованная в классе linear_model.LogisticRegression;

-линейный метод опорных векторов (linear support vector machines SVM), реализованный в классе svm.LinearSVC (support vector classifier – классификатор опорных векторов).

Несмотря на название, логистическая регрессия является алгоритмом классификации, а не алгоритмом регрессии.

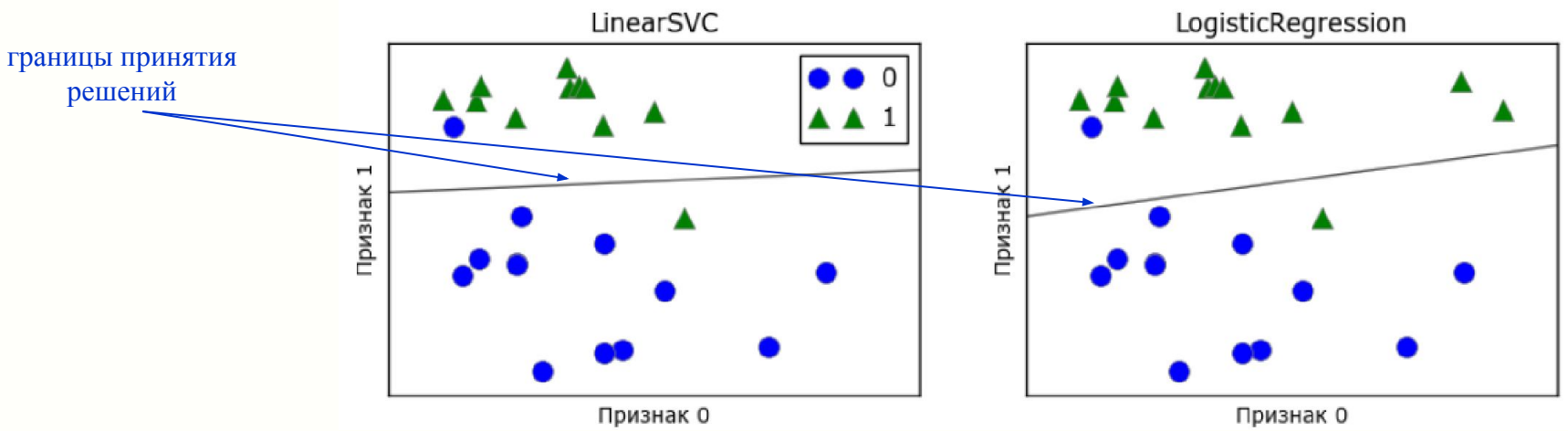
Используем набор forge из библиотеки mglearn. Применим два классификатора:

-LogisticRegression из sklearn.linear_model;

- LinearSVC из sklearn.svm. По умолчанию обе модели используют L2 регуляризацию.

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
X, y = mglearn.datasets.make_forge()
fig, axes = plt.subplots(1, 2, figsize=(10, 3))
for model, ax in zip([LinearSVC(), LogisticRegression()], axes):
    clf = model.fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=False, eps=0.5,ax=ax, alpha=.7)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{}".format(clf.__class__.__name__))
    ax.set_ylabel("Признак 1")
axes[0].legend()
```

3.6.1 Бинарная классификация



Границы принятия решений отделяют 2 области: область значений, классифицированных как класс 1 (верхняя часть графика), и как класс 0. Классификаторы строят схожие границы принятия решений – оба неправильно классифицировали по 2 точки.

Параметр регуляризации в обоих классификаторах называется C:

-более высокие значения C соответствуют меньшей регуляризации ⇒ качество работы модели на обучающем наборе улучшается с $\uparrow C$ **(а как в Ridge и в Lasso?)**;

-чем $C \downarrow$, тем значения коэффициентов w_i больше $\rightarrow 0$.

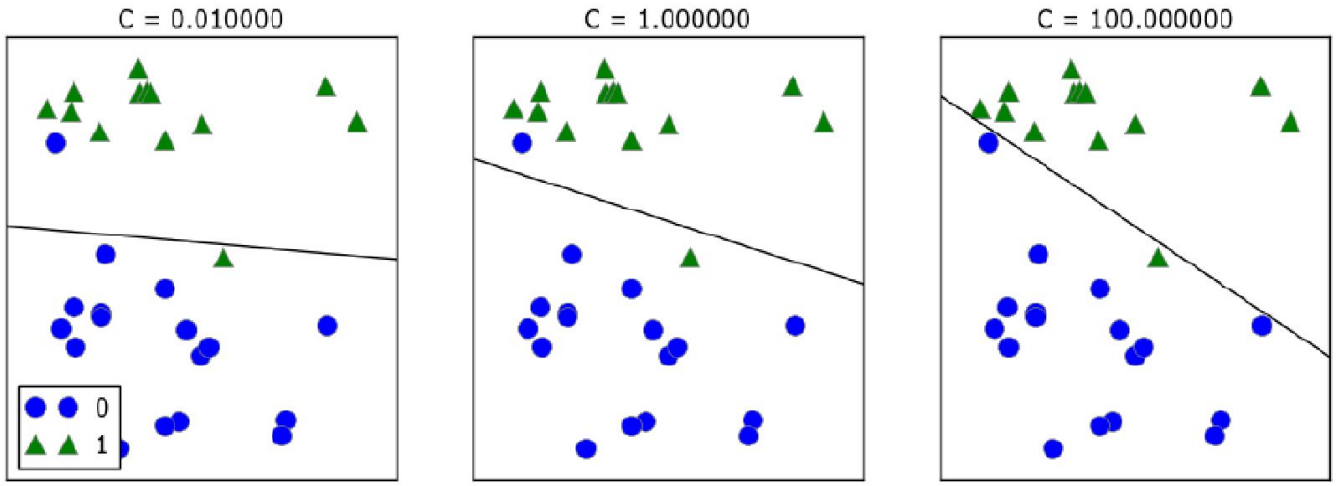
Следовательно:

при $C \downarrow$ алгоритмы пытаются подстроиться под «большинство» точек данных

$C \uparrow$ подчеркивает важность того, чтобы каждая отдельная точка данных была классифицирована правильно

3.6.1 Бинарная классификация

