Лекция 9

Классификация

ROC-кривая

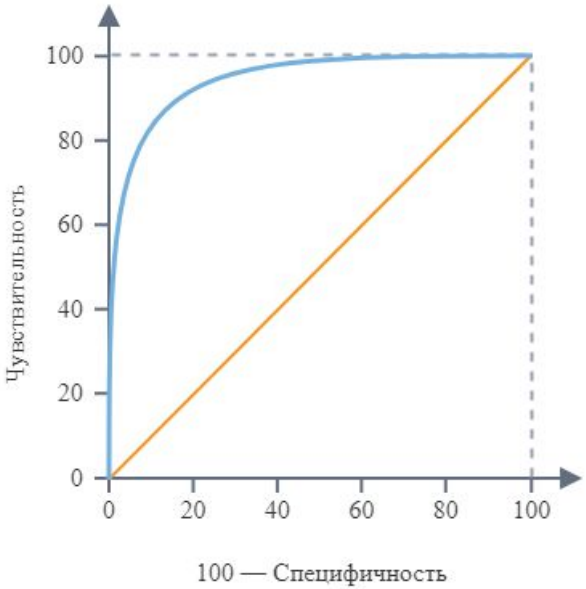
- ROC - receiver operating characteristic/кривая ошибки
- Визуальный способ оценить бинарную классификацию
- Показывает отношение между чувствительностью и специфичностью:
  - Чувствительность - доля истинных ответов:

$$S_e = TPR = \frac{TP}{TP + FN} * 100\%$$

  - Специфичность - доля правильно определенных отрицательных ответов:

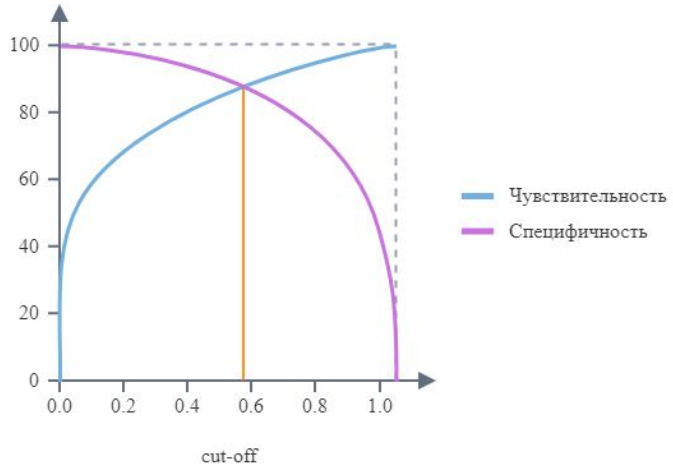
$$S_p = \frac{TN}{TN + FP} * 100\%$$

$$FPR = 100 - S_p$$



Чувствительность и специфичность

- Чувствительный классификатор - вероятность получить истинный правильный ответ
- Специфичный классификатор - вероятность получить ложный правильный ответ
- Меняя порог принятия решения можно получить разные оценки
- Для построения ROC-кривой, необходимо считать значения TPR и FPR при различных значения порога (**$O(N^2)$**)
- Для выбора порога можно:
  - Поиск минимально необходимого значения чувствительности/специфичности
  - $\max(S_e + S_p)$
  - $\min(|S_e - S_p|)$



Алгоритм построения O(N)

1. Провести сортировку по убыванию значения приятия истинного класса
2. Единичный квадрат разбить на m (кол-во 1) горизонтальных и n (кол-во 0) вертикальных линий. Получается квадрат разбитый на m*n блоков
3. Начиная с точки (0,0) постройте линию по блокам:
    a. Если предсказание равно 1 - смещение на блок вверх
    b. Если предсказание равно 0 - смещение на блок вправо

Гарантируется, что попадем в точку (1,1)

Если значения предсказаний равны, то необходимо сдвинуться на a (кол-во меток 1) вверх и b вправо (кол-во меток 0) блоков.

Алгоритм построения O(N) - пример

id	оценка	класс
1	0.5	0
2	0.1	0
3	0.2	0
4	0.6	1
5	0.2	1
6	0.3	1
7	0.0	0

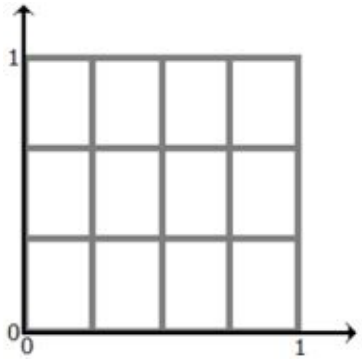
Табл. 1

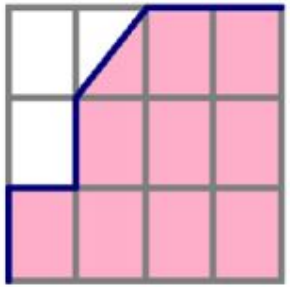
id	оценка	класс
4	0.6	1
1	0.5	0
6	0.3	1
3	0.2	0
5	0.2	1
2	0.1	0
7	0.0	0

