**ПРАКТИЧЕСКАЯ РАБОТА №7**

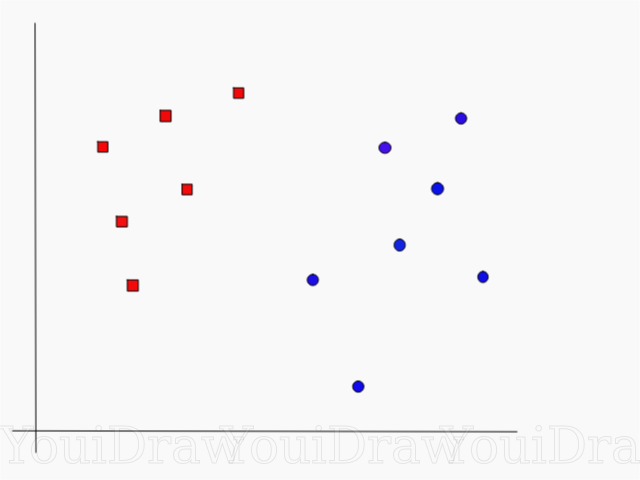
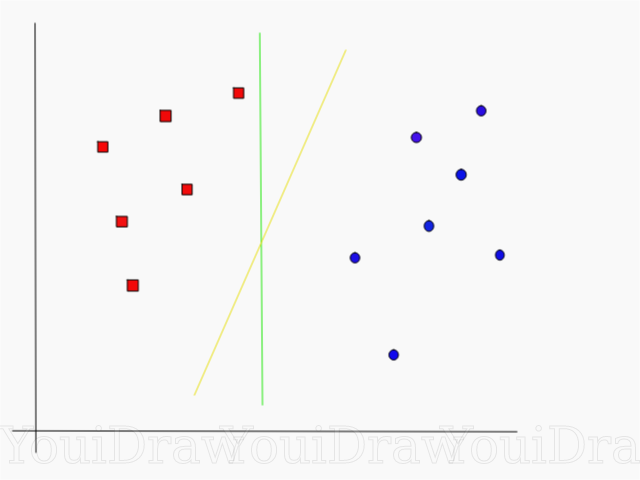
Краткий обзор алгоритма машинного обучения Метод Опорных Векторов (SVM)

План:

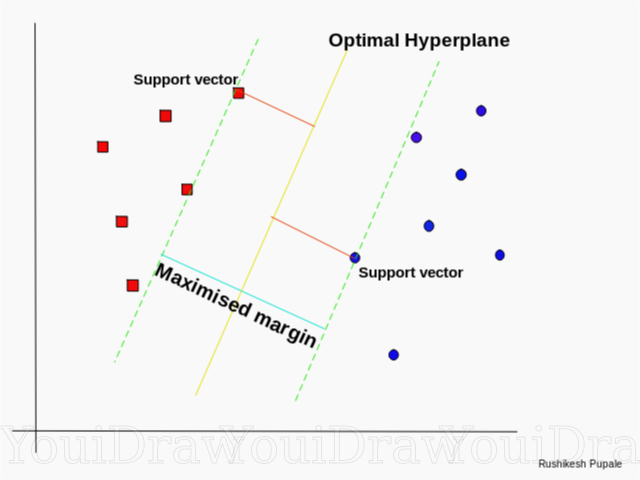
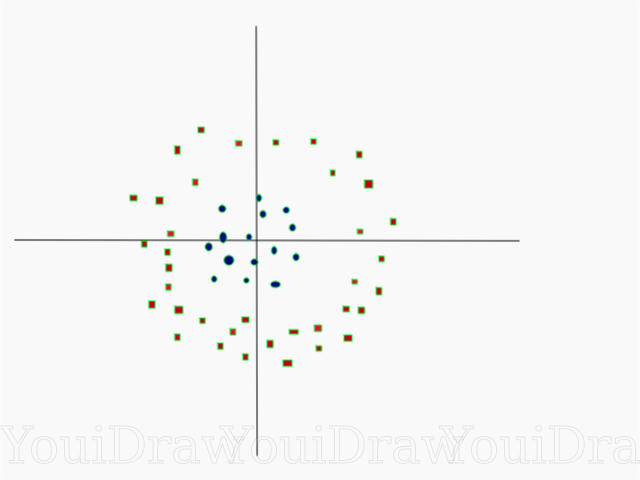
* теоретическую составляющую SVM;
* как алгоритм работает на выборках, которые невозможно разбить на классы линейно;
* пример использования на Python и имплементация алгоритма в библиотеке SciKit Learn.

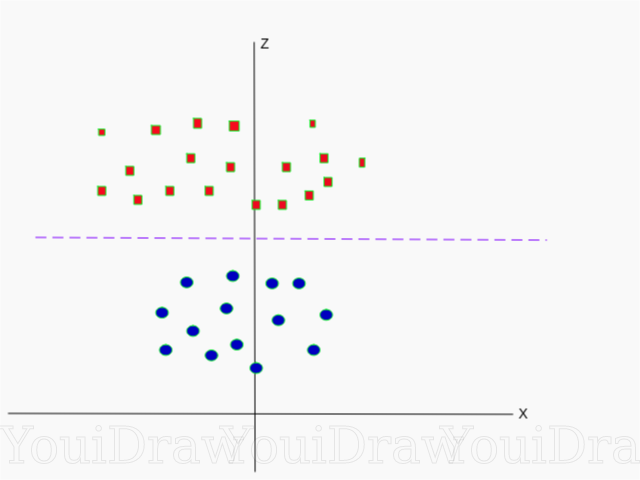
Как известно, задачи машинного обучения разделены на две основные категории — классификация и регрессия. В зависимости от того, какая из этих задач перед нами стоит, и какой у нас имеется датасет для этой задачи, мы выбираем какой именно алгоритм использовать.  
  
Метод Опорных Векторов или SVM (от англ. Support Vector Machines) — это линейный алгоритм используемый в задачах классификации и регрессии. Данный алгоритм имеет широкое применение на практике и может решать как линейные так и нелинейные задачи. Суть работы “Машин” Опорных Векторов проста: алгоритм создает линию или гиперплоскость, которая разделяет данные на классы.

Теория

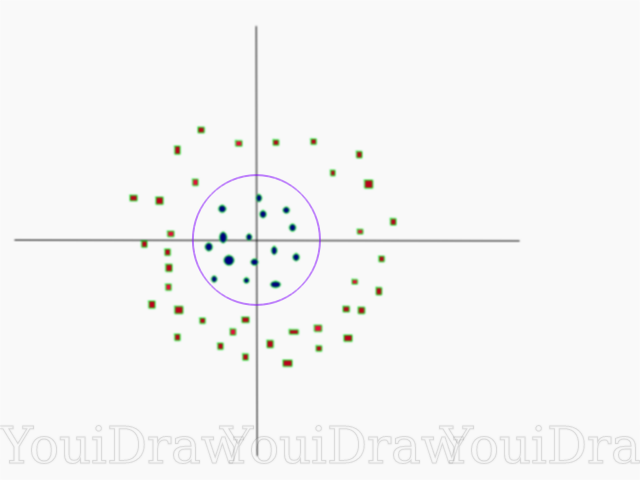
Основной задачей алгоритма является найти наиболее правильную линию, или гиперплоскость разделяющую данные на два класса. SVM это алгоритм, который получает на входе данные, и возвращает такую разделяющую линию.  
  
Рассмотрим следующий пример. Допустим у нас есть набор данных, и мы хотим классифицировать и разделить красные квадраты от синих кругов (допустим позитивное и отрицательное). Основной целью в данной задаче будет найти “идеальную” линию которая разделит эти два класса.  
  
  
  
Найдите идеальную линию, или гиперплоскость, которая разделит набор данных на синий и красный классы.  
  
На первый взгляд, не так уж и сложно, правда?  
  
Но, как вы можете заметить, нет одной, уникальной, линии, которая бы решала такую задачу. Мы можем подобрать бесконечное множество таких линий, которые могут разделить эти два класса. Как же именно SVM находит “идеальную” линию, и что в его понимании “идеальная”?  
  
Взгляните на пример ниже, и подумайте какая из двух линий (желтая или зеленая) лучше всего разделяет два класса, и подходит под описаниие “идеальной”?  
  
  
  
Какая линия лучше разделяет набор данных по вашему мнению?  
  
Если вы выбрали желтую прямую, я вас поздравляю: это та самая линия, которую бы выбрал алгоритм. В данном примере мы можем интуитивно понять что желтая линия разделяет и соответственно классифицирует два класса лучше зеленой.  
  
В случае с зеленой линией — она расположена слишком близко к красному классу. Несмотря на то, что она верно классифицировала все объекты текущего набора данных, такая линия не будет генерализованной — не будет так же хорошо вести себя с незнакомым набором данных. Задача нахождения генерализованной разделяющей двух классов является одной из основных задач в машинном обучении.

Как SVM находит лучшую линию

Алгоритм SVM устроен таким образом, что он ищет точки на графике, которые расположены непосредственно к линии разделения ближе всего. Эти точки называются опорными векторами. Затем, алгоритм вычисляет расстояние между опорными векторами и разделяющей плоскостью. Это расстояние которое называется зазором. Основная цель алгоритма — максимизировать расстояние зазора. Лучшей гиперплоскостью считается такая гиперплоскость, для которой этот зазор является максимально большим.  
  
  
  
Довольно просто, не так ли? Рассмотрим следующий пример, с более сложным датасетом, который нельзя разделить линейно.  
  
  
  
Очевидно, что этот набор данных нельзя разделить линейно. Мы не можем начертить прямую линию, которая бы классифицировала эти данные. Но, этот датасет можно разделить линейно, добавив дополнительное измерение, которое мы назовем осью Z. Представим, что координаты на оси Z регулируются следующим ограничением:

Таким образом, ордината Z представлена из квадрата расстояния точки до начала оси.  
Ниже приведена визуализация того же набора данных, на оси Z.  
  
  
  
Теперь данные можно разделить линейно. Допустим пурпурная линия разделяющая данные z=k, где k константа. Если

, то следовательно и

— формула окружности. Таким образом, мы можем спроэцировать наш линейный разделитель, обратно к исходному количеству измерений выборки, используя эту трансформацию.  
  
  
  
В результате, мы можем классифицировать нелинейный набор данных добавив к нему дополнительное измерение, а затем, привести обратно к исходному виду используя математическую трансформацию. Однако, не со всеми наборами данных можно с такой же легкостью провернуть такую трансформацию. К счастью, имплементация этого алгоритма в библиотеке sklearn решает эту задачу за нас.

Гиперплоскость

Теперь, когда мы ознакомились с логикой алгоритма, перейдем к формальному определению гиперплоскости  
  
Гиперплоскость — это n-1 мерная подплоскость в n-мерном Евклидовом пространстве, которая разделяет пространство на две отдельные части.  
  
Например, представим что наша линия представлена в виде одномерного Евклидова пространства (т.е. наш набор данных лежит на прямой). Выберите точку на этой линии. Эта точка разделит набор данных, в нашем случае линию, на две части. У линии есть одна мера, а у точки 0 мер. Следовательно, точка — это гиперплоскость линии.  
  
Для двумерного датасета, с которым мы познакомились ранее, разделяющая прямая была той самой гиперплоскостью. Проще говоря, для n-мерного пространства существует n-1 мерная гиперплоскость, разделяющая это пространство на две части.  
  
КОД

**import** numpy **as** np

X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])

y = np.array([1, 1, 2, 2])

Точки представлены в виде массива X, а классы к которому они принадлежат в виде массива y.  
Теперь мы обучим нашу модель этой выборкой. Для данного примера я задал линейный параметр “ядра” классификатора (kernel).

**from** sklearn.svm **import** SVC

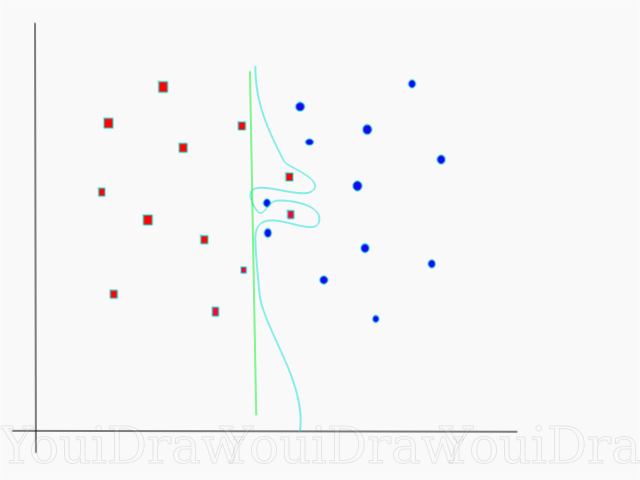
clf = SVC(kernel='linear')

clf = SVC.fit(X, y)

Предсказание класса нового объекта

prediction = clf.predict([[0,6]])

Настройка параметров

Параметры — это аргументы которые вы передаете при создании классификатора. Ниже я привел несколько самых важных настраиваемых параметров SVM:  
  
**“C”**  
  
Данный параметр помогает отрегулировать ту тонкую грань между “гладкостью” и точностью классификации объектов обучающей выборки. Чем больше значение “С” тем больше объектов обучающей выборки будут правильно классифицированы.  
  