```
# зависимость работы классификатора лин. SVM от параметра регуляризации C
mglearn.plots.plot_linear_svc_regularization()
```



Сильно регуляризованная модель дает почти горизонтальную линию.

Очень высокое *C* сильно наклоняет границу, правильно классифицируя все точки класса 0. Но одна из точек класса 1 при этом классифицирована неправильно. Т.К. невозможно при использовании линейной границы принятия решения.

Скорее всего, классификатор при *C*=100 переобучен.

Недостаток рассмотренных моделей**:** линейность границы. Это сильно проявляется для данных низкой размерности.

НО (!) для данных большой размерности такие классификаторы эффективны, важно не переобучить (!).

3.6.1 Бинарная классификация

Для высокоразмерных наборов данных линейные модели становятся сложными и существует высокая вероятность переобучения.
Для набора cancer:

Задание9: Загрузить набор данных cancer из mglearnsklearn.datasets
Разбить на обучающий и тестовый наборы. (random_state=42)
Используя модель LogisticRegression из модуля sklearn.linear_model создать и обучить линейный классификатор. (# вызов «Все по умолчанию» приводит к предупреждениям. Стоит задать параметр solver='liblinear') . По умолчанию *C*=1.
Оценить точность классификатора на обучающем и на тестовом наборах методом score.

Неплохое качество модели – правильность на обучающем и тестовом наборах ≈ 95%.

Однако Т.К. качество модели на обучающем ≈ на тестовом наборах, то вполне вероятно, что модель недообучена.

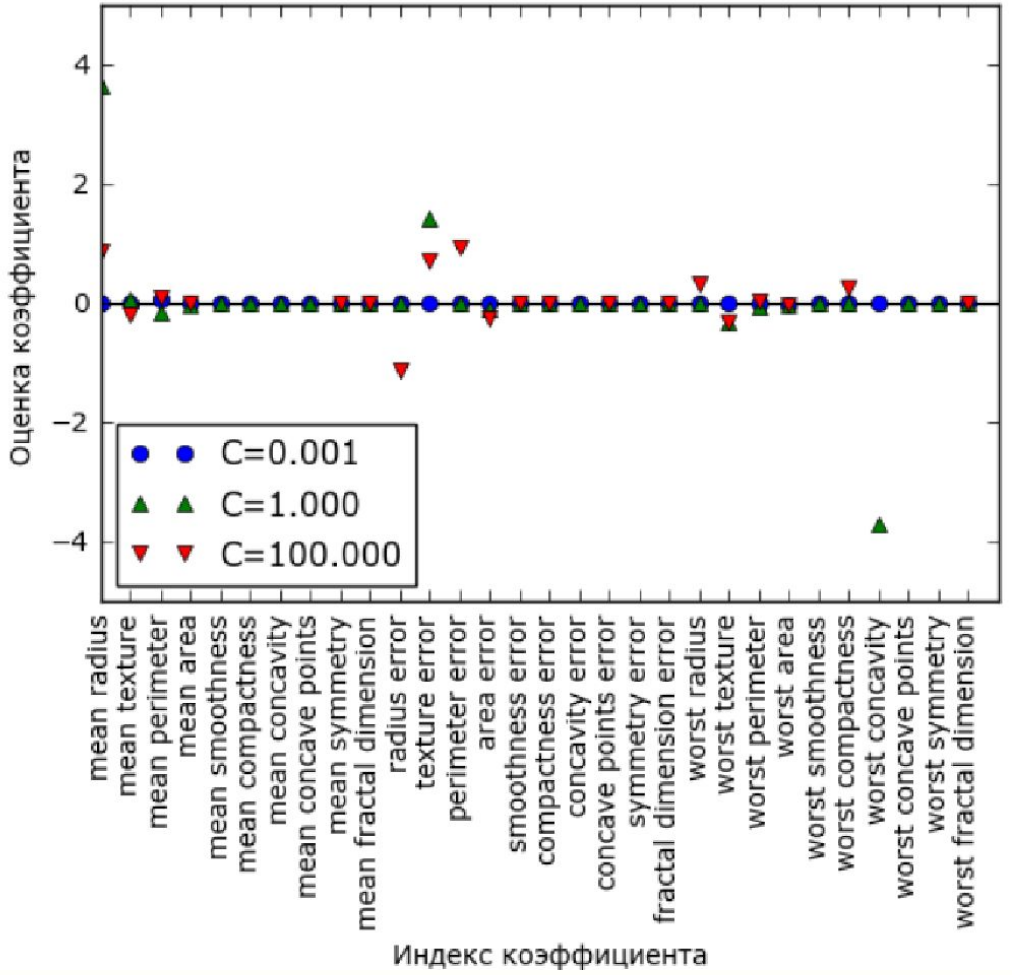
Задание10: Принять C=100 и C=0.01. Оценить качество моделей, сделать выводы.

При C=100 точность ↑ на обоих наборах ⇒ улучшилась работа в целом.
При C=0.01 модель более регуляризованная, точность ↓, модель недообучена.

3.6.1 Бинарная классификация

По умолчанию используется L_2 регуляризация.
(!) Регуляризация L_1 позволяет отобрать признаки по важности и исключить из модели неинформативные признаки, установив соответствующие коэффициенты =0. (!)

```
# исп. L1 регуляризац для оценки информативности признаков
# оценим при этом точность
for C, marker in zip([0.001, 1, 100], ['o', '^', 'v']):
lr_l1 = LogisticRegression(C=C, penalty="l1", solver='liblinear', max_iter=1000).fit(X_train, y_train)
plt.plot(lr_l1.coef_.T, marker, label="C={:.3f}".format(C))
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.hlines(0, 0, cancer.data.shape[1])
plt.xlabel("Индекс коэффициента")
plt.ylabel("Оценка коэффициента")
plt.ylim(-5, 5)
plt.legend(loc=3)
```



ИНТЕРПРЕТИРОВАТЬ КОЭФФИЦИЕНТЫ ЛИНЕЙНЫХ МОДЕЛЕЙ НУЖНО С ОСТОРОЖНОСТЬЮ И СКЕПТИЦИЗМОМ

3.6.2 Множественная классификация

Способ расширения алгоритма бинарной классификации до случаев мультиклассовой классификации называется «один против всех» (one-vs.-rest).

При этом для каждого класса строится бинарная модель, которая пытается отделить этот класс от всех остальных ⇒

⇒ количество моделей = количеству классов;

⇒ для каждого класса рассчитывается свой вектор коэффициентов $w=\{w_1, w_1, \dots\}$ и своя константа b

Для классификации (получения прогноза) новый ввод подается на все бинарные классификаторы.

Классификатор, который выдает по своему классу наибольшее значение, «побеждает» и метка этого класса возвращается в качестве прогноза.

Класс, который получает наибольшее значение согласно:

$$\max_{p \subset P} \left\{ \left(w_1^{\,1} x_1 + \mathsf{L} + w_n^{\,1} x_n + b^1 \right), \mathsf{K}, \left(w_1^{\,p} x_1 + \mathsf{L} + w_n^{\,p} x_n + b^p \right), \mathsf{K} \right\}$$

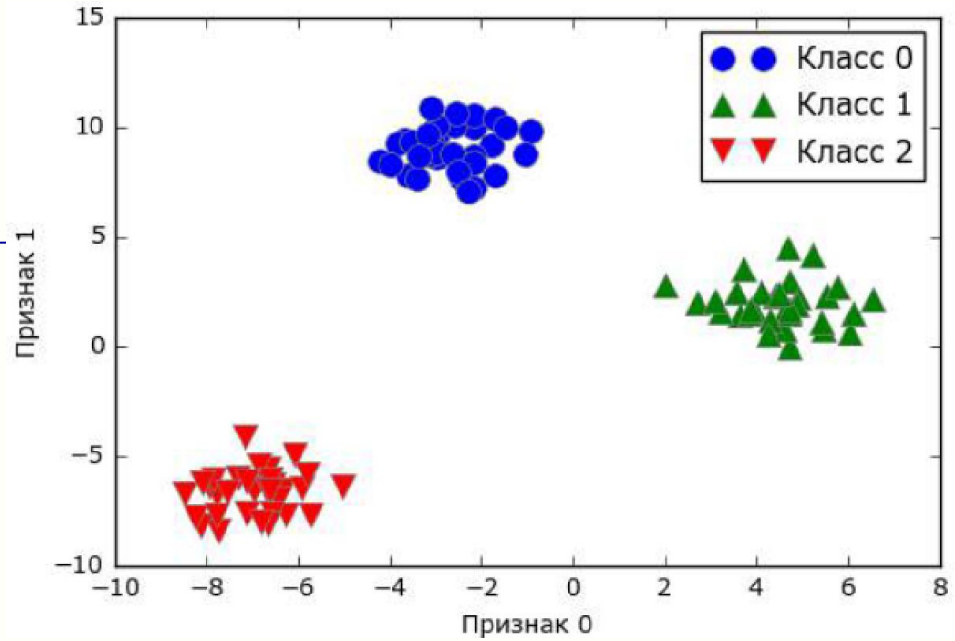
P – множество классов

становится меткой класса.

3.6.2 Множественная классификация

Применим к двумерному массиву данных, где каждый класс задается гауссовским распределением:

```
X, y = make_blobs(random_state=42)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.legend(["Класс 0", "Класс 1", "Класс 2"])
```



Задание11: Обучаем на этом наборе данных классификатор LinearSVC. Распечатать формы массивов коэффициентов linear_svm.coef_ и констант linear_svm.intercept_

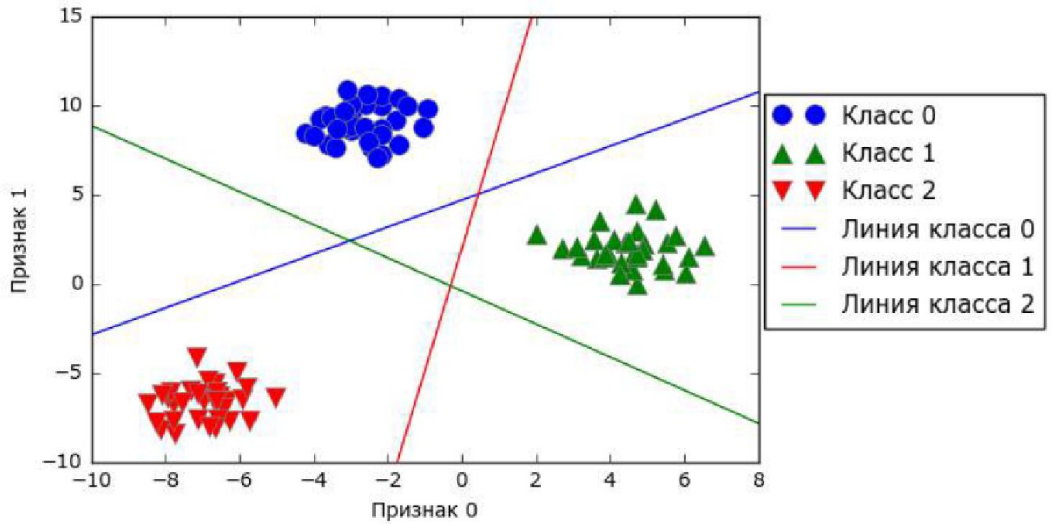
Атрибут coef_ имеет форму (3, 2), это означает, что каждая строка coef_ содержит вектор коэффициентов для каждого из трех классов, а каждый столбец содержит значение коэффициента для конкретного признака (в этом наборе данных их два).

Атрибут intercept_ является одномерным массивом, в котором записаны константы классов.

3.6.2 Множественная классификация

Визуализация границ принятия решений, полученных на трех бинарных классификаторах.

```
# исп. L1 регуляризац для оценки информативности признаков
# оценим при этом точность
for C, marker in zip([0.001, 1, 100], ['o', '^', 'v']):
lr_l1 = LogisticRegression(C=C, penalty="l1", solver='liblinear', max_iter=1000).fit(X_train, y_train)
plt.plot(lr_l1.coef_.T, marker, label="C={:.3f}".format(C))
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.hlines(0, 0, cancer.data.shape[1])
plt.xlabel("Индекс коэффициента")
plt.ylabel("Оценка коэффициента")
plt.ylim(-5, 5)
plt.legend(loc=3)
```



Все точки класса 0, находятся выше линии, соответствующей этом классу. Точки классов 1 и 2 находятся ниже линии ⇒ они классифицируются бинарным классификатором для класса 0 как «остальные».

Точки класса 0, находятся слева от линии класса 1 ⇒ бинарный классификатор для класса 1 классифицирует их как «остальные».

……..

Треугольник в середине графика: Все три бинарных классификатора относят точки, расположенные там, к «остальным».
Точке, расположенной в треугольнике будет присвоен класс, получивший наибольшее значение по формуле классификации –класс ближайшей линии.

3.6.3 Выводы по линейным моделям регрессии в задачах дискретной классификации

Основные параметры линейных моделей:

-параметр регуляризации («alpha» в моделях регрессии и «*C*» в LinearSVC и LogisticRegression).

-вид регуляризации (L_1 или L_2).

Достоинства:

-быстро обучаются, и быстро работают при классификации (прогнозе);
-легко интерпретировать и формализовать результаты;
-регуляризация L_1 позволяет отобрать признаки по важности и исключить из модели неинформативные признаки;
-масштабируются на большие наборы данных (часто используются на очень больших наборах данных только потому, что не представляется возможным обучить другие модели);
-хорошо работают с разреженными данными.

Недостатки:

-плохо работают на данных с высоко коррелированными признаками;
-в низкоразмерном пространстве могут иметь обобщающую способность ниже, чем другие модели.

Особенности:

-для данных из сотен тысяч или миллионов примеров использовать опцию solver='sag' в LogisticRegression и Ridge;
-хорошо работают, когда количество признаков превышает количество наблюдений.

3.6.3 Выводы по линейным моделям регрессии в задачах дискретной классификации

Рекомендации:

Цепочки функций (*method chaining*)

Method chaining – это конкатенация вызовов:

```
# в одной строке цепочка методов _init_, а затем fit
logreg = LogisticRegression().fit(X_train, y_train)

# связывание методов fit и predict в одной строке в scikit-learn
logreg = LogisticRegression()
y_pred = logreg.fit(X_train, y_train).predict(X_test)
y_pred = LogisticRegression().fit(X_train, y_train).predict(X_test)
```

Не всегда удобно:

- код трудночитаемый.
- обученная модель логистической регрессии не сохранена в какой-то определенной переменной, поэтому нельзя проверить ее, использовать, чтобы получить прогнозы для других данных, и т.п.

3.7 Наивные байесовские классификаторы

Наивные байесовские классификаторы рассматривают каждый признак отдельно:

рассчитываются статистические характеристики признаков

Виды наивных байесовских классификаторов в scikit-learn:

-GaussianNB – для обработки массивов непрерывных данных;

-BernoulliNB – работает с бинарными данными;

-MultinomialNB – счетные или дискретные данные (то есть каждый признак представляет собой подсчет целочисленных значений какой-то характеристики, например, речь может идти о частоте встречаемости слова в предложении).

3.7 Наивные байесовские классификаторы

Классификатор BernoulliNB подсчитывает ненулевые частоты признаков по каждому классу:

```
# Здесь 4  samples по 4 бинарных features
X = np.array([[0, 1, 0, 1], [1, 0, 1, 1], [0, 0, 0, 1], [1, 0, 1, 0]])
y = np.array([0, 1, 0, 1])                                  # категориальная переменная = метки 2х классов
                                                            # (классу 0 соответств. 1й и 3й пример из массива данных X, кл.1 -> 2,4

# подсчитываются частоты классов 1 и 0 по каждому признаку
counts = {}
for label in np.unique(y):                  # цикл по каждому классу
    counts[label] = X[y == label].sum(axis=0)

print("Массив данных X: \n" , X)
print("Частоты признаков:\n{}".format(counts))
print("T.e. - Да в Классе A (частота, кол-во раз по каждому признаку):\n{}".format(counts[0]))
print("Да в Классе B:\n{}".format(counts[1]))
```