Табл. 2

id	> 0.25	класс
4	1	1
1	1	0
6	1	1
3	0	0
5	0	1
2	0	0
7	0	0

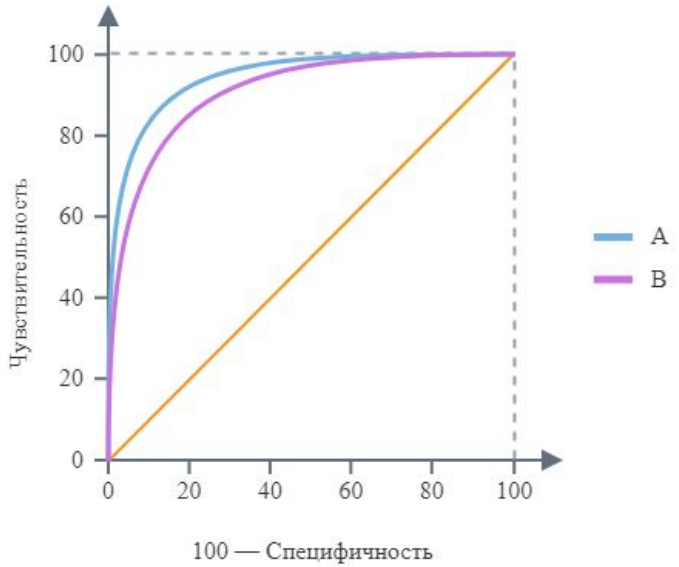
Табл. 3





AUC

- ROC-кривые позволяют сравнить 2 классификатора. Лучше тот, у которого крива находится левее и выше.
- Такая характеристика не всегда показательная, поэтому считают AUC - area under the curve
- AUC показывает площадь под кривой, и может быть рассчитана методом трапеций
- Значения AUC говорят о качестве модели:
  - 0.5-0.6 - плохой классификатор
  - 0.6-0.8 - удовлетворительный классификатор
  - 0.8-1.0 - отличный классификатор

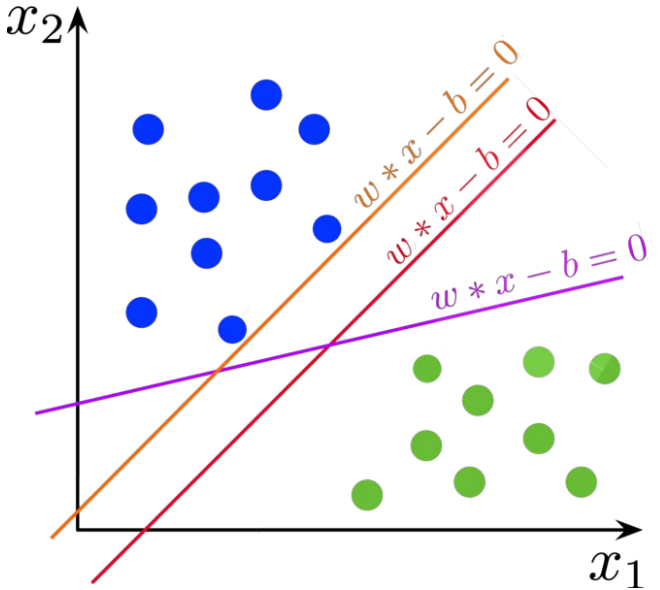


Метод опорных векторов (SVM)

- SVM - support vector machine. Метод основывающийся на том, что между классами можно провести разделяющую гиперплоскость, если классы линейно разделимы.
- Обучение сводится к тому, чтобы найти такую разделяющую плоскость:

$$\begin{cases} \langle x, w \rangle - b > 0, \forall x \in C_1 \\ \langle x, w \rangle - b < 0, \forall x \in C_2 \end{cases}$$

- Таких гиперплоскостей может быть сколь угодно много. Поэтому нужно найти такую гиперплоскость, которая равноудалена от об обоих классов.



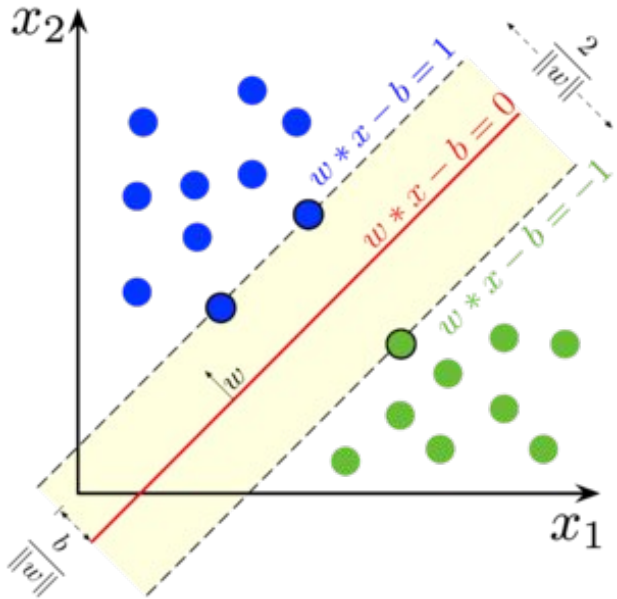
Полностью разделимый случай - жесткий зазор (маржа)