  
  
В данном примере есть несколько порогов принятия решений, которые мы можем определить для этой конкретной выборки. Обратите внимание на прямую (представлена на графике в виде зеленой линии) порога решений. Она довольно таки проста, и по этой причине, несколько объектов были классифицированы неверно. Эти точки, которые были классифицированы неправильно называются выбросами в данных.  
  
Мы также можем настроить параметры таким образом, что в конечном итоге получим более изогнутую линию (светло-голубой порог решений), которая будет классфицировать асболютно все данные обучающей выборки правильно. Конечно, в таком случае, шансы того, что наша модель сможет генерализовать и показать столь же хорошие результаты на новых данных, катастрофически мала. Следовательно, если вы пытаетесь достигнуть точности при обучении модели, вам стоит нацелиться на что-то более ровное, прямое. Чем выше число “С” тем более запутанная гиперплоскость будет в вашей модели, но и выше число верно-классифицированных объектов обучающей выборки. Поэтому, важно “подкручивать” параметры модели под конкретный набор данных, чтобы избежать переобучения но, в то же время достигнуть высокой точности.  
  
**Гамма**  
  
В официальной документации библиотека SciKit Learn говорится, что гамма определяет насколько далеко каждый из элементов в наборе данных имеет влияние при определении “идеальной линии”. Чем ниже гамма, тем больше элементов, даже тех, которые достаточно далеки от разделяющей линии, принимают участие в процессе выбора этой самой линии. Если же, гамма высокая, тогда алгоритм будет “опираться” только на тех элементах, которые наиболее близки к самой линии.  
Если задать уровень гаммы слишком высоким, тогда в процессе принятия решения о расположении линии будут учавствовать только самые близкие к линии элементы. Это поможет игнорировать выбросы в данных. Алгоритм SVM устроен таким образом, что точки расположенные наиболее близко относительно друг друга имеют больший вес при принятии решения. Однако при правильной настройке «C» и «gamma» можно добиться оптимального результата, который построит более линейную гиперплоскость игнорирующую выбросы, и следовательно, более генерализуемую.

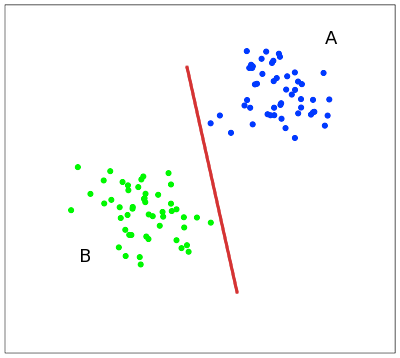
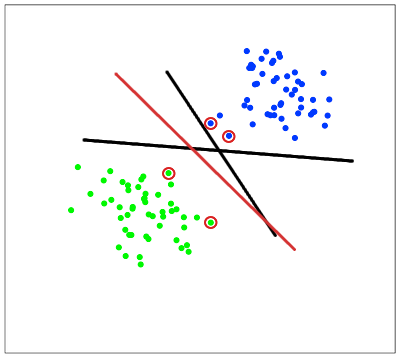
**Классификация данных методом опорных векторов**

В данной статье я хочу рассказать о проблеме классификации данных методом опорных векторов (Support Vector Machine, SVM). Такая классификация имеет довольно широкое применение: от распознавания образов или создания спам-фильтров до вычисления распределения горячих аллюминиевых частиц в ракетных выхлопах.  
Сначала несколько слов об исходной задаче. Задача классификации состоит в определении к какому классу из, как минимум, двух изначально известных относится данный объект. Обычно таким объектом является вектор в *n*-мерном вещественном пространстве https://habrastorage.org/r/w1560/storage/habraeffect/ec/9e/ec9e94384dbf5d14049bc5fa7e7d0aeb.png. Координаты вектора описывают отдельные аттрибуты объекта. Например, цвет ***c***, заданный в модели RGB, является вектором в трехмерном пространстве: ***c***=(*red, green, blue*).  
Если классов всего два («спам / не спам», «давать кредит / не давать кредит», «красное / черное»), то задача называется бинарной классификацией. Если классов несколько — многоклассовая (мультиклассовая) классификация. Также могут иметься образцы каждого класса — объекты, про которые заранее известно к какому классу они принадлежат. Такие задачи называют *обучением с учителем*, а известные данные называются *обучающей выборкой*. (Примечание: если классы изначально не заданы, то перед нами *задача кластеризации.*)  
Итак, математическая формулировка задачи классификации такова: пусть **X** — пространство объектов (например, https://habrastorage.org/r/w1560/storage/habraeffect/ec/9e/ec9e94384dbf5d14049bc5fa7e7d0aeb.png), ***Y*** — наши классы (например, ***Y*** = {-1,1}). Дана обучающая выборка: https://habrastorage.org/r/w1560/storage/habraeffect/eb/d6/ebd63e72e26bec724d5b6934f3b22c11.png. Требуется построить функцию https://habrastorage.org/r/w1560/storage/habraeffect/2a/bb/2abbfe4e68a84c7d15cf6d55d4f1e3f5.png (классификатор), сопоставляющий класс ***y*** произвольному объекту ***x.***

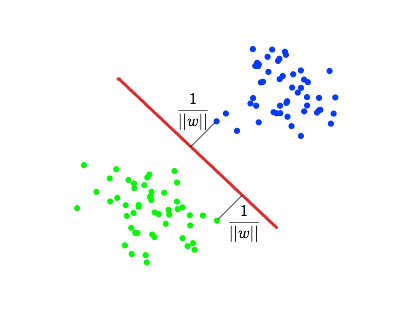
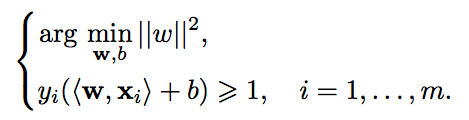
**Метод опорных векторов**

Данный метод изначально относится к бинарным классификаторам, хотя существуют способы заставить его работать и для задач мультиклассификации.

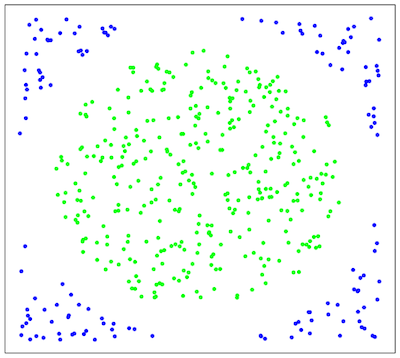
Идея метода

Идею метода удобно проиллюстрировать на следующем простом примере: даны точки на плоскости, разбитые на два класса (рис. 1). Проведем линию, разделяющую эти два класса (красная линия на рис. 1). Далее, все новые точки (не из обучающей выборки) автоматически классифицируются следующим образом:  
  
точка выше прямой попадает в класс **A**,  
точка ниже прямой — в класс **B**.  
  
  
  
Такую прямую назовем *разделяющей* прямой. Однако, в пространствах высоких размерностей прямая уже не будет разделять наши классы, так как понятие «ниже прямой» или «выше прямой» теряет всякий смысл. Поэтому вместо прямых необходимо рассматривать гиперплоскости — пространства, размерность которых на единицу меньше, чем размерность исходного пространства. В https://habrastorage.org/r/w1560/storage/habraeffect/3e/89/3e89720bd70e886270233185080bfd1a.png, например, гиперплоскость — это обычная двумерная плоскость.  
  
В нашем примере существует несколько прямых, разделяющих два класса (рис. 2):  
  
  
  
С точки зрения точности классификации лучше всего выбрать прямую, расстояние от которой до каждого класса максимально. Другими словами, выберем ту прямую, которая разделяет классы наилучшим образом (красная прямая на рис.2). Такая прямая, а в общем случае — гиперплоскость, называется оптимальной разделяющей гиперплоскостью.  
  
Вектора, лежащие ближе всех к разделяющей гиперплоскости, называются *опорными векторами* (support vectors). На рисунке 2 они помечены красным.

Немного математики

Пусть имеется обучающая выборка: https://habrastorage.org/r/w1560/storage/habraeffect/ec/0f/ec0fd2147020136102f199d847315336.png.  
  
Метод опорных векторов строит классифицирующую функцию *F* в виде https://habrastorage.org/r/w1560/storage/habraeffect/9e/39/9e39396ca18b921d9afbef6d92607ddb.png,  
где https://habrastorage.org/r/w1560/storage/habraeffect/bc/aa/bcaa9f6be3acdb74579d883fa63c21f3.png — скалярное произведение, ***w*** — нормальный вектор к разделяющей гиперплоскости,*b* — вспомогательный параметр. Те объекты, для которых *F(****x****) = 1* попадают в один класс, а объекты с*F(****x****) = -1* — в другой. Выбор именно такой функции неслучаен: любая гиперплоскость может быть задана в виде https://habrastorage.org/r/w1560/storage/habraeffect/bb/77/bb77637fa821ab46507330ed645ceb33.png для некоторых ***w*** и *b.*  
  
  
  
Далее, мы хотим выбрать такие ***w*** и *b* которые максимизируют расстояние до каждого класса. Можно подсчитать, что данное расстояние равно https://habrastorage.org/r/w1560/storage/habraeffect/31/86/3186bd99eb78eadf2af35359b431c58e.png. Проблема нахождения максимума https://habrastorage.org/r/w1560/storage/habraeffect/31/86/3186bd99eb78eadf2af35359b431c58e.png эквивалентна проблеме нахождения минимума https://habrastorage.org/r/w1560/storage/habraeffect/05/47/0547d2ad77f3163ca1641d382a173ad2.png. Запишем все это в виде задачи оптимизации:  
  
  
  
которая является стандартной задачей квадратичного программирования и решается с помощью множителей Лагранжа. Описание данного метода можно найти [в Википедии.](http://en.wikipedia.org/wiki/Lagrange_multipliers)

Линейная неразделимость

На практике случаи, когда данные можно разделить гиперплоскостью, или, как еще говорят, *линейно*, довольно редки. Пример линейной неразделимости можно видеть на рисунке 3:  
  
  
  
В этом случае поступают так: все элементы обучающей выборки вкладываются в пространство **X** более высокой размерности с помощью специального отображения https://habrastorage.org/r/w1560/storage/habraeffect/ea/d4/ead4cdea29783cc66b34f5d3706f86e8.png. При этом отображение https://habrastorage.org/r/w1560/storage/habraeffect/2b/4c/2b4ccca025f8774728d4201a3d645406.png выбирается так, чтобы в новом пространстве **X** выборка была *линейно* разделима.  
  
Классифицирующая функция *F* принимает вид https://habrastorage.org/r/w1560/storage/habraeffect/46/9f/469fa9d46d14e1afb2d5b26ff598851b.png. Выражение https://habrastorage.org/r/w1560/storage/habraeffect/8f/3b/8f3bee609b3f807563af6f7e4ae5ba37.png называется *ядром* классификатора. С математической точки зрения ядром может служить любая положительно определенная симметричная функция двух переменных. Положительная определенность необходимо для того, чтобы соответствующая функция Лагранжа в задаче оптимизации была ограничена снизу, т.е. задача оптимизации была бы корректно определена.  
  
Точность классификатора зависит, в частности, от выбора ядра. На видео можно увидеть иллюстрирацию классификации при помощи полиномиального ядра:

Чаще всего на практике встречаются следующие ядра:  
  
**Полиномиальное:** https://habrastorage.org/r/w1560/storage/habraeffect/8d/73/8d734dc93985270c76bfc0e059556264.png  
  
**Радиальная базисная функция:** https://habrastorage.org/r/w1560/storage/habraeffect/df/c6/dfc66ea3b4a833ef4033ba07362b31d3.png  
  
**Гауссова радиальная базисная функция:** https://habrastorage.org/r/w1560/storage/habraeffect/4d/b0/4db09c6643fd0bb3d19f4458707209ec.png  
  
**Сигмоид:** https://habrastorage.org/r/w1560/storage/habraeffect/35/37/353764f90f9af20442bcc97c6a9b4a08.png  
  
ПРАКТИЧЕСКАЯ ЧАСТЬ

# 1.4. Метод опорных векторов SVM [¶](https://scikit-learn.ru/1-4-support-vector-machines/#supervised-learning)

**Метод опорных векторов (Support Vector Machines** — **SVM**) — это набор контролируемых методов обучения, используемых для [классификации](https://scikit-learn.ru/1-4-support-vector-machines/#classification) , [регрессии](https://scikit-learn.ru/1-4-support-vector-machines/#regression) и [обнаружения выбросов](https://scikit-learn.ru/1-4-support-vector-machines/#density-estimation-novelty-detection) .