```
Массив данных X:
 [[0 1 0 1]
 [1 0 1 1]
 [0 0 0 1]
 [1 0 1 0]]
Частоты признаков:
{0: array([0, 1, 0, 2]), 1: array([2, 0, 2, 1])}
T.e. - Да в Классе A (частота, кол-во раз по каждому признаку):
[0 1 0 2]
Да в Классе B:
[2 0 2 1]
```

3.7 Наивные байесовские классификаторы

MultinomialNB и GaussianNB считают иные статистические параметры каждого признака для каждого класса:

-MultinomialNB - среднее значение (м.о.);

-GaussianNB - среднее значение (м.о.), стандартное отклонение (СКО).

Классификатор сравнивает новый ввод со статистиками для каждого класса и прогнозирует наиболее подходящий класс.

Имеют один параметр alpha, который контролирует сложность модели:

-алгоритм добавляет к данным зависящее от alpha определенное количество искусственных наблюдений с положительными значениями для всех признаков. Это приводит к «сглаживанию» статистик:

-↑ alpha ⇒ высокая степень сглаживания, упрощение классификации;

-alpha незначительно влияет на работу классификатора, позволяет немного увеличивать правильность.

3.7 Наивные байесовские классификаторы

Наивные байесовские модели разделяют многие преимущества и недостатки линейных моделей.

Достоинства:

-быстро обучаются и прогнозируют;
-процесс обучения легко интерпретировать;
-хорошо работают с высокоразмерными разреженными данными;
-относительно устойчивы к изменениям параметров;
-часто используются на очень больших наборах данных, где обучение линейной модели может занять много времени.

Недостатки:

- более низкая обобщающая способность по сравнению с линейными классификаторами LogisticRegression и LinearSVC.

Особенности:

-GaussianNB в основном используется на с очень высокой размерностью;
-BernoulliNB и MultinomialNB используются для разреженных дискретных данных, например, при классификации текста;
-на наборах данных с относительно большим количеством признаков, имеющих ненулевые частоты (на больших документах), MultinomialNB работает лучше, чем BernoulliNB.

3.8 Деревья решений

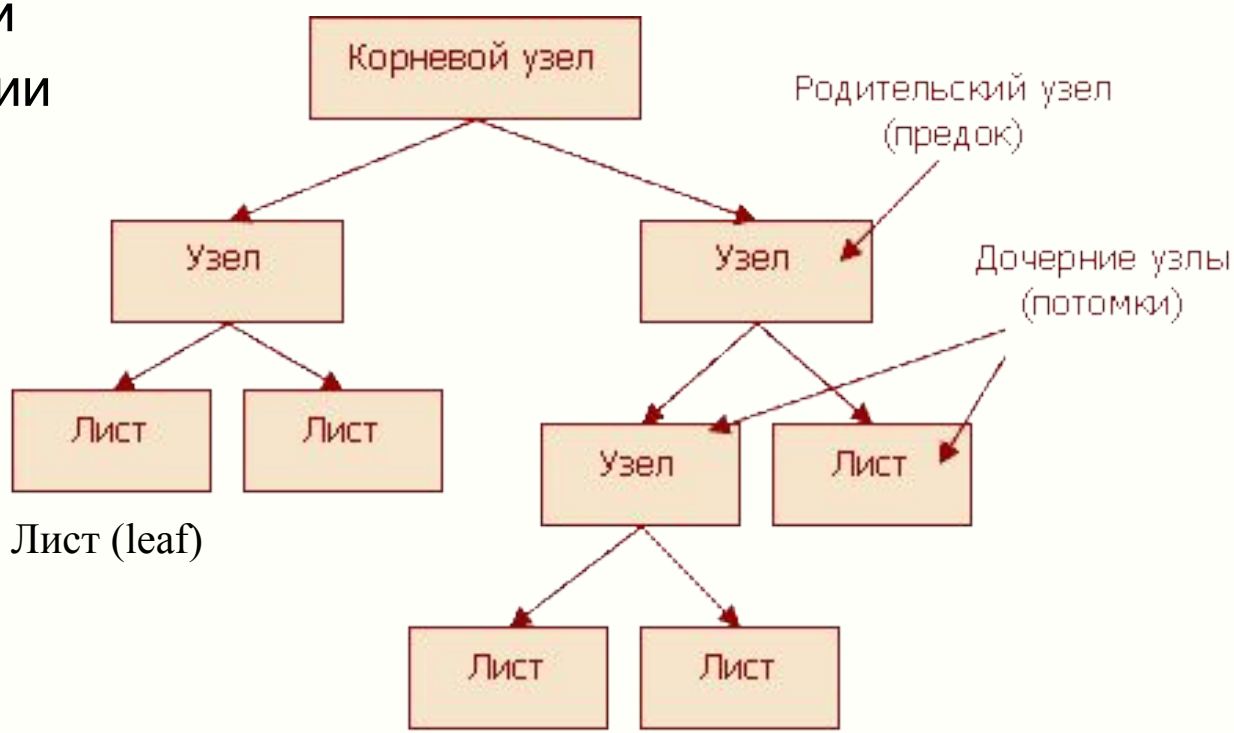
РЕШАЕМЫЕ ЗАДАЧИ

Задачи описания данных

Задачи классификации

Задачи регрессии

Т.Е. Деревья можно использовать для решения задач
классификации
-и дискретных,
-и непрерывных *(regression)*
величин



Лист (leaf)

Пример: методика построения

По сути - задаются вопросы и выстраивается иерархия правил «если… то», приводящая к решению

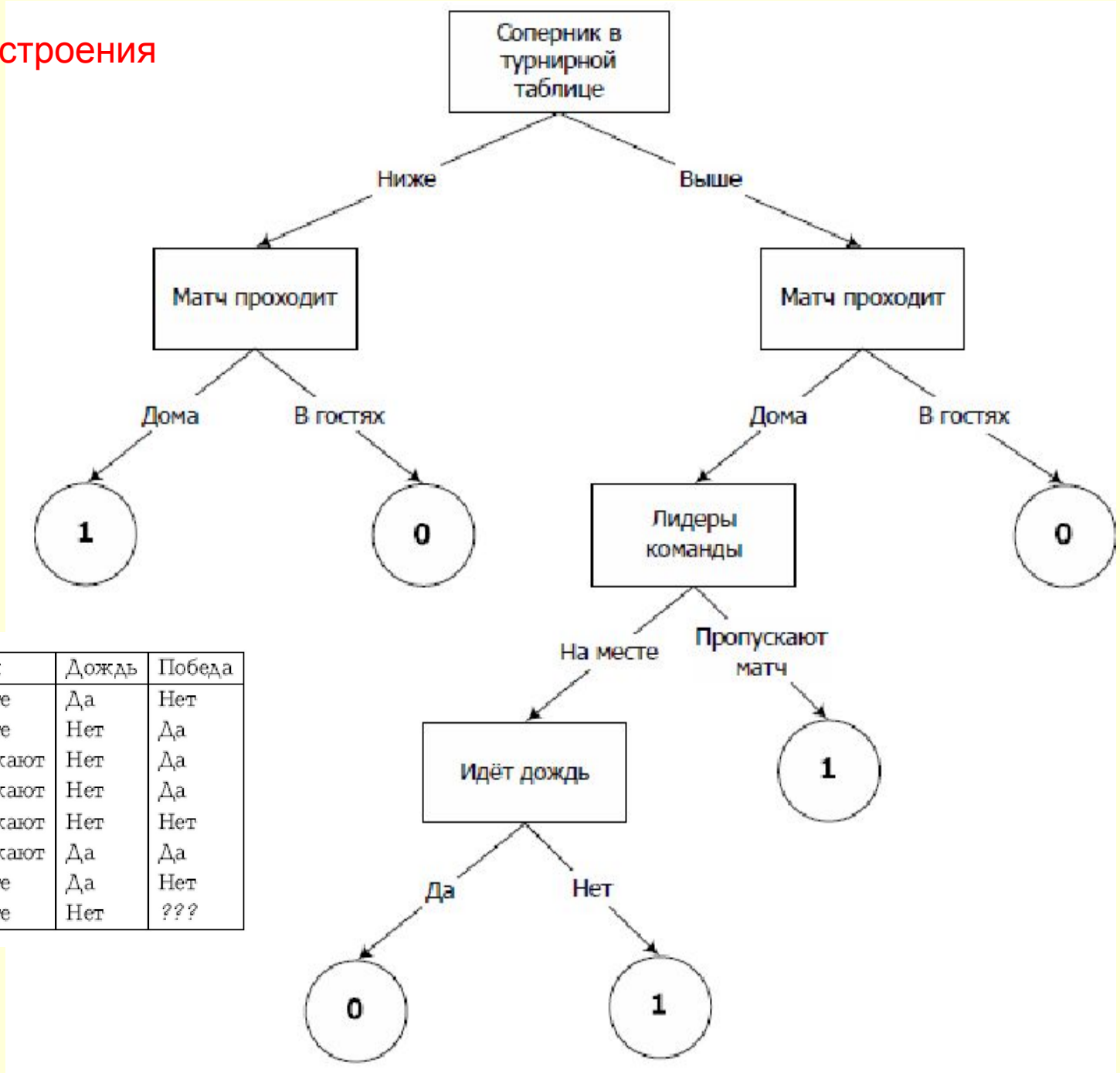
Дискретные признаки:

«да/нет»

Непрерывные:

«Признак *i* больше значения *a*?»