- Предположим, что метки класса имеют значения {1, -1}
- Можно рассчитать значение отступа для каждой точки так, что отступ для каждой точки положителен

$$M_i(w, b) = y_i(\langle x_i, w\rangle - b) > 0, \forall i$$

- Нормирование параметров не меняет разделяющую плоскость. И сделать так, чтобы минимально возможный отступ был равен 1
- Поэтому в каждом классе можно найти такие граничные точки

$$\begin{cases} M_+(w, b) = (+1)(\langle x_+, w\rangle - b) = 1 \\ M_-(w, b) = (-1)(\langle x_-, w\rangle - b) = 1 \end{cases}$$



Обучение SVM

- Так как выборка линейно разделима, то в полосе $-1 < \langle x, w \rangle - b < 1$ не может лежать ни одной точки. Ширина полосы равен $x_+ - x_-$
- Задача сводится к нахождению полосы максимальной ширины

$$\frac{\langle x_+ - x_-, w \rangle}{\|w\|} = \frac{\langle x_+, w \rangle - \langle x_-, w \rangle - b + b}{\|w\|} =$$
$$\frac{M_+ + M_-}{\|w\|} = \frac{2}{\|w\|} \to max \Rightarrow \|w\| \to min$$

- Задача сводится к оптимизации в квадратичном программировании

$$\begin{cases} \|w\|^2 \to \min \\ M_i(w, b) \geq 1, \forall i \end{cases}$$

Не линейная разделимость - мягкий зазор

- Не всегда можно найти границу, четко разделяющую классы. Поэтому от для каждой точки от отступа можно отнять положительную величину ε_i, и будем требование, что такие отступы должны быть минимальны
- Тогда получим задачу оптимизации с мягким зазором

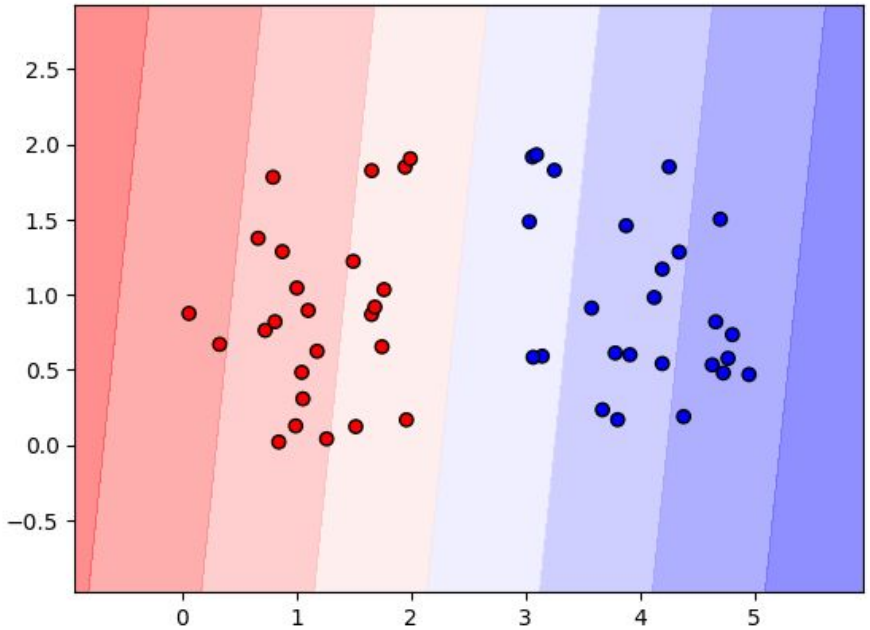
$$\begin{cases} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\varepsilon_i \to \min \\ M_i(w,b) \geq 1 - \varepsilon_i, \forall i \\ \varepsilon_i \geq 0, \forall i \end{cases}$$