Преимущества:

* Эффективен в пространствах больших размеров.
* По-прежнему эффективен в случаях, когда количество измерений превышает количество образцов.
* Использует подмножество обучающих точек в функции принятия решений (называемых опорными векторами), поэтому это также эффективно с точки зрения памяти.
* Универсальность: для функции принятия решения могут быть указаны различные [функции ядра](https://scikit-learn.ru/1-4-support-vector-machines/#kernel-functions) . Предоставляются общие ядра, но также можно указать собственные ядра.

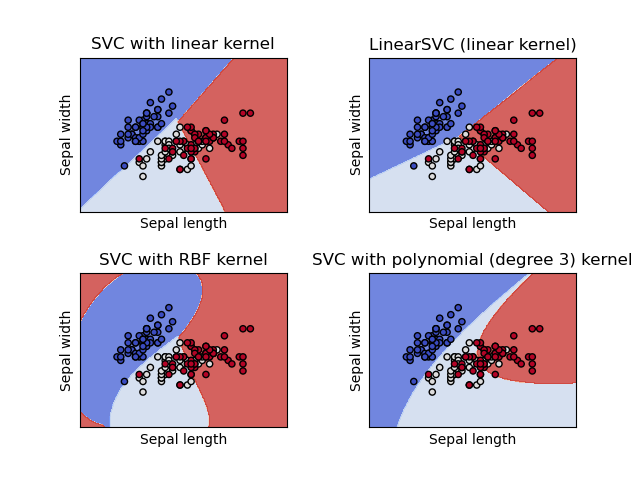
К недостаткам опорных векторных машин можно отнести:

* Если количество функций намного превышает количество выборок, избегайте чрезмерной подгонки при выборе [функций ядра,](https://scikit-learn.ru/1-4-support-vector-machines/#kernel-functions) и термин регуляризации имеет решающее значение.
* SVM не предоставляют напрямую оценки вероятностей, они рассчитываются с использованием дорогостоящей пятикратной перекрестной проверки (см. [Оценки и вероятности](https://scikit-learn.ru/1-4-support-vector-machines/#scores-probabilities) ниже).

Метод опорных векторов в scikit-learn поддерживают как плотные ( numpy.ndarray и конвертируемые в это numpy.asarray), так и разреженные (любые  scipy.sparse) выборочные векторы в качестве входных данных. Однако, чтобы использовать SVM для прогнозирования разреженных данных, он должен соответствовать этим данным. Для оптимальной производительности используйте C-порядковый numpy.ndarray (плотный) или scipy.sparse.csr\_matrix (разреженный) с dtype=float64.

## 1.4.1. Классификация

[SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC), [NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank) и [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank) являются классами, способными выполнять двоичную и мультиклассовую классификацию набора данных.



[SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) и [NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank) являются аналогичными методами, но принимают несколько разные наборы параметров и имеют разные математические формулировки (см. раздел [Математическая формулировка](https://scikit-learn.ru/1-4-support-vector-machines/#svm-mathematical-formulation) ). С другой стороны, [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank) это еще одна (более быстрая) реализация классификации опорных векторов для случая линейного ядра. Обратите внимание, что [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank) параметр не принимает **kernel**, так как предполагается, что он линейный. Ему также не хватает некоторых атрибутов [SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) и **[NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank)**, например **support\_**.

Как и другие классификаторы [**SVC**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC), **[NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank)** и **[LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank)** в качестве входных двух массивов: массив X формы **(n\_samples, n\_features),** проведение обучающих выборок, и массив y из класса меток (строки или целые числа), в форме (**n\_samples**):

1

>>> **from** sklearn **import** svm

2

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

3

>>> y = [0, 1]

4

>>> clf = svm.SVC()

5

>>> clf.fit(X, y)

6

SVC()

После установки модель может быть использована для прогнозирования новых значений:

1

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

2

array([1])

Функция принятия решения SVM (подробно описанная в [математической формулировке](https://scikit-learn.ru/1-4-support-vector-machines/#svm-mathematical-formulation) ) зависит от некоторого подмножества обучающих данных, называемых опорными векторами. Некоторые свойства этих опорных векторов можно найти в атрибутах **support\_vectors\_**, **support\_а** также **n\_support\_**:

1

>>> *# get support vectors*

2

>>> clf.support\_vectors\_

3

array([[0., 0.],

4

[1., 1.]])

5

>>> *# get indices of support vectors*

6

>>> clf.support\_

7

array([0, 1]...)

8

>>> *# get number of support vectors for each class*

9

>>> clf.n\_support\_

10

array([1, 1]...)

**Примеры:**

* [SVM: максимальное поле, разделяющее гиперплоскость](https://scikit-learn.ru/example/svm-maximum-margin-separating-hyperplane/) ,
* [Нелинейный SVM](https://scikit-learn.ru/example/non-linear-svm-2/)
* [SVM-Anova: SVM с одномерным выбором функций](https://scikit-learn.ru/example/svm-anova-svm-with-univariate-feature-selection/) ,

### 1.4.1.1. Мультиклассовая классификация

[SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) и [NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank) реализовать подход «один против одного» для классификации по нескольким классам. Всего построены**n\_classes \* (n\_classes — 1) / 2**классификаторов, каждый из которых обучает данные из двух классов. Чтобы обеспечить согласованный интерфейс с другими классификаторами, опция **decision\_function\_shape** позволяет монотонно преобразовывать результаты классификаторов «один против одного» в функцию принятия решения формы «один против остальных» **(n\_samples, n\_classes)**.

1

>>> X = [[0], [1], [2], [3]]

2

>>> Y = [0, 1, 2, 3]

3

>>> clf = svm.SVC(decision\_function\_shape='ovo')

4

>>> clf.fit(X, Y)

5

SVC(decision\_function\_shape='ovo')

6

>>> dec = clf.decision\_function([[1]])

7

>>> dec.shape[1] *# 4 classes: 4\*3/2 = 6*

8

6

9

>>> clf.decision\_function\_shape = "ovr"

10

>>> dec = clf.decision\_function([[1]])

11

>>> dec.shape[1] *# 4 classes*

12

4

С другой стороны, [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank) реализует мультиклассовую стратегию «один против остальных», таким образом обучая **n\_classes** модели.

1

>>> lin\_clf = svm.LinearSVC()

2

>>> lin\_clf.fit(X, Y)

3

LinearSVC()

4

>>> dec = lin\_clf.decision\_function([[1]])

5

>>> dec.shape[1]

6

4

См. « [Математическая формулировка»](https://scikit-learn.ru/1-4-support-vector-machines/#svm-mathematical-formulation)  для полного описания решающей функции.

Обратите внимание, что [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank) также реализуется альтернативная мультиклассовая стратегия, так называемая мультиклассовая SVM, сформулированная Краммером и Зингером [16](https://scikit-learn.ru/1-4-support-vector-machines/#link) , с использованием этой опции **multi\_class='crammer\_singer'**. На практике обычно предпочтительнее использовать классификацию «один против остальных», поскольку результаты в основном схожи, но время выполнения значительно меньше.

Для «один против остальных» [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank) атрибуты **coef\_** и **intercept\_** имеют форму **(n\_classes, n\_features)**и **(n\_classes, )**соответственно. Каждая строка коэффициентов соответствует одному из классификаторов **n\_classes** «один против остальных» и аналогичных для перехватов в порядке класса «один».

В случае «один против одного» [SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) и [NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank) расположение атрибутов немного сложнее. В случае линейного ядра атрибуты **coef\_** и **intercept\_** имеют форму **(n\_classes \* (n\_classes — 1) / 2, n\_features)**и **(n\_classes \* (n\_classes — 1) / 2)**соответственно. Это похоже на схему [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank), описанную выше, где каждая строка теперь соответствует двоичному классификатору. Порядок для классов от 0 до n: «0 против 1», «0 против 2»,… «0 против n», «1 против 2», «1 против 3», «1 против n»,. . . «П-1 против п».

Форма **dual\_coef\_** является**(n\_classes-1, n\_SV)**с довольно трудно макетом обхвата. Столбцы соответствуют опорным векторам, включенным в любой из **n\_classes \* (n\_classes — 1) / 2**классификаторов «один на один». Каждый из опорных векторов используется в**n\_classes — 1** классификаторах. Эти **n\_classes — 1** записи в каждой строке соответствует двойственным коэффициентам для этих классификаторов.

Это может быть яснее на примере: рассмотрим проблему трех классов с классом 0, имеющим три опорных вектора.  v00,v01,v02 и классы 1 и 2, имеющие два опорных вектора v10,v11 а также v20,v21 соответственно. Для каждого опорного вектора vij, есть два двойственных коэффициента. Назовем коэффициент опоры вектором vij в классификаторе между классами i а также k αi,kj. Тогда **dual\_coef\_** выглядит так:

|  |  |  |
| --- | --- | --- |
| α0,10 | α0,20 | Коэффициенты для КА класса 0 |
| α0,11 | α0,21 | Коэффициенты для КА класса 0 |
| α0,12 | α0,22 | Коэффициенты для КА класса 0 |
| α1,00 | α1,20 | Коэффициенты для КА класса 1 |
| α1,01 | α1,21 | Коэффициенты для КА класса 1 |
| α2,00 | α2,10 | Коэффициенты для КА 2 класса |
| α2,01 | α2,11 | Коэффициенты для КА 2 класса |

**Примеры:**

* [Постройте различные классификаторы SVM в наборе данных радужной оболочки глаза](https://scikit-learn.ru/example/plot-different-svm-classifiers-in-the-iris-dataset/) ,

### 1.4.1.2. Результаты и вероятности

Метод **decision\_function** из [SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) и [NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank) дает по классам баллов для каждого образца (или единого показателя на образец в двоичном случае). Если для параметра конструктора **probability** установлено значение **True**, включаются оценки вероятности членства в классе (из методов **predict\_proba** и **predict\_log\_proba**). В двоичном случае вероятности калибруются с использованием шкалы Платта [9](https://scikit-learn.ru/1-4-support-vector-machines/#link) : логистическая регрессия по оценкам SVM, согласованная с помощью дополнительной перекрестной проверки данных обучения. В случае мультикласса это расширяется в соответствии с [10](https://scikit-learn.ru/1-4-support-vector-machines/#link) .

**Примечание**

Та же процедура калибровки вероятности доступна для всех оценщиков через [CalibratedClassifierCV](https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html" \l "sklearn.calibration.CalibratedClassifierCV" \t "_blank) (см. [Калибровка вероятности](https://scikit-learn.org/stable/modules/calibration.html#calibration) ). В случае [SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) и [NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank) эта процедура встроена в [libsvm,](https://www.csie.ntu.edu.tw/~cjlin/libsvm/" \t "_blank) которая используется под капотом, поэтому она не полагается на scikit-learn **[CalibratedClassifierCV](https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html" \l "sklearn.calibration.CalibratedClassifierCV" \t "_blank)**.

Перекрестная проверка, связанная с масштабированием Platt, — дорогостоящая операция для больших наборов данных. Кроме того, оценки вероятности могут не соответствовать оценкам:

* «argmax» оценок может не быть argmax вероятностей.
* в бинарной классификации образец может быть помечен **predict** как принадлежащий к положительному классу, даже если выход **predict\_probaс** оставляет менее 0,5; и аналогично, он может быть помечен как отрицательный, даже если выход **predict\_proba** больше 0,5.

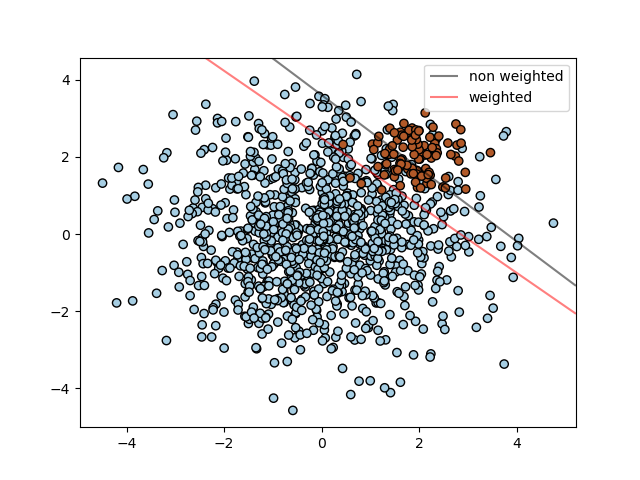
Известно также, что метод Платта имеет теоретические проблемы. Если требуются оценки достоверности, но они не обязательно должны быть вероятностями, то рекомендуется установить **probability=False** и использовать **decision\_function** вместо **predict\_proba**.