В машинном обучении эти правила называются тестами (tests).
Не путать с тестовым набором!



Соперник	Играем	Лидеры	Дождь	Победа
Выше	Дома	На месте	Да	Нет
Выше	Дома	На месте	Нет	Да
Выше	Дома	Пропускают	Нет	Да
Ниже	Дома	Пропускают	Нет	Да
Ниже	В гостях	Пропускают	Нет	Нет
Ниже	Дома	Пропускают	Да	Да
Выше	В гостях	На месте	Да	Нет
Ниже	В гостях	На месте	Нет	*???*

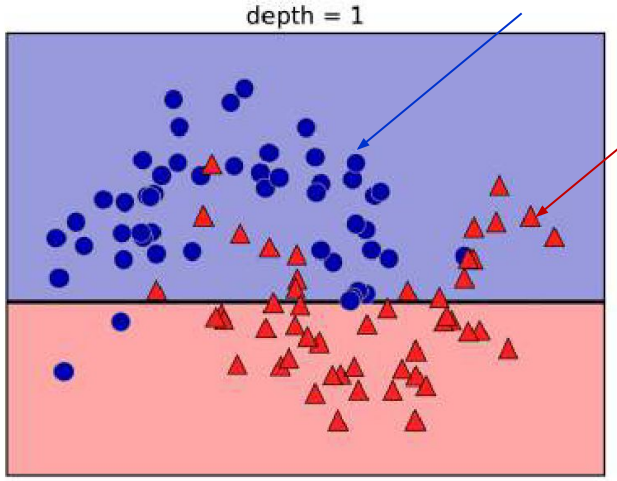
3.8.1 Деревья решений в задаче дискретной классификации

Первый прогон (тест). «тест» здесь ≠ тестовая выборка!

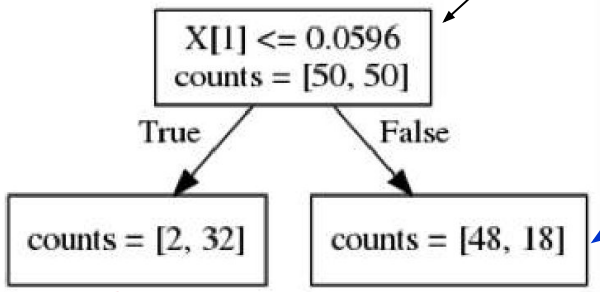
Разделение набора данных по горизонтали в точке x[1]=0.0596 дает наиболее полную информацию - лучше всего разделяются классы «0» и «1» по правилу

if x[i] <= 0.0596 then x[i]∈ “1”
else x[i]∈ “0”

В результате 1го прогона весь набор делится на классы с минимальными ошибками:



Модель уточняется за счет последующих прогонов



Верхний узел (корень, root), представляет собой весь набор обучающих данных: 50 точек класса «0», и 50 - класса «1»

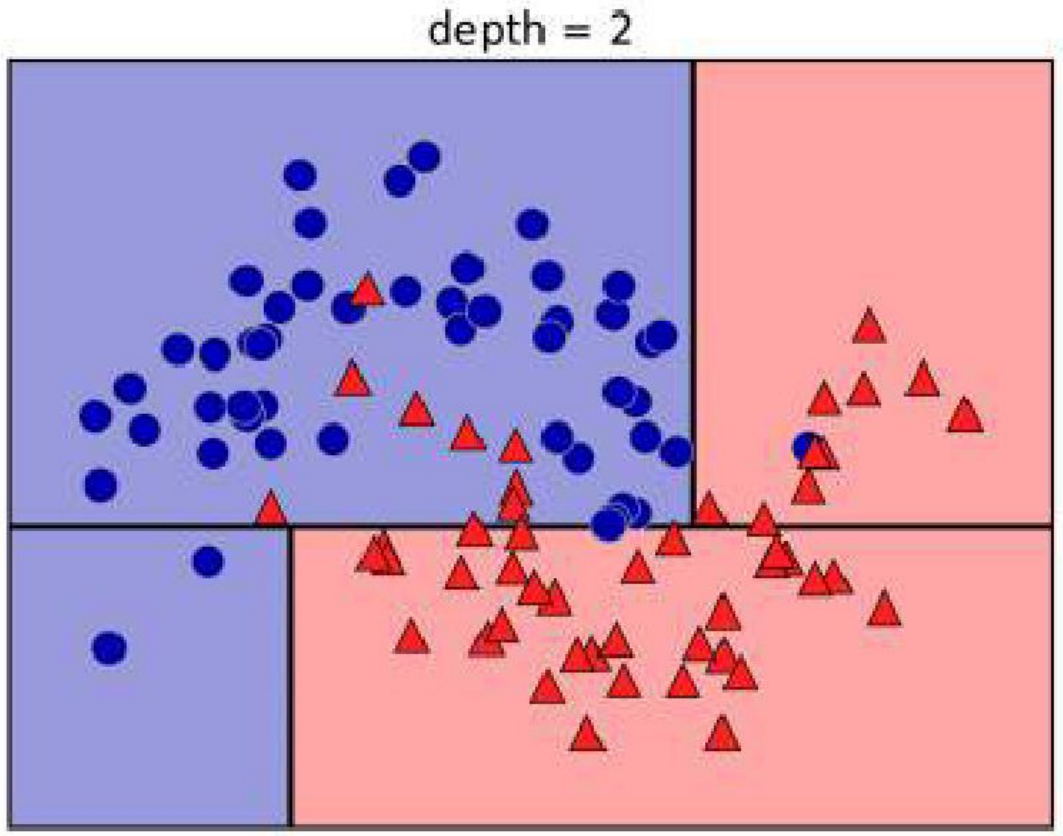
Правый узел {False = класс «0», верхняя часть диаграммы} содержит:
-48 точек, принадлежащих классу «0»,
-18 - классу «1».

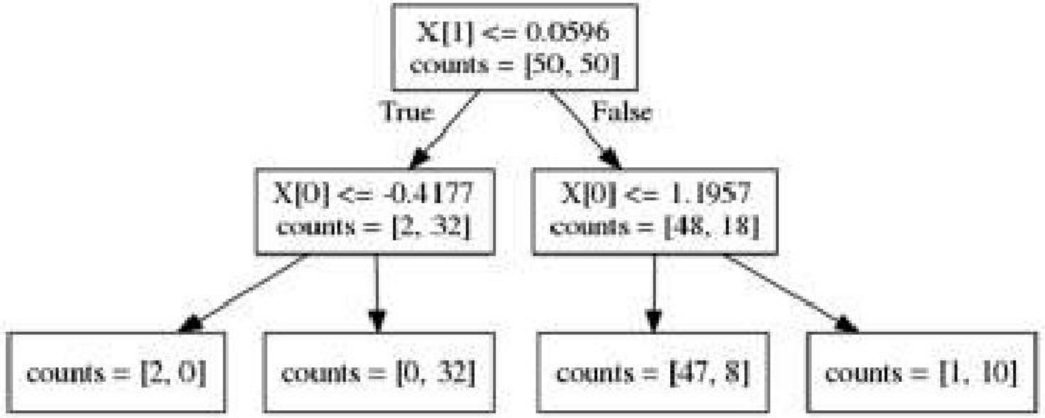
Левый узел {True = класс «1»} содержит:
-2 точки, принадлежащие классу «0»,
-32 точки - классу «1».

Прогон 2.

Разделение отдельно каждого узла по вертикали. Наиболее информативное разбиение на анализе x[0]:

Задание12: Записать условия разделений левого и правого узлов. Проанализировать точности





3.8.1 Деревья решений в задаче дискретной классификации

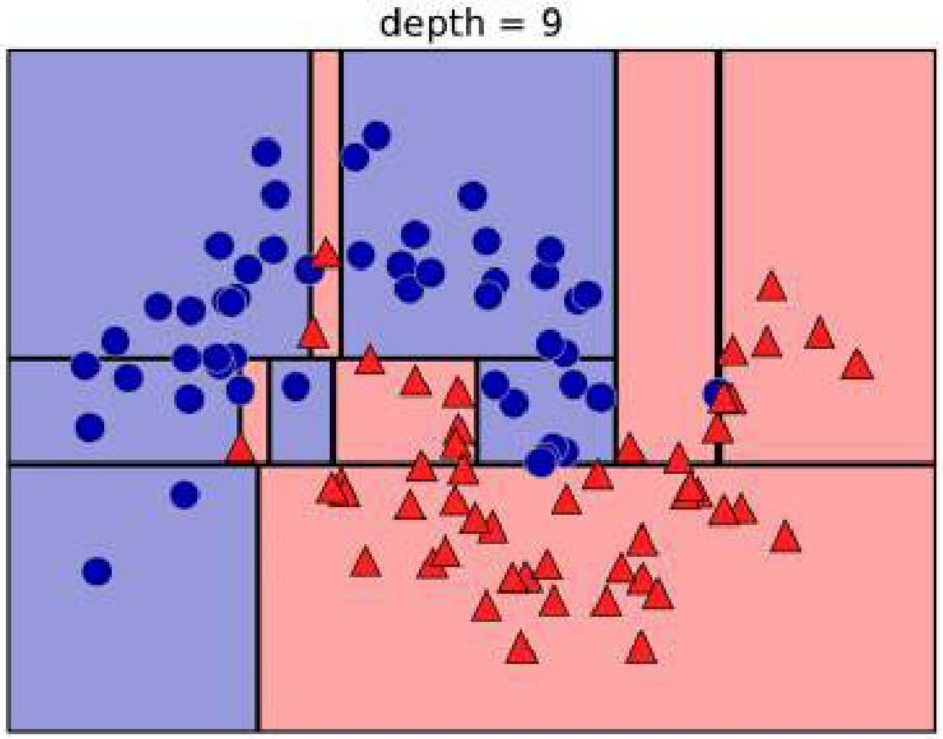
Рекурсивный процесс прогонов строит в итоге бинарное дерево решений.

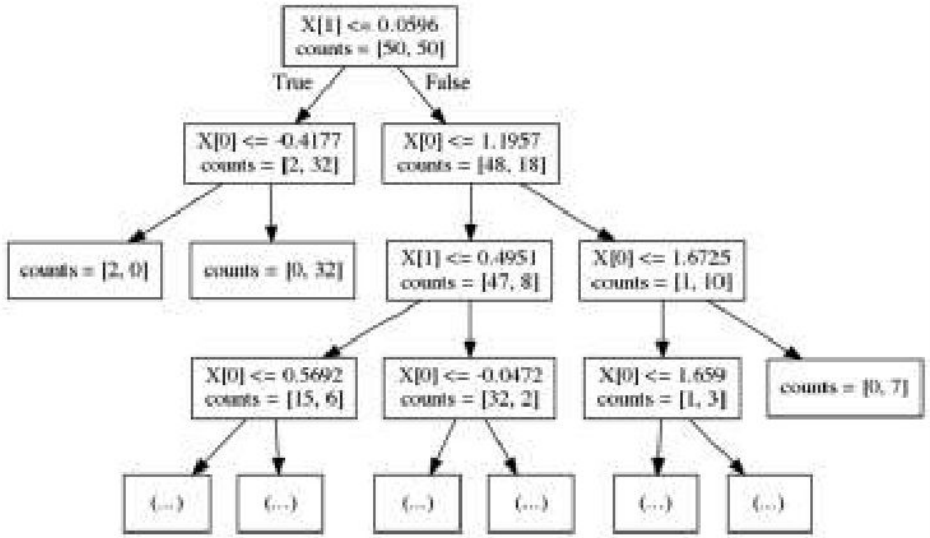
Поскольку каждый тест рассматривает только один признак ⇒ области, получающиеся в результате разбиения, всегда имеют границы, параллельные осям.

Рекурсивное разбиение данных повторяется до тех пор, пока все точки данных в каждой области разбиения (каждом листе дерева решений) не будут принадлежать одному и тому же значению целевой переменной (классу или количественному значению непрерывной переменной).

Лист дерева, который содержит точки данных, относящиеся к одному и тому же значению целевой переменной, называется чистым (pure).

Итоговое разбиение:





Прогноз для нового ввода:

-находится область разбиения пространства, к которому относится новая точка (область может быть найдена с помощью обхода дерева, начиная с корневого узла и путем перемещения влево или вправо, в зависимости от того, выполняется ли условие).;