- Коэффициент C определяет силу влияния допущений для отступов

Линейный SVM в SKLearn

В случае множественной классификации используется OvO

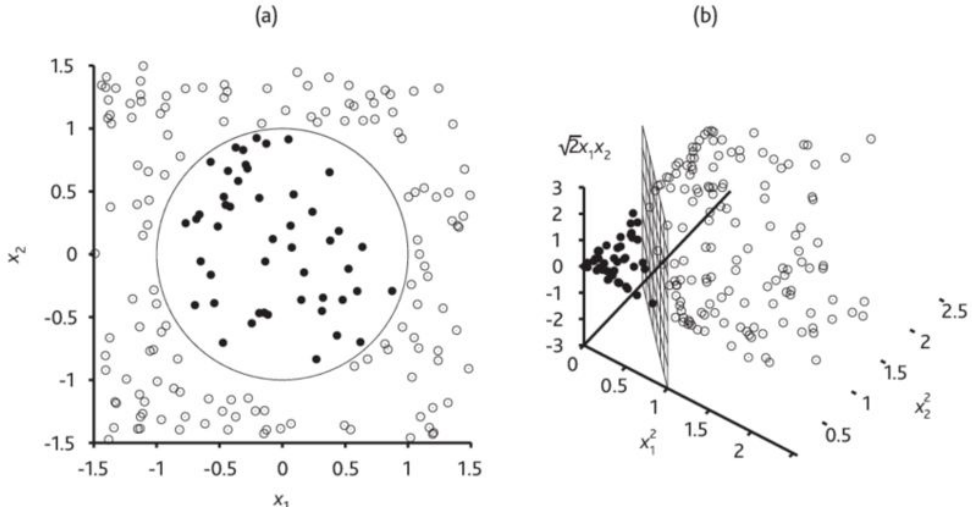
```
from sklearn.svm import LinearSVC
svc1 = LinearSVC()
svc1.fit(X,Y)
Yp = svc1.predict(X)
```



Повышение размерности

Предположения

- Если в пространстве X классы линейно неразделимы, то можно сделать преобразование в пространство больше размерности в котором классы становят разделимы.
- Многим методам нужна только информация о схожести объектов, поэтому можно использовать матрицу схожести (скал. произведение)
- Например, переход от признаков $[x_1,x_2]$ к признакам $[x_1^2,x_2^2,sqrt(2)*x_1*x_2]$



Ядерный трюк (kernel trick)

- Есть преобразование нелинейное преобразование $\varphi(x) : X \to H$
- Если в H задано скалярное произведение, то $\langle \varphi(x), \varphi(x') \rangle = K(x, x')$ называют ядром
- Ядро, определяет то, как считается скалярное произведение в пространстве H без самого преобразования признаков в пространство H
- Ядра обеспечивают линейную вычислительную эффективность в нелинейных случаях, а также преимущества линейных методов.
- Не каждая функция φ может быть использована в ядре и требует соблюдения определенных свойств

Виды ядер

- Линейное (linear) - $K(x, x') = \langle x, x' \rangle$
- Полиномиальное (poly) - $K(x, x') = (\gamma \langle x, x' \rangle + c_0)^d$, где d степень полинома. Позволяет строить кривые линии в линейном пространстве

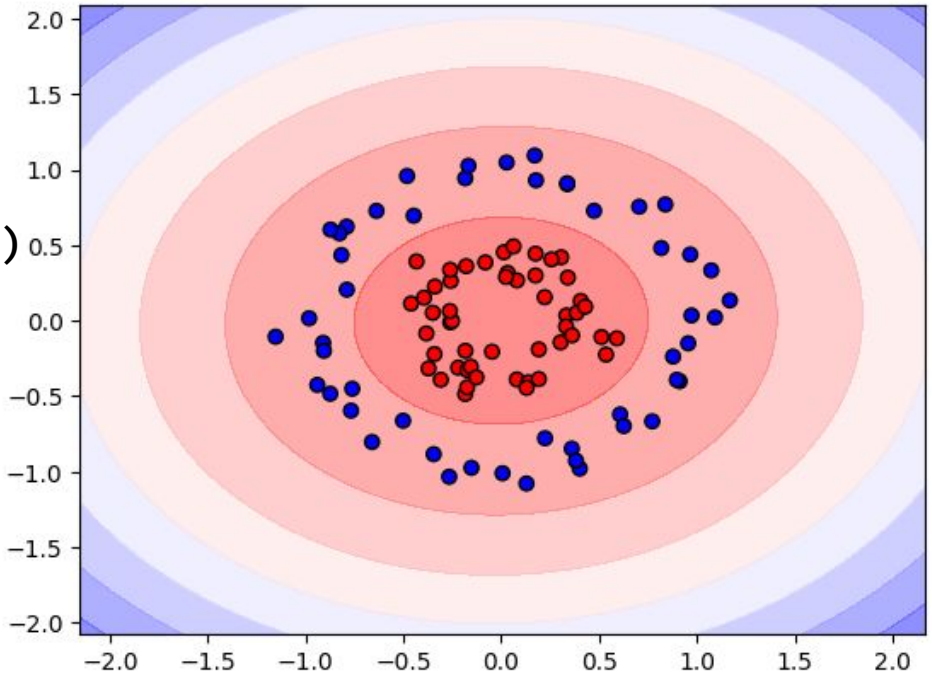
$$K(x, y) = (\langle x, y \rangle + 1)^2 = (1 + x_1 y_1 + x_2 y_2)^2 =$$
$$1 + x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 y_1 x_2 y_2 =$$
$$\langle \varphi(x), \varphi(y) \rangle \Rightarrow \varphi(x) = (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2)$$