Обратите внимание, что когда **decision\_function\_shape='ovr'**и **n\_classes > 2**, в отличие от **decision\_function**, метод **predict** не пытается разорвать связи по умолчанию. Вы можете установить **break\_ties=True**, чтобы вывод **predict** был таким же, как **np.argmax(clf.decision\_function(…), axis=1)**, иначе всегда будет возвращаться первый класс среди связанных классов; но имейте в виду, что это связано с вычислительными затратами. См. [Пример разрыва связи SVM в примере](https://scikit-learn.ru/example/svm-tie-breaking-example/) .

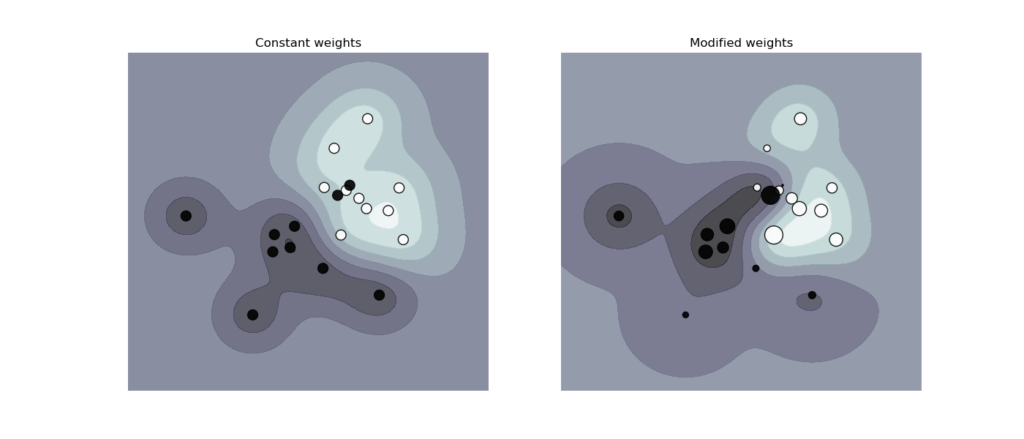
### 1.4.1.3. Несбалансированные проблемы

В задачах, где желательно придать большее значение определенным классам или определенным отдельным образцам, можно использовать параметры **class\_weight** и **sample\_weight**.

[SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) (но не **[NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank)**) реализует параметр **class\_weight** в **fit** методе. Это словарь формы **{class\_label : value}**, где значение — это число с плавающей точкой number > 0, которое устанавливает **С** для параметра класса **class\_label** значение **C \* value**. На рисунке ниже показана граница решения несбалансированной проблемы с поправкой на вес и без нее.



[**SVC**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC), [**NuSVC**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html#sklearn.svm.NuSVC), [**SVR**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR), [**NuSVR**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVR.html#sklearn.svm.NuSVR), [**LinearSVC**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC), [**LinearSVR**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html#sklearn.svm.LinearSVR) и [**OneClassSVM**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html#sklearn.svm.OneClassSVM) осуществить также веса для отдельных образцов в **fit**методе через **sample\_weight** параметр. Подобно тому **class\_weight**, это устанавливает параметр **C** для i-го примера равным **C \* sample\_weight[i]**, что побуждает классификатор правильно определять эти образцы. На рисунке ниже показано влияние взвешивания выборки на границу принятия решения. Размер кружков пропорционален весу образца:



**Примеры:**

* [SVM: разделяющая гиперплоскость для несбалансированных классов](https://scikit-learn.ru/example/svm-separating-hyperplane-for-unbalanced-classes/)
* [SVM: взвешенные образцы](https://scikit-learn.ru/example/svm-weighted-samples/) ,

## 1.4.2. Регрессия

Метод классификации опорных векторов может быть расширен для решения задач регрессии. Этот метод называется регрессией опорных векторов.

Модель, созданная с помощью классификации опорных векторов (как описано выше), зависит только от подмножества обучающих данных, потому что функция затрат для построения модели не заботится о точках обучения, которые лежат за пределами поля. Аналогичным образом модель, созданная с помощью регрессии опорных векторов, зависит только от подмножества обучающих данных, поскольку функция стоимости игнорирует образцы, прогноз которых близок к их целевому значению.

Есть три различных реализаций опорных векторов регрессии: [**SVR**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR), **[NuSVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVR.html" \l "sklearn.svm.NuSVR" \t "_blank)** и **[LinearSVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html" \l "sklearn.svm.LinearSVR" \t "_blank)**. **[LinearSVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html" \l "sklearn.svm.LinearSVR" \t "_blank)** обеспечивает более быструю реализацию, чем, [SVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR) но учитывает только линейное ядро, но [NuSVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVR.html" \l "sklearn.svm.NuSVR" \t "_blank) реализует несколько иную формулировку, чем [SVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR) и **[LinearSVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html" \l "sklearn.svm.LinearSVR" \t "_blank)**. См. [Подробности реализации](https://scikit-learn.ru/1-4-support-vector-machines/#svm-implementation-details) для получения дополнительной информации.

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

1

>>> **from** sklearn **import** svm

2

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

3

>>> y = [0.5, 2.5]

4

>>> regr = svm.SVR()

5

>>> regr.fit(X, y)

6

SVR()

7

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

8

array([1.5])

**Примеры:**

* [Поддержка векторной регрессии (SVR) с использованием линейных и нелинейных ядер](https://scikit-learn.ru/example/support-vector-regression-svr-using-linear-and-non-linear-kernels/)

## 1.4.3. Оценка плотности, обнаружение новизны

Класс [OneClassSVM](https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html" \l "sklearn.svm.OneClassSVM" \t "_blank) реализует одноклассную SVM, которая используется для обнаружения выбросов.

См. Раздел [Обнаружение](https://scikit-learn.org/stable/modules/outlier_detection.html#outlier-detection) новинок и выбросов для описания и использования **OneClassSVM**.

## 1.4.4. Сложность

Машины опорных векторов — мощные инструменты, но их требования к вычислениям и хранению быстро растут с увеличением числа обучающих векторов. Ядром SVM является задача квадратичного программирования (QP), отделяющая опорные векторы от остальной части обучающих данных. Решатель QP, используемый реализацией на основе [libsvm,](https://www.csie.ntu.edu.tw/~cjlin/libsvm/" \t "_blank) масштабируется между O(nfeatures×nsamples2) а также O(nfeatures×nsamples3) в зависимости от того, насколько эффективно используется кеш [libsvm](https://www.csie.ntu.edu.tw/~cjlin/libsvm/" \t "_blank) на практике (зависит от набора данных). Если данные очень скудныеnfeatures следует заменить на среднее количество ненулевых функций в векторной выборке.

Для линейного случая, алгоритм , используемый в [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank) по [liblinear](https://www.csie.ntu.edu.tw/~cjlin/liblinear/" \t "_blank) реализации является гораздо более эффективным , чем его [libsvm](https://www.csie.ntu.edu.tw/~cjlin/libsvm/" \t "_blank)  [SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) аналог и может масштабироваться почти линейно миллионы образцов и/или функций.

## 1.4.5. Советы по практическому использованию

* **Избегая копирования данных** : Для [**SVC**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC), [**SVR**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR), **[NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank)** и **[NuSVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVR.html" \l "sklearn.svm.NuSVR" \t "_blank)** если данные , передаваемые в некоторых методов не C упорядоченный непрерывный и двойной точности, то он будет скопирован перед вызовом основной реализации C. Вы можете проверить, является ли данный массив numpy непрерывным, проверив его **flags** атрибут.  
    
  For [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank) (и **[LogisticRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html" \l "sklearn.linear_model.LogisticRegression" \t "_blank)**) любой ввод, переданный как массив numpy, будет скопирован и преобразован в [liblinear](https://www.csie.ntu.edu.tw/~cjlin/liblinear/" \t "_blank) внутреннее разреженное представление данных (числа с плавающей запятой двойной точности и индексы int32 ненулевых компонентов). Если вы хотите вписать крупномасштабный линейный классификатор без копирования плотного numpy C-смежного массива двойной точности в качестве входных данных, мы предлагаем [SGDClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html" \l "sklearn.linear_model.SGDClassifier" \t "_blank) вместо этого использовать этот класс. Целевая функция может быть настроена почти так же, как [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank) модель.
* **Ядро Размер кэша**: Для [**SVC**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC), [**SVR**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR), **[NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank)** и **[NuSVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVR.html" \l "sklearn.svm.NuSVR" \t "_blank)**, размера кэша ядра оказывает сильное влияние на время наработки на более серьезные проблемы. Если у вас достаточно оперативной памяти, рекомендуется установить **cache\_size** более высокое значение, чем значение по умолчанию 200 (МБ), например 500 (МБ) или 1000 (МБ).
* **Настройка C** : C это 1 по умолчанию , и это разумный выбор по умолчанию. Если у вас много зашумленных наблюдений, вам следует уменьшить его: уменьшение C соответствует большей регуляризации.  
    
  [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC) и [LinearSVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html" \l "sklearn.svm.LinearSVR" \t "_blank) менее чувствительны к тому, **C**когда он становится большим, а результаты прогнозирования перестают улучшаться после определенного порога. Между тем, большие **C** значения потребуют больше времени для обучения, иногда до 10 раз дольше, как показано на рисунке [11](https://scikit-learn.ru/1-4-support-vector-machines/#link).
* Алгоритмы машины опорных векторов не масштабируются, поэтому **настоятельно рекомендуется масштабировать данные** . Например, масштабируйте каждый атрибут во входном векторе X до [0,1] или [-1, + 1] или стандартизируйте его, чтобы он имел среднее значение 0 и дисперсию 1. Обратите внимание, что такое же масштабирование должно быть применено к проверочному вектору, чтобы получить значимые результаты. Это легко сделать с помощью **[Pipeline](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html" \l "sklearn.pipeline.Pipeline" \t "_blank)**:

1

>>> **from** sklearn.pipeline **import** make\_pipeline

2

>>> **from** sklearn.preprocessing **import** StandardScaler

3

>>> **from** sklearn.svm **import** SVC

4

​

5

>>> clf = make\_pipeline(StandardScaler(), SVC())

См. Раздел « [Предварительная обработка данных»](https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing) для получения более подробной информации о масштабировании и нормализации.

* Что касается **shrinking** параметра, цитируя [12](https://scikit-learn.ru/1-4-support-vector-machines/#link) : мы обнаружили, что если количество итераций велико, то сжатие может сократить время обучения. Однако, если мы решим проблему оптимизации слабо (например, используя большой допуск остановки), код без использования сжатия может быть намного быстрее.
* Параметр **nu** в  **[NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank)**/**[OneClassSVM](https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html" \l "sklearn.svm.OneClassSVM" \t "_blank)**/**[NuSVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVR.html" \l "sklearn.svm.NuSVR" \t "_blank)** приближает долю ошибок обучения и опорных векторов.
* В [SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) случае, если данные несбалансированы (например, много положительных и мало отрицательных), установите **class\_weight='balanced'** и/или попробуйте разные параметры штрафа **C**.
* **Случайность базовых реализаций** : базовые реализации [SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) и [NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank) используют генератор случайных чисел только для перетасовки данных для оценки вероятности (когда **probability** установлено значение **True**). Этой случайностью можно управлять с помощью **random\_state** параметра. Если **probability** установлено значение, **False** эти оценки не являются случайными и random\_stateне влияют на результаты. Базовая [OneClassSVM](https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html" \l "sklearn.svm.OneClassSVM" \t "_blank) реализация аналогична реализациям [SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) и **[NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \t "_blank)**. Поскольку оценка вероятности не предусмотрена **[OneClassSVM](https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html" \l "sklearn.svm.OneClassSVM" \t "_blank)**, она не является случайной.  
    
  Базовая [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank) реализация использует генератор случайных чисел для выбора функций при подгонке модели с двойным координатным спуском (т. е. Когда **dual** установлено значение **True**). Таким образом, нередки случаи, когда для одних и тех же входных данных получаются несколько разные результаты. Если это произойдет, попробуйте использовать меньший **tol** параметр. Этой случайностью также можно управлять с помощью **random\_state** параметра. Когда **dual** установлено для **False** базовой реализации, [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank) не является случайным и **random\_state** не влияет на результаты.
* Использование штрафных санкций L1, как предусмотрено в**LinearSVC(penalty=’l1′, dual=False)**, дает разреженное решение, т.е. только подмножество весов признаков отличается от нуля и вносит вклад в функцию принятия решения. Увеличение **C** дает более сложную модель (выбирается больше функций). Значение **C**, которое дает модель «нулевой» (все веса равны нулю) может быть вычислено с использованием [l1\_min\_c](https://scikit-learn.org/stable/modules/generated/sklearn.svm.l1_min_c.html#sklearn.svm.l1_min_c).