-определяется класс (присваивается метка класс), к которому относится большинство точек в этой области (либо единственный класс в области, если лист является чистым).

Контроль сложности

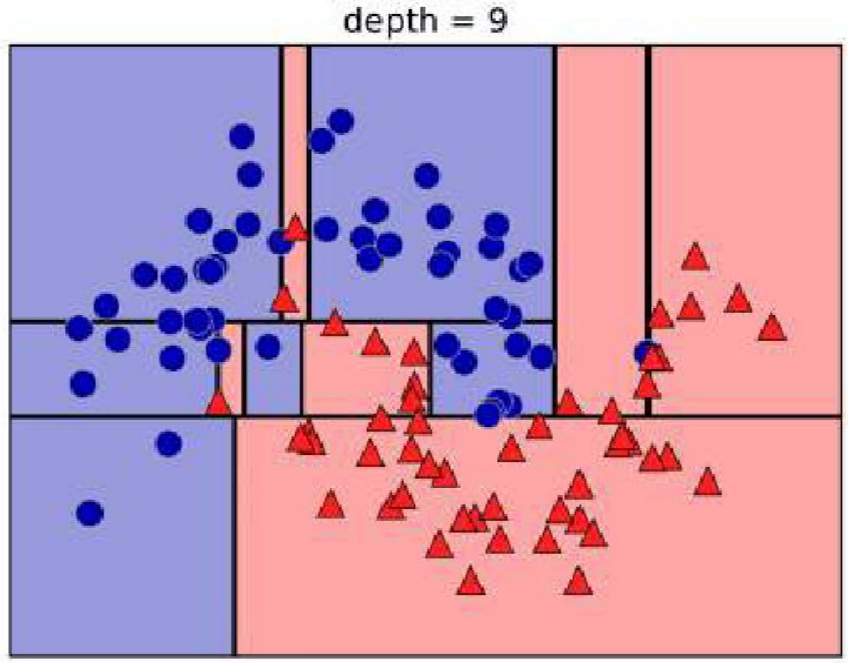
Построение дерева продолжается, как правило, до тех пор, пока все листья не станут чистыми:

наличие только чистых листьев означает, что дерево имеет 100%-ную правильность на обучающей выборке.

НО**!** Модели могут получаться сложными и характеризоваться сильным переобучением на обучающих данных.

3.8.1 Деревья решений в задаче дискретной классификации

Граница принятия решений фокусируется на отдельных точках-выбросах:



Стратегии от переобучения:

-ранняя остановка построения дерева (предварительная обрезка, pre-pruning).

-построение дерева с последующим удалением или сокращением малоинформативных узлов (пост-обрезка, post-pruning, или обрезка pruning).

Критерии предварительной обрезки:

-ограничение максимальной глубины дерева max_depth,

-ограничение максимального количества листьев max_leaf_nodes,

-минимальное количество наблюдений в узле min_samples_leaf, необходимое для разбиения.

В библиотеке scikit-learn деревья решений для дискретной классификации реализованы в классе DecisionTreeClassifier

Реализована лишь предварительная обрезка.

3.8.1 Деревья решений в задаче дискретной классификации

Пример предварительной обрезки для набора Breast Cancer.

Задание13: Загрузить набор данных Breast Cancer.
Разбить на обучающий и тестовый наборы. (random_state=42)
Используя модель DecisionTreeClassifier с параметрами (random_state=0) из модуля sklearn.tree создать и обучить классификатор.

Использовали настройки по умолчанию - дерево «выращивалось» до тех пор,
пока все листья не стали чистыми.

Оценить точность классификатора на обучающем и на тестовом наборах методом score.

Правильность на обучающем наборе: 1.000
Правильность на тестовом наборе: 0.937

Правильность на обучающем наборе составляет 100%

Правильность на тестовом наборе хуже, чем при использовании ранее рассмотренных линейных моделей (там = около 95%)

3.8.1 Деревья решений в задаче дискретной классификации

Продолжение Примера предварительной обрезки.

Необрезанные деревья склонны к переобучению и плохо обобщают результат на новые данные. Применить к дереву предварительную обрезку, которая остановит процесс построения дерева. Выбрать вариант останова

«по достижении определенной глубины : max_depth=4»