- Гаусово (rbf) - $K(x, x') = \exp(-\gamma \|x - x'\|^2) = \exp(-\frac{\|x - x'\|^2}{2\sigma^2})$ позволяет оценивать близость точек на основе соответствия нормально распределению
- Сигмоидальное (sigmoid) - $K(x, x') = \tanh(\gamma \langle x, x' \rangle + c_o)$ не удовлетворяет всем условиям ядра, но на практике работает хорошо

Нелинейный SVM в SKLearn

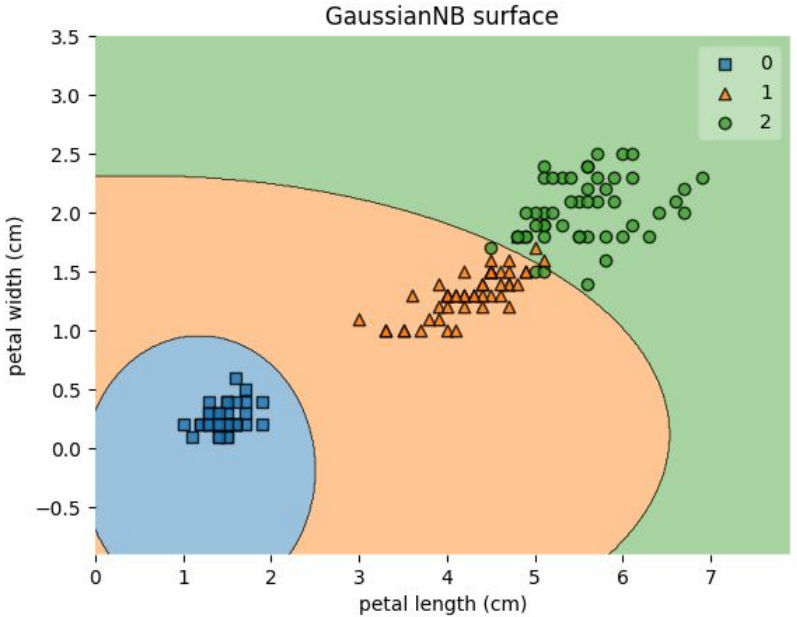
По умолчанию ядро rbf

```
from sklearn.svm import SVC
svcn = SVC(kernel = 'poly', degree = 2)
svcn.fit(Xn,Yn)
Ynp = svcn.predict(Xn)
```



Наивный Байесовский классификатор

- Наивный Байесовский классификатор - вероятностный классификатор, который предполагает, что признаки между собой независимые.
- Классификатор строит функцию распределения для каждого класса. Для построения используется EM-алгоритм.
- Наблюдения относится к тому классу, для которого вероятность принадлежности наибольшая.
- В основе классификатора лежит условная модель (теорема Байеса)



Модель Байесовского классификатора

- Вероятностная условная модель:

$$P(y|x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)}$$

- Знаменатель - константа. Числитель - совместная вероятность:

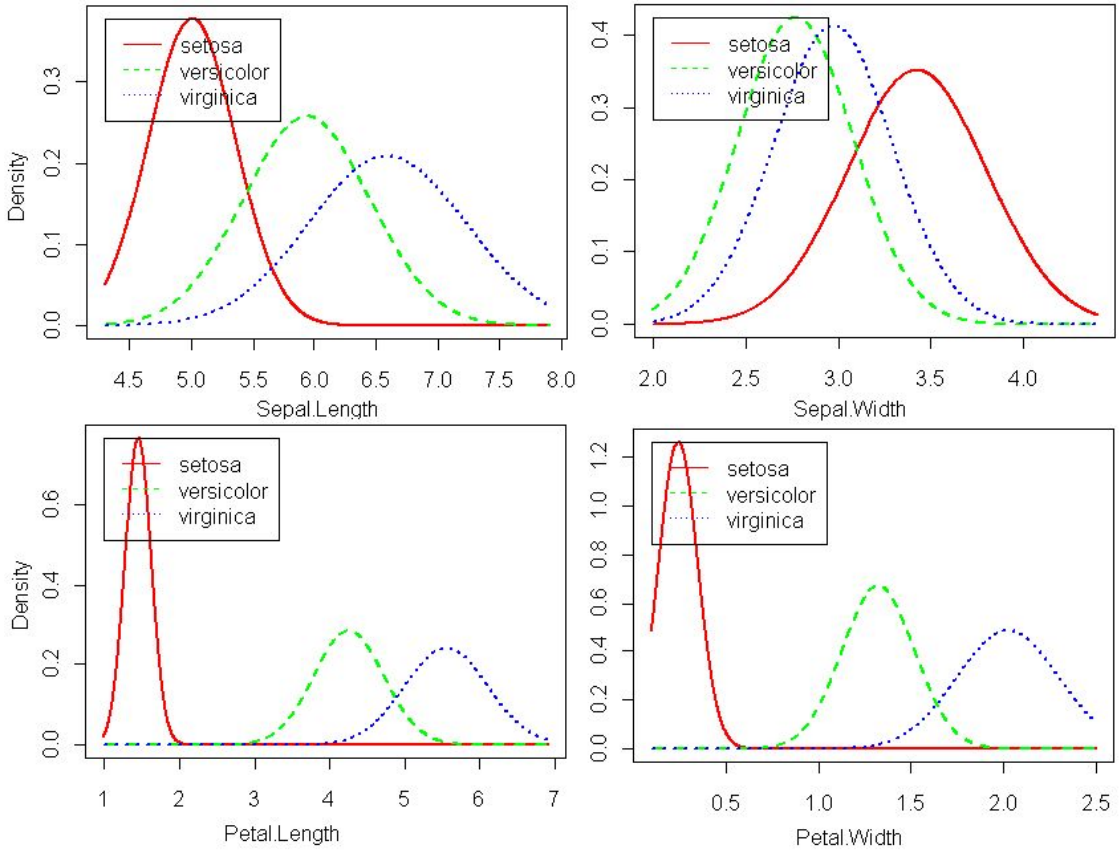
$$P(y, x_1, ..., x_n) = P(y)P(x_1, ..., x_n|y) =$$
$$P(y)P(x_1|y)P(x_2, ..., x_n|y, x_1) = ... =$$
$$P(y)(x_1|y)P(x_2|y, x_1)...P(x_n|y, x_1, x_2, ..., x_{n-1})$$