## 1.4.6. Функции ядра

Функция ядра может быть любой из следующих:

* линейный: ⟨x,x′⟩.
* полином: (γ⟨x,x′⟩+r)d, где dуказывается параметром **degree**, r по **coef0**.
* rbf: exp⁡(−γ|x−x′|2), где γ указывается параметром **gamma**, должно быть больше 0.
* сигмовидный tanh⁡(γ⟨x,x′⟩+r), где r указывается **coef0**.

Разные ядра задаются **kernel** параметром:

1

>>> linear\_svc = svm.SVC(kernel='linear')

2

>>> linear\_svc.kernel

3

'linear'

4

>>> rbf\_svc = svm.SVC(kernel='rbf')

5

>>> rbf\_svc.kernel

6

'rbf'

### 1.4.6.1. Параметры ядра RBF

При обучении SVM с ядром Радиальной Базовой Функции (Radial Basis Function — RBF) необходимо учитывать два параметра: **C** и **gamma**. Параметр **C**, общий для всех ядер SVM, компенсирует неправильную классификацию обучающих примеров простотой поверхности принятия решений. Низкое значение **C**делает поверхность принятия решения гладкой, а высокое **C** правильные классификации всех обучающих примеров. **gamma**определяет, какое влияние имеет один обучающий пример. Чем больше **gamma**, тем ближе другие примеры должны быть затронуты.

Правильный выбор **C** и **gamma** имеет решающее значение для производительности SVM. Рекомендуется использовать [GridSearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html" \l "sklearn.model_selection.GridSearchCV" \t "_blank) с **C** и **gamma** экспоненциально далеко друг от друга, чтобы выбрать хорошие значения.

**Примеры:**

* [Параметры RBF SVM](https://scikit-learn.ru/example/rbf-svm-parameters/)
* [Нелинейный SVM](https://scikit-learn.ru/example/non-linear-svm-2/)

### 1.4.6.2. Пользовательские ядра

Вы можете определить свои собственные ядра, указав ядро ​​как функцию Python или предварительно вычислив матрицу Грама.

Классификаторы с пользовательскими ядрами ведут себя так же, как и любые другие классификаторы, за исключением того, что:

* Поле **support\_vectors\_**теперь пустое, в нем хранятся только индексы опорных векторов **support\_**
* Ссылка (а не копия) первого аргумента **fit()**метода сохраняется для использования в будущем. Если этот массив изменится между использованием **fit()**и **predict()**вы получите неожиданные результаты.

### 1.4.6.2.1. Использование функций Python в качестве ядер

Вы можете использовать собственные определенные ядра, передав функцию **kernel** параметру.

Ваше ядро должно принимать в качестве аргументов две матрицы формы **(n\_samples\_1, n\_features)**, **(n\_samples\_2, n\_features)** и возвращает матрицу ядра в форме **(n\_samples\_1, n\_samples\_2)**

Следующий код определяет линейное ядро ​​и создает экземпляр классификатора, который будет использовать это ядро:

1

>>> **import** numpy **as** np

2

>>> **from** sklearn **import** svm

3

>>> **def** my\_kernel(X, Y):

4

... return np.dot(X, Y.T)

5

...

6

>>> clf = svm.SVC(kernel=my\_kernel)

**Примеры:**

* [SVM с кастомным ядром](https://scikit-learn.ru/example/svm-with-custom-kernel/) .

### 1.4.6.2.2. Используя матрицу Грама

Вы можете передать предварительно вычисленные ядра, используя **kernel='precomputed'** параметр. Затем вы должны передать матрицу Грама вместо X к **fit** и **predict** методам. Должны быть предоставлены значения ядра между всеми обучающими векторами и тестовыми векторами:

1

>>> **import** numpy **as** np

2

>>> **from** sklearn.datasets **import** make\_classification

3

>>> **from** sklearn.model\_selection **import** train\_test\_split

4

>>> **from** sklearn **import** svm

5

>>> X, y = make\_classification(n\_samples=10, random\_state=0)

6

>>> X\_train , X\_test , y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

7

>>> clf = svm.SVC(kernel='precomputed')

8

>>> *# linear kernel computation*

9

>>> gram\_train = np.dot(X\_train, X\_train.T)

10

>>> clf.fit(gram\_train, y\_train)

11

SVC(kernel='precomputed')

12

>>> *# predict on training examples*

13

>>> gram\_test = np.dot(X\_test, X\_train.T)

14

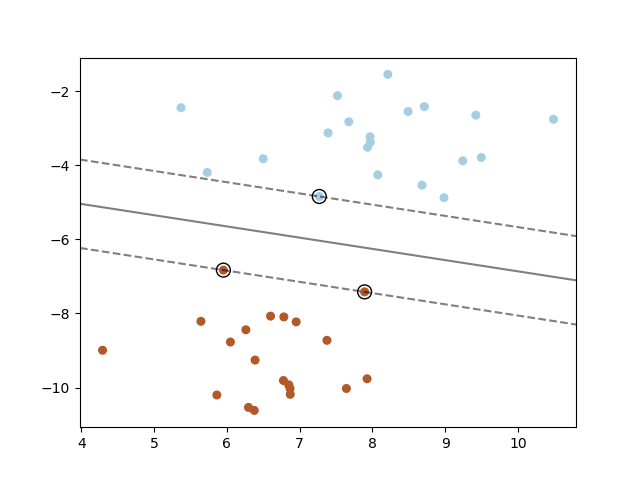
>>> clf.predict(gram\_test)

15

array([0, 1, 0])

## 1.4.7. Математическая постановка

Машина опорных векторов конструирует гиперплоскость или набор гиперплоскостей в пространстве большой или бесконечной размерности, которые можно использовать для классификации, регрессии или других задач. Интуитивно хорошее разделение достигается за счет гиперплоскости, которая имеет наибольшее расстояние до ближайших точек обучающих данных любого класса (так называемый функциональный запас), поскольку, как правило, чем больше запас, тем ниже ошибка обобщения классификатора. На рисунке ниже показана функция решения для линейно разделяемой задачи с тремя образцами на границах запаса, называемыми «векторами поддержки»:



В общем, когда проблема не является линейно разделимой, опорные векторы являются выборками в пределах границ поля.

Мы рекомендуем [13](https://scikit-learn.ru/1-4-support-vector-machines/#link) и [14](https://scikit-learn.ru/1-4-support-vector-machines/#link) как хорошие справочные материалы по теории и практике SVM.

### 1.4.7.1. SVC

Данные обучающие векторы xi∈Rp, i = 1,…, n, в двух классах, и вектор y∈{1,−1}n наша цель найти w∈Rp а также b∈R такой, что предсказание, данное sign(wTϕ(x)+b) подходит для большинства образцов.

SVC решает следующую основную проблему:  
minw,b,ζ12wTw+C∑i=1nζi  
subject to yi(wTϕ(xi)+b)≥1—ζi,  
ζi≥0,i=1,…,n

Интуитивно мы пытаемся максимизировать маржу (минимизируя ||w||2=wTw), при этом налагается штраф, если образец неправильно классифицирован или находится в пределах границ маржи. В идеале значение yi(wTϕ(xi)+b)  было бы ≥1 для всех образцов, что указывает на идеальный прогноз. Но обычно проблемы не всегда можно идеально отделить с помощью гиперплоскости, поэтому мы позволяем некоторым образцам находиться на расстоянии ζi от их правильной границы поля. Срок штрафа **C**контролирует силу этого штрафа и, как результат, действует как параметр обратной регуляризации (см. Примечание ниже).

Двойственная проблема первобытного:  
minα12αTQα—eTα  
subject to yTα=0  
0≤αi≤C,i=1,…,n

где e вектор всех единиц, а Q является n от n положительно полуопределенная матрица, Qij≡yiyjK(xi,xj), где K(xi,xj)=ϕ(xi)Tϕ(xj) это ядро. Условияαi называются двойственными коэффициентами, и они ограничены сверху величиной C. Это двойное представление подчеркивает тот факт, что обучающие векторы неявно отображаются в более высокое (возможно, бесконечное) пространство с помощью функции ϕ: см. [трюк с ядром](https://en.wikipedia.org/wiki/Kernel_method) .

Как только проблема оптимизации решена, выходные данные [функции решения](https://scikit-learn.org/stable/glossary.html#term-decision_function) для данной выборкиx становится:  
∑i∈SVyiαiK(xi,x)+b,

и предсказанный класс соответствуют его знаку. Нам нужно только просуммировать по опорным векторам (то есть выборкам, которые лежат в пределах поля), потому что двойные коэффициенты αi равны нулю для остальных образцов.

Доступ к этим параметрам можно получить через атрибуты, **dual\_coef\_** содержащие продукт.yiαi, **support\_vectors\_** который содержит опорные векторы и **intercept\_**независимый член b

**Примечание**

В то время как модели SVM, полученные из [libsvm](https://www.csie.ntu.edu.tw/~cjlin/libsvm/" \t "_blank) и [liblinear,](https://www.csie.ntu.edu.tw/~cjlin/liblinear/" \t "_blank) используют в **C**качестве параметра регуляризации, большинство других оценщиков используют alpha. Точная эквивалентность между степенью регуляризации двух моделей зависит от точной целевой функции, оптимизированной моделью. Например, когда используется оценка [Ridge](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html" \l "sklearn.linear_model.Ridge" \t "_blank) регрессии, связь между ними задается как C=1alpha

### 1.4.7.2. LinearSVC

Первичную задачу можно эквивалентно сформулировать как  
minw,b12wTw+C∑i=1max(0,yi(wTϕ(xi)+b)),

где мы используем [потерю шарнира](https://en.wikipedia.org/wiki/Hinge_loss) . Это форма, которая напрямую оптимизируется **[LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank)**, но, в отличие от двойной формы, она не включает внутренние продукты между семплами, поэтому знаменитый трюк с ядром не может быть применен. Вот почему только линейное ядро ​​поддерживается [LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \t "_blank) (ϕ — тождественная функция).

### 1.4.7.3. NuSVC

В ν-SVC рецептура [15](https://scikit-learn.ru/1-4-support-vector-machines/#link) представляет собой повторную параметризацию C-SVC и, следовательно, математически эквивалентен.

Мы вводим новый параметр ν (вместо C), который контролирует количество опорных векторов и маржинальных ошибок :ν∈(0,1] — это верхняя граница доли маржинальных ошибок и нижняя граница доли опорных векторов. Погрешность маржи соответствует образцу, лежащему не на той стороне границы маржи: она либо неправильно классифицирована, либо классифицирована правильно, но не выходит за пределы маржи.

### 1.4.7.4. SVR

Данные обучающие векторы xi∈Rp, i = 1,…, n, и вектор y∈Rn ε-SVR решает следующую основную проблему:  
minw,b,ζ,ζ∗12wTw+C∑i=1n(ζi+ζi∗)  
subject to yi—wTϕ(xi)—b≤ε+ζi,  
wTϕ(xi)+b—yi≤ε+ζi∗,  
ζi,ζi∗≥0,i=1,…,n

Здесь мы штрафуем образцы, прогноз которых не ниже εподальше от их истинной цели. Эти образцы штрафуют цель на ζi или ζi∗, в зависимости от того, лежат ли их прогнозы выше или ниже ε диапазона.

Двойная проблема:  
minα,α∗12(α—α∗)TQ(α—α∗)+εeT(α+α∗)—yT(α—α∗)  
subject to eT(α—α∗)=0  
0≤αi,αi∗≤C,i=1,…,n

где e вектор всех единиц, Q является n от n положительно полуопределенная матрица, Qij≡K(xi,xj)=ϕ(xi)Tϕ(xj) это ядро. Здесь обучающие векторы неявно отображаются в более высокое (возможно, бесконечное) пространство с помощью функции ϕ.

Прогноз такой:  
∑i∈SV(αi—αi∗)K(xi,x)+b

К этим параметрам можно получить доступ через атрибуты, **dual\_coef\_**которые содержат разницу αi—αi∗ , **support\_vectors\_** который содержит опорные векторы и **intercept\_** независимый член

### 1.4.7.5. LinearSVR

Первичную задачу можно эквивалентно сформулировать как  
minw,b12wTw+C∑i=1max(0,|yi—(wTϕ(xi)+b)|—ε),

где мы используем нечувствительные к эпсилону потери, т. е. ошибки менее ε игнорируются. Это форма, которая напрямую оптимизируется **[LinearSVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html" \l "sklearn.svm.LinearSVR" \t "_blank)**.

## 1.4.8. Детали реализации

Внутри мы используем [libsvm](https://www.csie.ntu.edu.tw/~cjlin/libsvm/" \t "_blank)[12](https://scikit-learn.ru/1-4-support-vector-machines/#id14) и [liblinear](https://www.csie.ntu.edu.tw/~cjlin/liblinear/" \t "_blank)[11](https://scikit-learn.ru/1-4-support-vector-machines/#id13) для обработки всех вычислений. Эти библиотеки обернуты с использованием C и Cython. Описание реализации и детали используемых алгоритмов см. В соответствующих документах.