```
# оценим при этом точность# предварительная обрезка по глубине max_depth=4
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)
```

Снова оценить точность классификатора на обучающем и на тестовом наборах методом score.

Правильность на обучающем наборе: 0.988
Правильность на тестовом наборе: 0.951

Ограничение глубины дерева уменьшает переобучение.

Это приводит к более низкой правильности на обучающем наборе, но улучшает правильность на тестовом наборе

Визуализация дерева, анализ важности признаков

Для визуализации используется функция export_graphviz из модуля tree или функция Image оболочки Ipython

Чаще продуктивнее рассмотреть важность признаков (feature importance): оценивается важность признака для получения решений. Выдает число [0; 1], где:

-0 = «не используется вообще»,

-1 = «отлично предсказывает целевую переменную».

Важности признаков в сумме всегда дают 1.

```
print("Важности признаков:\n{}".format(tree.feature_importances_))
```

```
Важности признаков
[ 0.          0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.01019737  0.04839825  0.          0.
  0.0024156   0.          0.          0.          0.          0.
  0.72682851  0.0458159   0.          0.          0.0141577   0.          0.018188
  0.1221132   0.01188548  0.        ]
```

Визуализация дерева, анализ важности признаков

```
# или для удобства
for name, score in zip(cancer["feature_names"], tree.feature_importances_):
    print(name, score)
```

Если признак имеет низкое значение feature_importance_, это не значит, что он неинформативен.

Это означает только то, что данный признак не был выбран деревом, поскольку, вероятно, другой признак содержит ту же самую информацию.

В отличие от коэффициентов линейных моделей важности признаков всегда положительны и они не указывают на взаимосвязь с каким-то конкретным классом.

3.8.2 Деревья решений в задаче классификации *regression*

Применение деревьев регрессии ≈ применению деревьев классификации. Особенности:

-не умеет экстраполировать или делать прогнозы вне диапазона значений обучающих данных.

-значение целевой переменной усредняется по всем обучающим точкам в листе.

В библиотеке scikit-learn деревья решений для *regression* реализованы в классе DecisionTreeRegressor

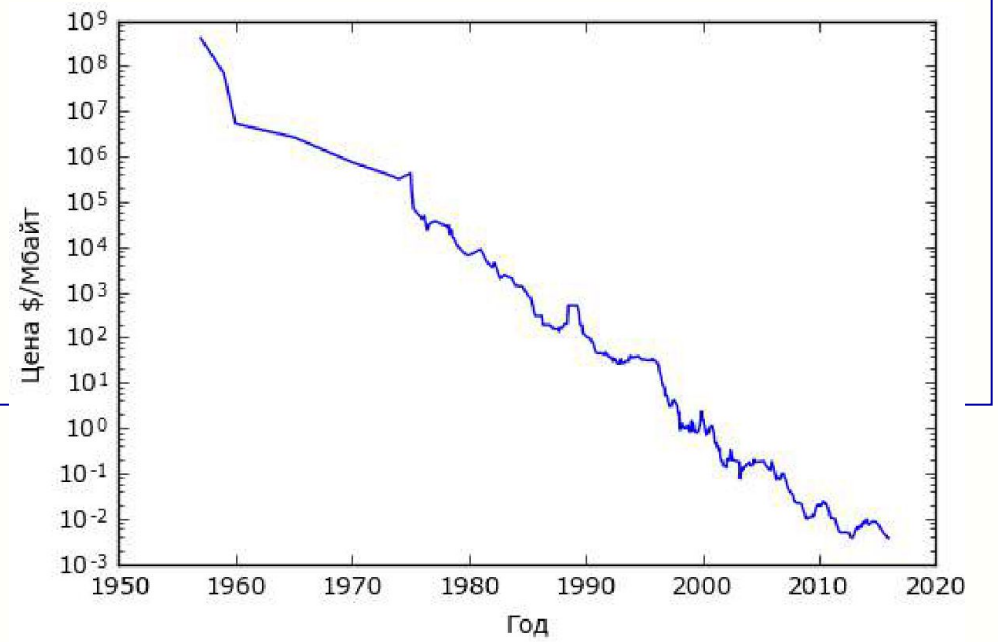
Реализована лишь предварительная обрезка.

Воспользоваться набором данных RAM Price (содержит исторические данные о ценах на компьютерную память):

-дата - по оси x,

-цена одного мегабайта ОЗУ в логарифмическом масштабе – по оси y:

```
import pandas as pd
ram_prices = pd.read_csv("C:/Users/User/2019-20/dannye/ram_price.csv")
plt.semilogy(ram_prices.date, ram_prices.price)
plt.xlabel("Год")
plt.ylabel("Цена $/Мбайт")
```



3.8.2 Деревья решений в задаче классификации *regression*

Пример: прогноз цен на период после 2000 года. Сравнение моделей DecisionTreeRegressor и LinearRegression.
Для количественной оценки мы будем рассматривать только тестовый набор.