- Так как признаки независимы, то $P(x_i|y, x_j) = P(x_i|y)$
- Таким образом совместная вероятность (числитель) равна $P(y)\prod_{i=1}^{m} P(x_i|y)$
- Итоговый класс определяется как $\widehat{y} = \underset{\mathbf{y}}{\mathbf{argmax}} P(y)\prod_{i=1}^{m} P(x_i|y)$

Виды Байесовских классификаторов в SKLearn

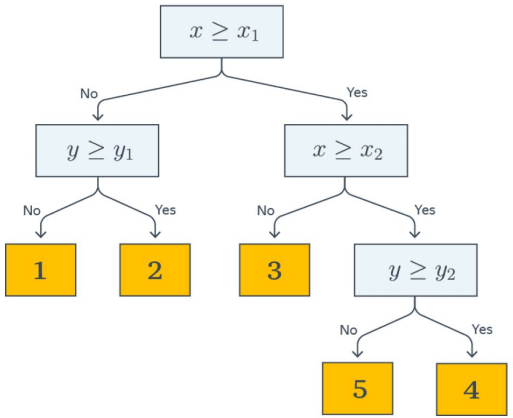
- Все находятся в подмодуле sklearn.naive_bayes
- GaussianNB - для подсчета условной вероятности используется функция Гаусса. Предполагается, что признаки распределены по нормальному закону.
- MultinomialNB - для мультиномиально распределенных данных
- ComplementNB - для мультиномиально распределенных данных с несбалансированными классами
- BernoulliNB - для данных распределенных по закону Бернулли
- CategoricalNB - для категориальных данных

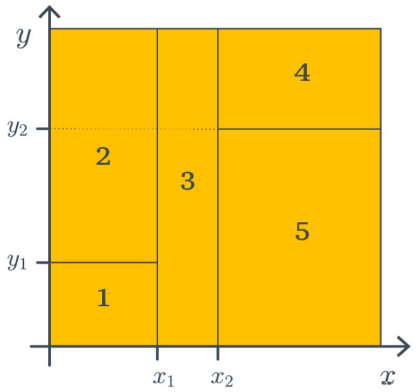
GaussianNB - вероятность по признакам



Решающие деревья

- Решающие деревья - семейство моделей машинного обучения, которые подходят для решения задач классификации и регрессии
- Предсказывает целевое значение через последовательность решающих правил. Хоть и не обладают хорошей обобщающей способностью, но часто используются в ансамблевых моделях (случайный лес)
- Легко интерпретируются





Построение деревьев

- Построить оптимальное дерево является вычислительно трудной задачей.
- Используется жадный алгоритм, где ищется наилучшее разделение пространства, чтобы минимизировать загрязнение. Существует огромное количество алгоритмов для построения дерева: ID3, C4.5, C5.0, CHAID, CN2, CART (в SKLearn)
- Регуляризация деревьев:
    - Максимальная глубина
    - Минимальное/максимальное кол-во наблюдений в листе
    - Минимальное кол-во наблюдений в узле для разбиения
    - Достигнут удовлетворяющий уровень загрязнения

Загрязнение узла

- Так как в каждый узел могут попадать значения разных классов, то необходимо оценивать это уровень загрязнения (степень качества разделения)
- Теоретико-информационный подход. Энтропия:

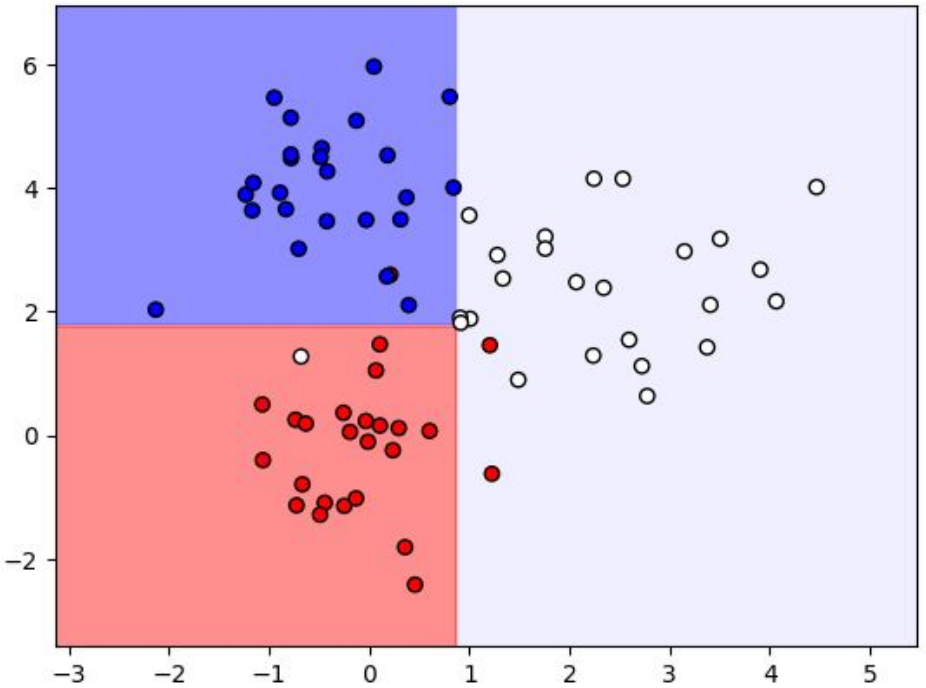
$$H(Q) = -\sum_{i=1}^{K} \frac{N_i}{N} \mathbf{log} \left(\frac{N_i}{N} \right)$$

- Статистический подход. Индекс Джини:

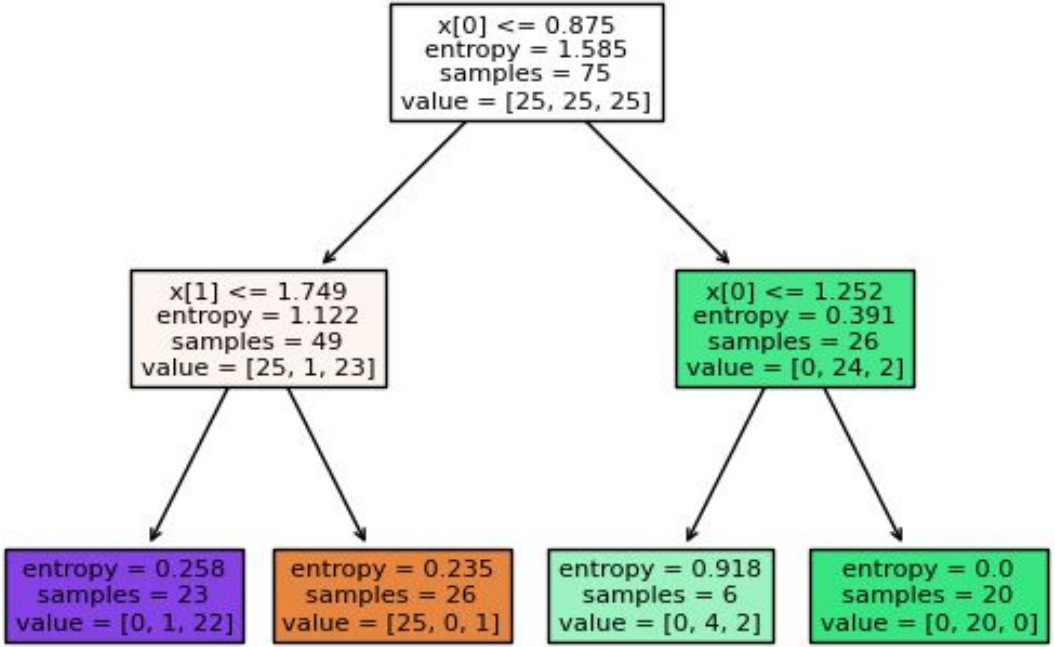
$$Gini(Q) = 1 - \sum_{i=1}^{K} \left(\frac{N_i}{N} \right)^2$$