**Рекомендации:**

1. Платт [«Вероятностные выходы для SVM и сравнения с регуляризованными методами правдоподобия»](https://www.cs.colorado.edu/~mozer/Teaching/syllabi/6622/papers/Platt1999.pdf) .
2. Ву, Линь и Вен, [«Оценки вероятности для классификации нескольких классов путем попарного связывания»](https://www.csie.ntu.edu.tw/~cjlin/papers/svmprob/svmprob.pdf) , JMLR 5: 975-1005, 2004.
3. Фан, Ронг-Эн и др., [«LIBLINEAR: библиотека для большой линейной классификации».](https://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf), Журнал исследований машинного обучения 9. августа (2008): 1871-1874.
4. Чанг и Лин, [LIBSVM: библиотека для машин опорных векторов](https://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf) .
5. Епископ, [Распознавание образов и машинное обучение](https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf) , глава 7 Машины с разреженным ядром
6. [«Учебное пособие по регрессии опорных векторов »](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.114.4288), Алекс Дж. Смола, Бернхард Шёлкопф — Архив статистики и вычислений Том 14, выпуск 3, август 2004 г., с. 199-222.
7. Schölkopf et. al [Новые алгоритмы опорных векторов](https://www.stat.purdue.edu/~yuzhu/stat598m3/Papers/NewSVM.pdf)
8. Краммер и Зингер [об алгоритмической реализации векторных машин на основе многоклассового ядра](http://jmlr.csail.mit.edu/papers/volume2/crammer01a/crammer01a.pdf) , JMLR 2001.

В данной статье рассмотрим метод опорных векторов (*англ. SVM, Support Vector Machine*) для задачи классификации. Будет представлена основная идея алгоритма, вывод настройки его весов и разобрана простая реализация своими руками. На примере датасета  будет продемонстрирована работа написанного алгоритма с линейно разделимыми/неразделимыми данными в пространстве  и визуализация обучения/прогноза. Дополнительно будут озвучены плюсы и минусы алгоритма, его модификации.

  
*Рисунок 1. Фото цветка ириса из открытых источников*

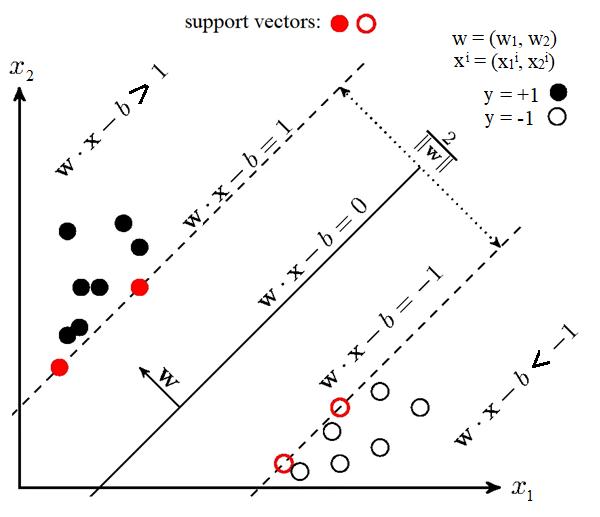
**Решаемая задача:**

Будем решать задачу бинарной (когда класса всего два) классификации. Сначала алгоритм тренируется на объектах из обучающей выборки, для которых заранее известны метки классов. Далее уже обученный алгоритм предсказывает метку класса для каждого объекта из отложенной/тестовой выборки. Метки классов могут принимать значения . Объект — вектор c N признаками  в пространстве . При обучении алгоритм должен построить функцию , которая принимает в себя аргумент  — объект из пространства  и выдает метку класса .

**Общие слова об алгоритме:**

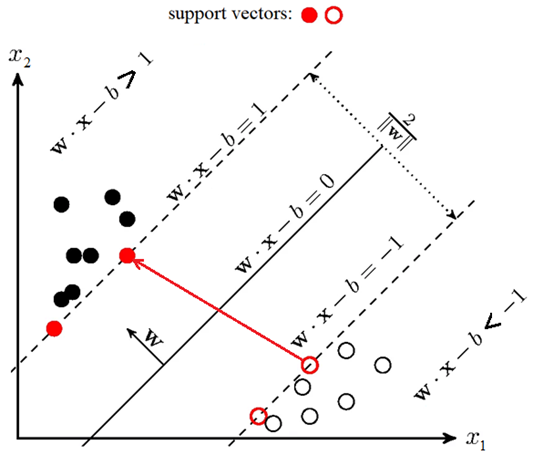
Задача классификации относится к обучению с учителем. SVM — алгоритм обучения с учителем. Наглядно многие алгоритмы машинного обучения можно посмотреть в [этой топовой статье](https://vas3k.ru/blog/machine_learning/) (см. раздел «Карта мира машинного обучения»). Нужно добавить, что SVM может применяться и для задач регрессии, но в данной статье будет разобран SVM-классификатор.

Главная цель SVM как классификатора — найти уравнение разделяющей гиперплоскости  
 в пространстве , которая бы разделила два класса неким оптимальным образом. Общий вид преобразования  объекта  в метку класса : . Будем помнить, что мы обозначили . После настройки весов алгоритма  и  (обучения), все объекты, попадающие по одну сторону от построенной гиперплоскости, будут предсказываться как первый класс, а объекты, попадающие по другую сторону — второй класс.

Внутри функции  стоит линейная комбинация признаков объекта с весами алгоритма, именно поэтому SVM относится к линейным алгоритмам. Разделяющую гиперплоскость можно построить разными способами, но в SVM веса  и  настраиваются таким образом, чтобы объекты классов лежали как можно дальше от разделяющей гиперплоскости. Другими словами, алгоритм максимизирует зазор (*англ. margin*) между гиперплоскостью и объектами классов, которые расположены ближе всего к ней. Такие объекты и называют опорными векторами (см. рис.2). Отсюда и название алгоритма.  
  
*Рисунок 2. SVM (основа рисунка*[*отсюда*](https://staesthetic.files.wordpress.com/2014/02/svm.png?w=1060)*)*

**Подробный вывод правил настройки весов SVM:**

Чтобы разделяющая гиперплоскость как можно дальше отстояла от точек выборки, ширина полосы должна быть максимальной. Вектор  — вектор нормали к разделяющей гиперплоскости. Здесь и далее будем обозначать скалярное произведение двух векторов как  или . Давайте найдем проекцию вектора, концами которого являются опорные вектора разных классов, на вектор . Эта проекция и будет показывать ширину разделяющий полосы (см. рис.3):

  
*Рисунок 3. Вывод правил настройки весов (основа рисунка*[*отсюда*](https://staesthetic.files.wordpress.com/2014/02/svm.png?w=1060)*)*

Отступом (*англ. margin*) объекта x от границы классов называется величина . Алгоритм допускает ошибку на объекте тогда и только тогда, когда отступ  отрицателен (когда  и  разных знаков). Если , то объект попадает внутрь разделяющей полосы. Если , то объект x классифицируется правильно, и находится на некотором удалении от разделяющей полосы. Т.е. алгоритм будет правильно классифицировать объекты, если выполняется условие:

Если объединить два выведенных выражения, то получим дефолтную настройку *SVM с жестким зазором* (*hard-margin SVM*), когда никакому объекту не разрешается попадать на полосу разделения. Решается аналитически через теорему Куна-Таккера. Получаемая задача эквивалентна двойственной задаче поиска седловой точки функции Лагранжа.

Всё это хорошо до тех пор, пока у нас классы линейно разделимы. Чтобы алгоритм смог работать и с линейно неразделимых данными, давайте немного преобразуем нашу систему. Позволим алгоритму допускать ошибки на обучающих объектах, но при этом постараемся, чтобы ошибок было поменьше. Введём набор дополнительных переменных , характеризующих величину ошибки на каждом объекте . Введём в минимизируемый функционал штраф за суммарную ошибку:

Будем считать количество ошибок алгоритма (когда M<0). Назовем это штрафом (*Penalty*). Тогда штраф для всех объектов будет равен сумме штрафов для каждого объекта , где  — пороговая функция (см. рис.4):

Далее сделаем штраф чувствительным к величине ошибки (чем сильнее  "уходит в минус" — тем больше штраф) и заодно введем штраф за приближение объекта к границе классов. Для этого возьмем функцию, которая ограничивает пороговую функцию ошибки (см. рис.4):

При добавлении к выражению штрафа слагаемое  получаем классическую фукцию потерь *SVM с мягким зазором* (*soft-margin SVM*) для одного объекта:

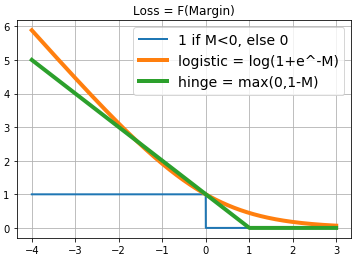
 — функция потерь, она же loss function. Именно ее мы и будем минимизировать с помощью градиентного спуска в реализации руками. Выведем правила изменения весов, где  – шаг спуска:

**Возможные вопросы на собеседованиях (основано на реальных событиях):**

После общих вопросов про SVM: *Почему именно Hinge\_loss максимизирует зазор?* – для начала вспомним, что гиперплоскость меняет свое положение тогда, когда изменяются веса  и . Веса алгоритма начинают меняться, когда градиенты лосс-функции не равны нулю (обычно говорят: “градиенты текут”). Поэтому мы специально подобрали такую лосс-функцию, у которой начинают течь градиенты в нужное время.  выглядит следующим образом: . Помним, что зазор . Когда зазор  достаточно большой ( или более), выражение  становится меньше нуля и  (поэтому градиенты не текут и веса алгоритма никак не изменяются). Если же зазор m достаточно мал (например, когда объект попадает на полосу разделения и/или отрицательный (при неверном прогнозе классификации), то Hinge\_loss становится положительной (), начинают течь градиенты и веса алгоритма изменяются. Резюмируя: градиенты текут в двух случаях: когда объект выборки попал внутрь полосы разделения и при неправильной классификации объекта.

Для проверки уровня иностранного языка возможны подобные вопросы: *What are the similarities and differences between LogisticRegression and SVM?* – firstly, we will talk about similarities: both of algorithms are linear classification algorithms in supervised learning. Some similarities are in their arguments of loss functions:  for LogReg and  for SVM (look at picture 4). Both of algorithms we can configure using gradient descent. Next let’s talk about differences: SVM return class label of object unlike LogReg, which return probability of class membership. SVM can’t work with class labels  (without renaming classes) unlike LogReg (LogReg loss finction for : , where  – real class label,  – algorithm’s return, probability of belonging object  to class). More than that, we can solve hard-margin SVM problem without gradient descent. The task of searching support vectors is reduced to search saddle point in the Lagrange function – this task refers to quadratic programming only.

**Loss function's code:**

  
*Риcунок 4. Функции потерь*

**Простая имплементация классического soft-margin SVM:**

Внимание! Ссылку на полный код вы найдете в конце статьи. Ниже будут представлены блоки кода, вырванные из контекста. Некоторые блоки можно запускать только после отработки предыдущих блоков. Под многими блоками будут размещены картинки, которые показывают, как отработал код, размещенный над ней.

**Сначала подрубим нужные библиотеки и функцию отрисовки линии:**

**Python код реализации soft-margin SVM:**

Подробно рассмотрим работу каждого блока строчек:

1) создаем функцию *add\_bias\_feature(a)*, которая автоматически расширяет вектор объектов, добавляя в конец каждого вектора число 1. Это нужно для того, чтобы «забыть» про свободный член b. Выражение  эквивалентно выражению . Мы условно считаем, что единица — это последняя компонента вектора для всех векторов x, а . Теперь настройку весов  и  будем производить одновременно.

**Код функции расширения вектора признаков:**

2) далее опишем сам классификатор. Он имеет внутри себя функции инициализации *init()*, обучения *fit()*, предсказания *predict()*, нахождения лосс функции *hinge\_loss()* и нахождения общей лосс функции классического алгоритма с мягким зазором *soft\_margin\_loss()*.

3) при инициализации вводятся 3 гиперпараметра: \_etha – шаг градиентного спуска (), \_alpha – коэффициент быстроты пропорционального уменьшения весов (перед квадратичным слагаемым в функции потерь ), \_epochs – количество эпох обучения.