```
# используем исторические данные для прогнозирования цен после 2000 года RAM цен
data_train = ram_prices[ram_prices.date < 2000]
data_test = ram_prices[ram_prices.date >= 2000]

# прогнозируем цены по датам
X_train = data_train.date[:, np.newaxis]

# используем логпреобразование, что получить простую взаимосвязь между данными и откликом
y_train = np.log(data_train.price)

from sklearn.tree import DecisionTreeRegressor
tree = DecisionTreeRegressor().fit(X_train, y_train)      # Деревья

from sklearn.linear_model import LinearRegression
linear_reg = LinearRegression().fit(X_train, y_train)     # Линейная регрессия

# прогнозируем по всем данным
X_all = ram_prices.date[:, np.newaxis]
pred_tree = tree.predict(X_all)
pred_lr = linear_reg.predict(X_all)

# экспоненцируем, чтобы обратить логарифмическое преобразование
price_tree = np.exp(pred_tree)
price_lr = np.exp(pred_lr)
```

3.8.2 Деревья решений в задаче классификации *regression*

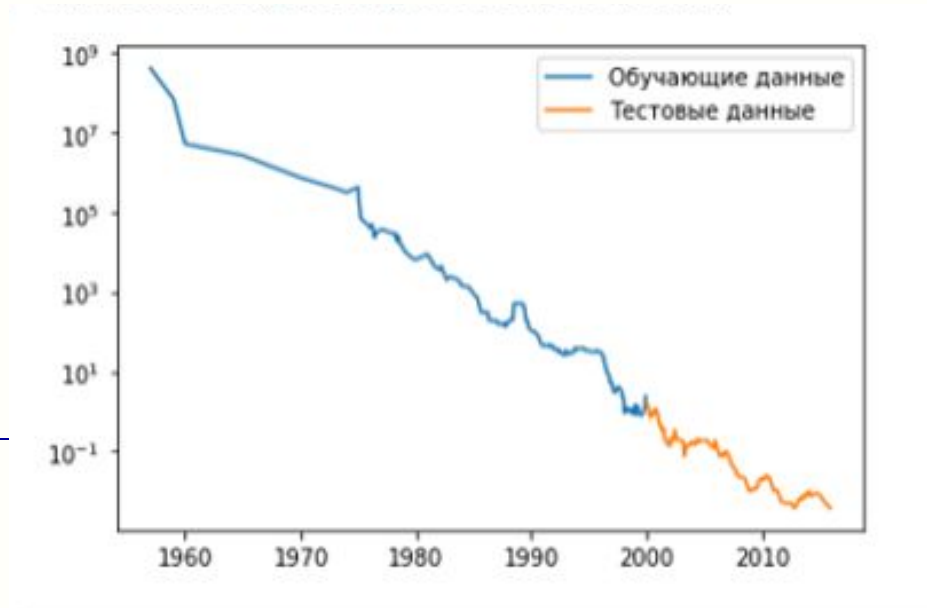
Пример: прогноз цен на период после 2000 года. Сравнение моделей DecisionTreeRegressor и LinearRegression.
Для количественной оценки мы будем рассматривать только тестовый набор.

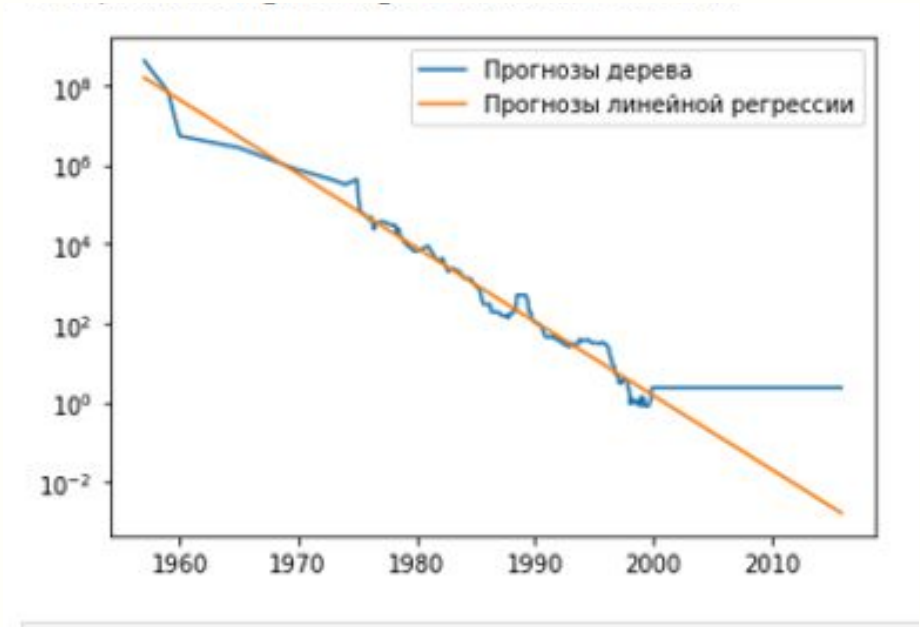
```
# используем исторические данные для прогнозирования цен после 2000 года RAM цен
data_train = ram_prices[ram_prices.date < 2000]
data_test = ram_prices[ram_prices.date >= 2000]

# прогнозируем цены по датам
X_train = data_train.date[:, np.newaxis]

# используем логпреобразование, что получить взаимосвязь между данными и откликом
y_train = np.log(data_train.price)

from sklearn.tree import DecisionTreeRegressor
tree = DecisionTreeRegressor().fit(X_train, y_train)      # Деревья

from sklearn.linear_model import LinearRegression
linear_reg = LinearRegression().fit(X_train, y_train)     # Линейная регрессия

# прогнозируем по всем данным
X_all = ram_prices.date[:, np.newaxis]
pred_tree = tree.predict(X_all)
pred_lr = linear_reg.predict(X_all)

# экспоненцируем, чтобы обратить логарифмическое преобразование
price_tree = np.exp(pred_tree)
price_lr = np.exp(pred_lr)
```

3.8.2 Деревья решений в задаче классификации *regression*

Сравнение прогнозов с реальными данными

```
data_train = ram_prices[ram_prices.date < 2000]
plt.semilogy(data_train.date, data_train.price, label="Обучающие данные")
plt.semilogy(data_test.date, data_test.price, label="Тестовые данные")
plt.legend()

plt.semilogy(ram_prices.date, price_tree, label="Прогнозы дерева")
plt.semilogy(ram_prices.date, price_lr, label="Прогнозы линейной регрессии")
plt.legend()
```





Линейная модель дает хороший прогноз для тестовых данных после 2000 года, сглаживая всплески в обучающих и тестовых данных

Дерево хорошо работает на обучающих данных, но неработоспособно вне диапазона обучающих значений

3.8.3 Выводы по Деревьям решений

Достоинства:

- модель может быть легко визуализирована и понята неспециалистам;
- деревья не требуют масштабирования данных: нормализации, стандартизация признаков.

Недостатки:

- даже при использовании предварительной обрезки, они склонны к переобучению;
- имеют низкую обобщающую способность.

Особенности:

- параметры, задающие сложность модели в деревьях решений – параметры предварительной обрезки max_depth, max_leaf_nodes, min_samples_leaf;
- хорошо работают, когда признаки имеют разную размерность, относятся к разным физическим величинам, представляют смесь бинарных и непрерывных признаков;
- вследствие недостатков в большинстве случаев вместо одиночного дерева решений используются ансамбли деревьев.