Решающее классифицирующее дерево в SKLearn

```
from sklearn.tree import
DecisionTreeClassifier, plot_tree
dtc = DecisionTreeClassifier(
    criterion = 'entropy',
    max_depth = 2)
dtc.fit(X2,Y2)
Y2_p = dtc.predict(X2)
plot_tree(dtc, filled = True)
```



Визуализация дерева

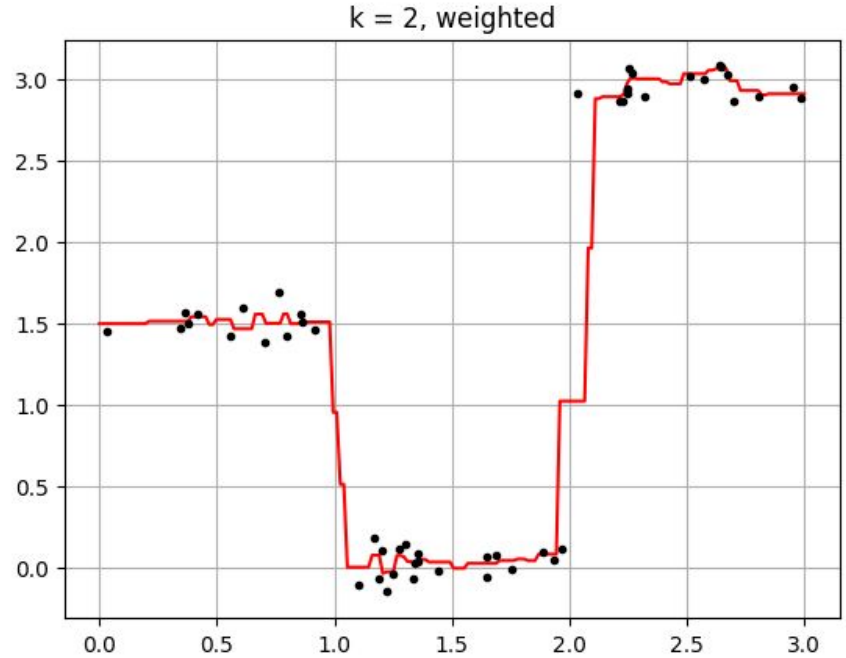


Классификаторы в регрессии

kNN регрессор

Предсказание для нового значения происходит как среднее [взвешенное значение] ближайших соседей

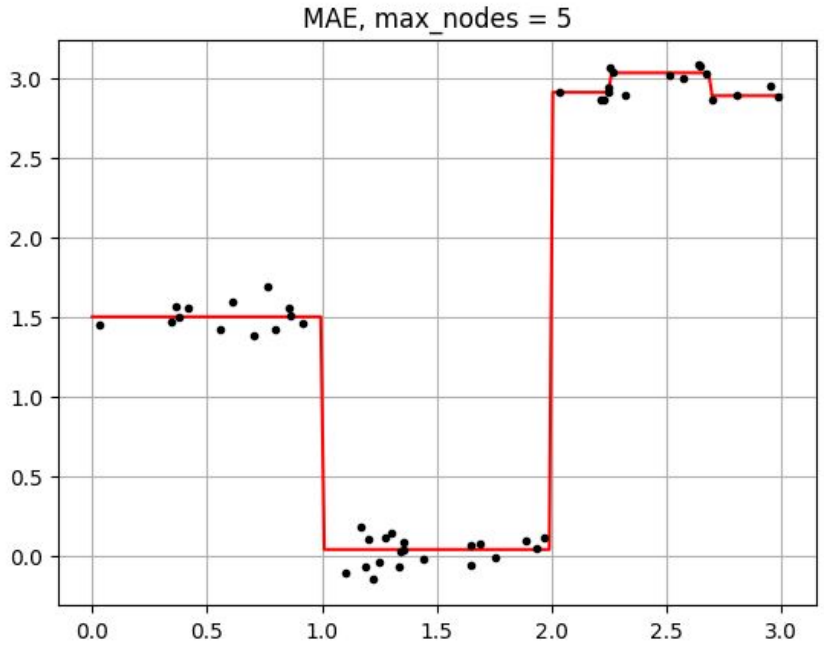
```
from sklearn.neighbors import
    KNeighborsRegressor
knr1 = KNeighborsRegressor(3,
    weights='uniform')
knr1.fit(Xr,Yr)
```



Регрессор на решающем дереве

Для каждой области предсказывает одно и то же значение. Хорошо аппроксимирует ступенчатые функции.

```
from sklearn.tree import
    DecisionTreeRegressor
dtr1 = DecisionTreeRegressor(
    criterion = 'absolute_error',
    max_leaf_nodes = 5)
dtr1.fit(Xr,Yr)
```



Метод опорных векторов для регрессии

Применяется редко. Особенность в том, что оценивается штраф - для значений попавших в полосу отступа штраф = 0.

```
from sklearn.svm import SVR
svm1 = SVR(kernel = 'rbf')
svm1.fit(Xr,Yr.ravel())
```