**Код функции инициализации:**

4) при обучении для каждой эпохи обучающей выборки (X\_train, Y\_train) мы будем брать по одному элементу из выборки, вычислять зазор между этим элементом и положением гиперплоскости в данный момент времени. Далее в зависимости от величины этого зазора мы будем изменять веса алгоритма с помощью градиента функции потерь . Заодно будем вычислять значение этой функции на каждой эпохе и сколько раз мы изменяем веса за эпоху. Перед началом обучения убедимся, что в функцию обучения пришло действительно не больше двух разных меток класса. Перед настройкой весов происходит их инициализация с помощью нормального распределения.

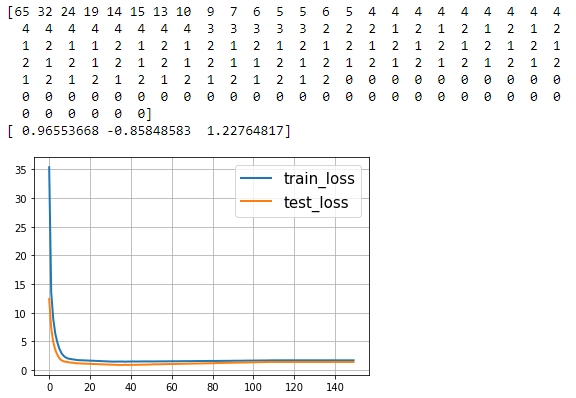
**Код функции обучения:**

**Проверка работы написанного алгоритма:**

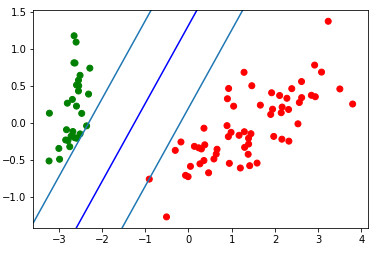
Проверим, что наш написанный алгоритм работает на каком-нибудь игрушечном наборе данных. Возьмем датасет Iris. Подготовим данные. Обозначим классы 1 и 2 как , а класс 0 как . С помощью алгоритма PCA (объяснение и применение [тут](https://habr.com/ru/company/ods/blog/325654/)) оптимальным образом сократим пространство 4-х признаков до 2-х с минимальными потерями данных (нам будет проще наблюдать за обучением и разультатом). Далее разделим на обучающую (трейн) выборку и отложенную (валидационную). Обучим на трейн выборке, прогнозируем и проверяем на отложенной. Подберем коэффициенты обучения таким образом, чтобы лосс функция падала. Во время обучения будем смотреть на лосс функцию обучающей и отложенной выборки.

**Блок подготовки данных:**

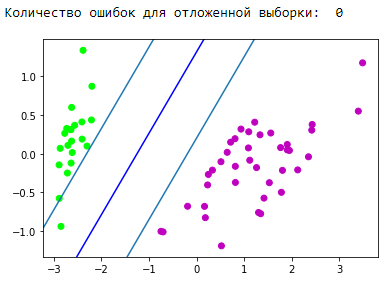
**Блок инициализации и обучения:**



**Блок визуализации получившейся разделяющей полосы:**



**Блок визуализации прогноза:**



Отлично! Наш алгоритм справился с линейно разделимыми данными. Теперь заставим его отделить классы 0 и 1 от класса 2:

**Блок подготовки данных:**

# блок подготовки данных

iris = load\_iris()

X = iris.data

Y = iris.target

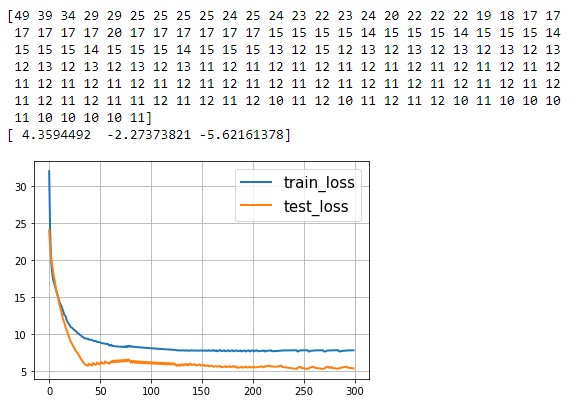
pca = PCA(n\_components=2)

X = pca.fit\_transform(X)

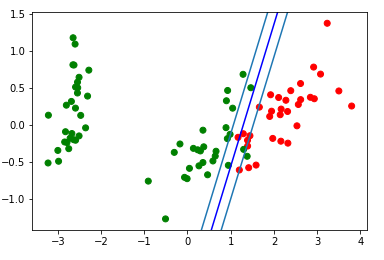
Y = (Y == 2).astype(int)\*2-1 # [0,1,2] --> [False,False,True] --> [0,1,1] --> [0,0,2] --> [-1,1,1]

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.4, random\_state=2020)

**Блок инициализации и обучения:**

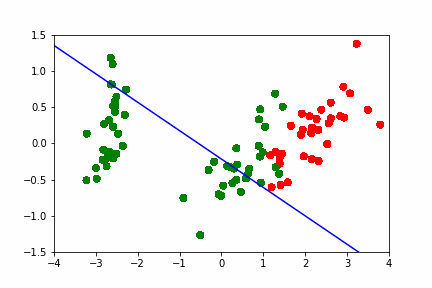


**Блок визуализации получившейся разделяющей полосы:**

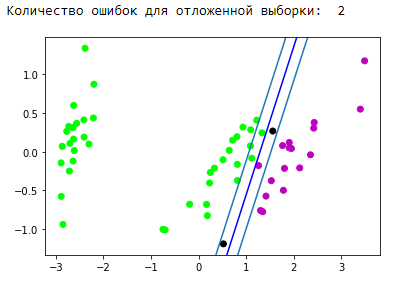


Посмотрим на гифку, которая покажет, как разделяющая прямая изменяла свое положение во время обучения (всего 500 кадров изменения весов. Первые 300 подряд. Далее 200 штук на каждый 130ый кадр):

**Код создания анимации:**



**Блок визуализации прогноза:**



**Спрямляющие пространства**

Важно понимать, что в реальных задачах не будет простого случая с линейно разделимыми данными. Для работы с подобными данными была предложена идея перехода в другое пространство, где данные будут линейно разделимы. Такое пространство и называется спрямляющим. В данной статье не будут затронуты спрямляющие пространства и ядра. Самую полную математическую теорию Вы сможете найти в [14,15,16 конспектах Е. Соколова](https://github.com/esokolov/ml-course-hse/tree/master/2017-spring/lecture-notes) и в [лекциях К.В.Воронцова](http://www.ccas.ru/voron/download/SVM.pdf).

**Применение SVM из sklearn:**

В действительности, почти все классические алгоритмы машинного обучения написаны за Вас. Приведем пример кода, алгоритм возьмем из библиотеки *sklearn*.

**Пример кода**

**Плюсы и минусы классического SVM:**

Плюсы:

1. хорошо работает с пространством признаков большого размера;
2. хорошо работает с данными небольшого объема;
3. алгоритм максимизирует разделяющую полосу, которая, как подушка безопасности, позволяет уменьшить количество ошибок классификации;
4. так как алгоритм сводится к решению задачи квадратичного программирования в выпуклой области, то такая задача всегда имеет единственное решение (разделяющая гиперплоскость с определенными гиперпараметрами алгоритма всегда одна).

Минусы:

1. долгое время обучения (для больших наборов данных);
2. неустойчивость к шуму: выбросы в обучающих данных становятся опорными объектами-нарушителями и напрямую влияют на построение разделяющей гиперплоскости;
3. не описаны общие методы построения ядер и спрямляющих пространств, наиболее подходящих для конкретной задачи в случае линейной неразделимости классов. Подбирать полезные преобразования данных – искусство.

**Применение SVM:**

Выбор того или иного алгоритма машинного обучения напрямую зависит от информации, полученной во время исследования данных. Но в общих словах, можно выделить следующие задачи:

* задачи с небольшим набором данных;
* задачи текстовой классификации. SVM дает неплохой baseline ([preprocessing] + [TF-iDF] + [SVM]), получаемая точность прогноза оказывается на уровне некоторых сверточных/рекуррентных нейронных сетей (рекомендую самостоятельно попробовать такой метод для закрепления материала). Отличный пример приведен [тут, "Часть 3. Пример одного из трюков, которым мы обучаем"](https://habr.com/ru/post/270449/);
* для многих задач со структурированными данными связка [feature engineering] + [SVM] + [kernel] "все еще торт";
* так как Hinge loss считается довольно быстро, ее можно встретить в Vowpal Wabbit (по умолчанию).

**Модификации алгоритма:**

Существуют различные дополнения и модификации метода опорных векторов, направленные на устранение определенных недостатков:

* Метод релевантных векторов (Relevance Vector Machine, RVM)
* 1-norm SVM (LASSO SVM)
* Doubly Regularized SVM (ElasticNet SVM)
* Support Features Machine (SFM)
* Relevance Features Machine (RFM)

**Дополнительные источники на тему SVM:**

1. [Текстовые лекции К.В.Воронцова](http://www.ccas.ru/voron/download/SVM.pdf)
2. [Конспекты Е.Соколова](https://github.com/esokolov/ml-course-hse/tree/master/2017-spring/lecture-notes) — 14,15,16
3. Крутой [источник](http://jermmy.xyz/images/2017-12-23/support_vector_machines_succinctly.pdf) by Alexandre Kowalczyk
4. На хабре есть 2 статьи, посвященные svm:

## ****Линейный метод опорных векторов, постановка задачи****

Даны наблюдения для обучения D, набор состоящий из n объектов, имеющих р параметров:

**D=**http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image002.png

где **y**принимает значения **-1** или **1**, определяя, какому классу принадлежит каждая точка http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image003.png.

Каждая точкаhttp://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image003.png – это вектор размерности **p**.

Требуется найти гиперплоскость максимальной разности, которая разделяет наблюдения http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image004.pngот объектов http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image005.png.

Используя знания аналитической геометрии, любую гиперплоскость можно записать как множество точек x, удовлетворяющих условию:

**w\*x-b=0,**

где \* - скалярное произведение нормали к гиперплоскости на вектор x.

Параметр http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image006.png определяет смещение гиперплоскости относительно начала координат вдоль нормали **w**.

Если обучающие данные являются линейно разделимыми, мы можем выбрать две гиперплоскости таким образом, что они отделят данные и точек между ними не будет. Затем, пытаются максимизировать расстояние между ними.

Области, ограниченной 2 гиперплоскостями называется "разностью (маржей)".

Эти гиперплоскости могут быть описаны уравнениями:

**w\*x-b=1**

**w\*x-b=-1**

Используя геометрическую интерпретацию, находим расстояние между этими гиперплоскостями - http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image007.png .

Для того чтобы дистанция была максимальной, минимизируем http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image008.png.

Чтобы исключить все точки из полосы, мы должны убедиться для всех наблюдений справедливо:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image009.png

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image010.png

Эквивалентно http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image011.png

Далее аналитическим решаем задачу оптимизации:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image012.png

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image013.png

Задача оптимизации представленная выше, трудно разрешима, так как она зависит от нормы **w**, которая включает в себя квадратный корень.

Однако задачу можно упростить, заменив на http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image014.png (коэффициент от 1/2 используются для математического удобства) без изменения решение (не менее оригинальные и модифицированные уравнения имеют одинаковый **w** и **b**) .

Это квадратичная задача оптимизации.

Более точно, нужно найти минимум:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image015.png

При ограничениях:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image013.png

Путем введения множителей Лагранжа, задача с ограничениями может быть выражена как задача без ограничений:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image016.png

Мы ищем **седловую точку**.

При этом, все точки которые могут быть отделены http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image017.png не имеют значения, поскольку мы должны установить соответствующее нулю.

Задача может быть решена с помощью квадратичного программирования.

«Стационарность» по **Куна-Такеру** означает, что решение может быть выражено как линейная комбинация обучающих векторов:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image018.png

Только несколько множителей http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image019.png будет больше 0.

Соответствующий http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image020.png – опорный вектор, который лежит на краю и выражен как http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image021.png.

Из этого следует, что опорные вектора также удовлетворяют:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image022.png

Последнее позволяет определить смещение b.

На практике применяют усреднение http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image023.png по всем опорным векторам:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image024.png

## ****Вторая форма****

Описывая правила классификации в своей безусловной форме показывает, что максимальная маржа гиперплоскости и, следовательно, задача классификации является лишь функцией опорных векторов.

Наблюдения для обучения лежат на краю.

Используя факт http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image025.png и подставляя http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image026.png**,**можно показать, что вторая форма метода опорных векторов позволяет решить проблему оптимизации:

Максимизировав по http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image019.png:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image027.png

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image028.png

Ограничение от минимизации для b

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image029.png

Ядро определено как **k(**http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image030.png**)=**http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image030.png

**W** может быть вычислено благодаря условиям:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image018.png

## ****Объективные и необъективные гиперплоскости****

Иногда требуется, чтобы гиперплоскость проходит через начало координат.

Такие гиперплоскости называются объективными, в то время как общее гиперплоскости не обязательно проходящей через начало координат называется предвзятым.

Объективная гиперплоскость может быть обеспечено путем установки **b=0** в прямой задачи оптимизации.

Соответствующая вторая форма совпадает с формой, приведенной выше, без ограничения равенства:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image031.png

## ****Случай линейной неразделимости****

В 1995 был предложен модифицированный метод Soft margin для максимизации разности, который позволяет допускать ошибки на обучающей выборке.

Если не существует гиперплоскости, которая способна разделить обучающую выборку строго на 2 класса, то метод Soft margin выберет гиперплоскость, которая разделит обучающую выборку настолько чисто (с минимальной ошибкой), насколько это возможно, в то же время, максимизируя расстояние до ближайшей точки на обучающей выборке.

Метод вводит переменную http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image032.png, которая измеряет оценку ошибки классификации http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image020.png.

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image033.png

Целевая функция возрастает за счет функции штрафа, которая возрастает, если http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image034.png.

Оптимизация теперь требует нахождения компромисса между большой разницей и штрафом за ошибку.

Если штрафная функция является линейной, то задача оптимизации становится:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image035.png

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image033.png

Минимизация http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image036.png может быть решена классическим методом Лагранжа (см. пример выше).

Вторая форма аналогична примеру выше.

Ключевым преимуществом линейной штрафной функции является то, что переменные http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image037.png исчезают в двойственной задаче, с постоянной С, появляющейся только как дополнительное ограничение на множители Лагранжа.

Нелинейные штрафных функций были использованы, в частности, для уменьшения влияния выбросов на классификаторе, но если при этом появляется проблема не выпуклости, то значительно более трудно найти глобальное решение.

## ****Нелинейный классификатор****

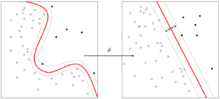


Рисунок 2 Метод с использование функции ядра

Оригинальный алгоритм для нахождения оптимальной плоскости предложил Vapnik в 1953 году – линейный классификатор.

Однако, в 1992, Bernhard E. Boser, Isabelle M. Guyon и Vladimir N. Vapnik предложили способ создания нелинейного классификатора с использованием произвольной функции ядра (kernel trick) (предложенный Айзерманом) для нахождения плоскостей максимальной разности.

Окончательный алгоритм формально прост, за исключением того, что каждое скалярное произведение заменяется нелинейной функции ядра. Это позволяет находить гиперплоскость максимально разности в измененном пространстве функций.

Изменение может быть нелинейно и трансформироваться в пространство с высокой размерностью.

Несмотря на то классификатор является гиперплоскость в многомерном пространстве функций, он может быть нелинейной в исходном пространстве обучающей выборки.

Если ядро использовало гауссовские функции радиального базиса, соответствующее пространство функций является Гильбертовым пространством с бесконечной размерностью.

Классификатор максимальной разности хорошо урегулирован, т.к. бесконечная размерность не испортит результаты.

Некоторые ядра представлены ниже:

* Однородный полином**:**http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image039.png
* Неоднородный полином: http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image040.png
* Гауссовскую функцию радиального базиса:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image041.png

* Гиперболический тангенс

Ядро связанно с преобразованием http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image042.png уравнения:

**k(**http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image043.png**)=**http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image044.png

Оценка **w** в трансформированном пространстве:

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image045.png

**Свойства**

Метод опорных векторов принадлежит к семейству общих линейных классификаторов и может быть интерпретирован как расширение персептрона. Они могут считаться специальными случаями регуляризации Тихонова.

Специальное свойство проявляется в том, что они симулируют минимальную эмпирическую ошибку классификации и максимизируют геометрическую разницу (маржу).

**Параметр выбора**

Эффективность метода опорных векторов зависит от выбора ядра, параметров ядра и чистоты параметра С для геометрической разницы.

Общий выбор – Гауссовское ядро, которое имеет один параметр http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image046.png.

Лучшую комбинацию С и γ обычно выбирают используя поиск по сетке с экспоненциальным ростом частоты http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image047.png и http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image046.png. К примеру,

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image048.png

http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image049.png

Каждая комбинация проверяется с использование кросс-проверке, и выбираются та, которая проявила себя лучше всех других на кросс-проверке.

Финальная модель, которая используя для тестирования и классификации новых данных, обучается затем на всем множестве с использованием выбранных параметров.

**Замечания**

Потенциальные недостатки метода опорных векторов включает три аспекта:

* Невозможность калибровки вероятности попадания в определенный класс
* Метод опорных векторов подходит только для решения задач с 2 классами
* Параметры модели сложно интерпретировать

**Дополнение**

Метод опорных векторов в случае существование нескольких классов

Данный метод предназначен для классификации объектов в случае нескольких (больше двух) классов.

Доминирующий подход - переход от задачи классификации на множества классов к множественной задаче разбиение на 2 класса. Общие методы для такого перехода включают:

Построение бинарных классификаторов, которые различают:

* один класс от остальных (один-против-всех)
* один класс от другого (один-против-одного)

Классификация новых объектов, используя подход один-против-всех, осуществляется путем стратегии - победитель получает все. Классификатор, с самым высоким значение функции выхода, присваивает новый объект к определенному классу (важно, что выход функции может быть прокалиброван для получения сопоставимых оценок).

Для подхода один-против-одного, классификация производится с использование стратегии - максимум-голосов победит, в которой каждый классификатор присваивает объект к одному из двух классов.

Класс с большинством голосов определяет, какой класс определить объект.

1. Ориентированный ациклический граф
2. Коды, корректирующие ошибки выхода

## ****Регрессия****

Версия метода опорных векторов для регрессии была предложена в 1996 году Vladimir N. Vapnik, Harris Drucker, Christopher J. C. Burges, Linda Kaufman и Alexander J. Smola.

Этот метод назван опорный вектор регрессии (SVR). Это модель построена с использованием опорного метода классификации (описанного выше) и зависит только от подмножества данных для обучения, т.к. штрафная функция при построении модели не обращает внимания о том, что точка лежит за пределами края.

Другая версия SVM хорошо известна как метод наименьшего квадратного опорного вектора (LS-SVM) был предложен Suykens и Vandewalle.

В машинном обучении, метод опорных векторов (SVMs) – контролируемое обучение моделей с использование схожих алгоритмов для анализа данных и распознавания шаблонов.

Данный метод используется для задач классификации и регрессионного анализа. Основной метод опорных векторов принимает набор входных данных и прогнозирует для каждого данного входа одну из двух возможных форм выхода.

Благодаря такому процессу, данный метод случайным является бинарным линейным классификатором.

Учитывая набор обучающих наблюдений (обучающую выборку), каждое из которых помечена как принадлежащая к одной из двух категорий, алгоритм обучения метода опорных векторов строит модель, которая определяет новые наблюдения в одну из категорий.

Модель метода опорных векторов – отображение данных точками в пространстве, так что между наблюдениями отдельных категорий имеется разрыв и он максимален.

Затем новые наблюдения отобразятся в том же пространстве и будут относиться к одной из категорий в зависимости от того, по какую сторону разрыва они отобразились.

В дополнение к выполнению линейной классификации, метод опорных векторов может эффективно выполнять нелинейную классификацию, используя **kernel** **trick** произвольная функция ядра), неявное отображая входные данные в многомерные пространства функций.

## ****Заключение****

Метод опорных векторов создает гиперплоскость или набор гиперплоскостей в многомерном или бесконечномерном пространстве, которые могут быть использованы для решения задач классификации, регрессии и других близких задач.

Интуитивно, хорошее разделение достигается за счет гиперплоскости, которая имеет самое большое расстояние до ближайшей точки обучающей выборке любого класса, поскольку в общем случае, чем больше расстояние, тем меньше ошибка классификатора.

В то время как исходная задача может быть сформулирована в конечномерном пространстве, часто случается, что наблюдения для дискриминации не являются линейно разделимыми в этом пространстве.

Поэтому было предложено исходное конечномерное пространство отображать в пространство большей размерности, что делает разделение намного проще.

Чтобы сохранить вычислительную нагрузку разумной, отображения, используемые в алгоритмах метода опорных векторов, обеспечивают простоту вычисления точек в терминах переменных в исходном пространстве, в терминах функции ядра **K(x,y)**.

Гиперплоскость в многомерном пространстве определяется как набор точек, для которых скалярное произведение с вектором в этом пространстве является константой.

Векторы, определяющие гиперплоскости, могут быть выбраны как линейные комбинации с параметрами http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image019.png изображения функции векторов, встречающихся в базе данных.

При таком выборе гиперплоскости, точки x в пространстве функций отображаются в гиперплоскость и определяются соотношением: http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image050.png.

Заметим, что если **K(x,y)** становиться малой при отдалении y от x, то каждый элемент в сумме измеряет степень близости от контрольной точки x до соответствующей точки http://statsoft.ru/upload/medialibrary/method-opornyh-vektorov/image020.png.

Таким образом, сумма ядер выше может быть использована для измерения относительной близости каждой контрольной точки до точек из данных, принадлежащих одной или другой категории при дискриминации.

Обратите внимание на то, что множество точек, отображающихся в любой гиперплоскости, может быть довольно запутанным.

В результате требуется более сложная дискриминация между наборами, которые не являются выпуклыми в целом в исходном пространстве.

Машины опорных векторов

Используется пакет LS-SVMLab vl.7. Пример классификации модельных данных. Каждая строка матрицы данных представляет вектор выборки.

X = 2.\*rand(30,2)-l;

Y = sign(sin(X(:,l)) + X(:,2));

X

* -0.2967 -0.0989
* 0.6617 -0.8324
* 0.1705 -0.5420
* 0.0994 0.8267
* 0.8344 -0.6952
* -0.4283 0.6516
* 0.5144 0.0767
* 0.5075 0.9923
* -0.2391 -0.8436
* 0.1356 -0.1146
* -0.8483 -0.7867
* -0.8921 0.9238
* 0.0616 -0.9907
* 0.5583 0.5498
* 0.8680 0.6346
* -0.7402 0.7374
* 0.1376 -0.8311
* -0.0612 -0.2004
* -0.9762 -0.4803
* -0.3258 0.6001
* -0.6756 -0.1372
* 0.5886 0.8213
* -0.3776 -0.6363
* 0.0571 -0.4724
* -0.6687 -0.7089
* 0.2040 -0.7279
* -0.4741 0.7386
* 0.3082 0.1594
* 0.3784 0.0997
* 0.4963 -0.7101

Для того, чтобы создать модель LS-SVM (с Гауссовым ядром RBF), требуются два параметра настройки:

у (gam) — параметр регуляризации, определяющий компромисс между минимумом ошибки обучения и гладкостью. В общем случае Гауссова ядра RBF, о2 (sig2) — квадрат полосы:

gam = 10;

sig2 = 0.4;

type = ’classification’;

[alpha,b] = trainlssvm( {X,Y,type,gam,sig2,’RBF\_kernel’});

Параметры и переменные относящиеся к LS-SVM передаются как одна ячейка (cell). Эта ячейка (cell) позволяет соответствующим образом обращаться с параметрами по умолчанию LS-SVM и синтаксически группировать связанные аргументы. Соответствующий объектно-ориентированный интерфейс LS-SVMlab приводит к более короткому вызову функций (см. demomodel). По умолчанию, данные обрабатываются применяя функцию prelssvm к необработанным данным и функцию postlssvm для предсказаний модели. Эта опция может быть явно выключена при вызове:

[alpha,b] = trainlssvm({X,Y, type, gam, sig2,’RBF\_kernel’, ’original’});

Или может быть включена (по умолчанию):

[alpha,b] = trainlssvm({X,Y, type, gam, sig2,’RBF\_kernel’, ’preprocess’});

Для анализа новых точек для этой модели, используется функция simlssvm.

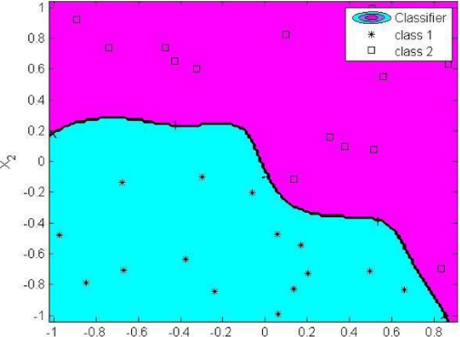
Xt = 2.\*rand(10,2) - 1;

Y test = simlssvm( {X,Y,type,gam,sig2, ’ RBF\_kernel ’}, {alpha,b} ,Xt);

Результат LS-SVM может быть показан, если входные данные двумерные.

plotlssvm( {X, Y,type,gam,sig2, ’RBF\_kernel’}, {alpha,b});

LS-SVM1}® „ with 2 different classes y=10 ,a =0.2



X1

Рис. 10.1. **Граница, генерированная в задаче классификации двумерных данных.**