ТЕМА 1: ЛИНЕЙНЫЕ МОДЕЛИ КЛАССИФИКАЦИИ И РЕГРЕССИИ

Детально обсудим очень важный класс моделей машинного обучения – линейных. Ключевое отличие нашей подачи материала от аналогичной в курсах эконометрики и статистики – это акцент на практическом применении линейных моделей в реальных задачах (хотя и математики тоже будет немало).

1. Линейная регрессия

***Метод наименьших квадратов***

Рассказ про линейные модели мы начнем с линейной регрессии. В первую очередь, необходимо задать модель зависимости объясняемой переменной  от объясняющих ее факторов, функция зависимости будет линейной: . Если мы добавим фиктивную размерность  для каждого наблюдения, тогда линейную форму можно переписать чуть более компактно, записав свободный член  под сумму: . Если рассматривать матрицу наблюдения-признаки, у которой в строках находятся примеры из набора данных, то нам необходимо добавить единичную колонку слева. Зададим модель следующим образом:



где

 – объясняемая (или целевая) переменная;

 – вектор параметров модели (в машинном обучении эти параметры часто называют весами);

 – матрица наблюдений и признаков размерности  строк на  столбцов (включая фиктивную единичную колонку слева) с полным рангом по столбцам: ;

 – случайная переменная, соответствующая случайной, непрогнозируемой ошибке модели.

Можем выписать выражение для каждого конкретного наблюдения



Также на модель накладываются следующие ограничения (иначе это будет какая то другая регрессия, но точно не линейная):

* матожидание случайных ошибок равно нулю: ;
* дисперсия случайных ошибок одинакова и конечна, это свойство называется гомоскедастичностью: ;
* случайные ошибки не скоррелированы: .

Оценка  весов  называется линейной, если



где  зависит только от наблюдаемых данных  и почти наверняка нелинейно. Так как решением задачи поиска оптимальных весов будет именно линейная оценка, то и модель называется линейной регрессией. Введем еще одно определение. Оценка  называется несмещенной тогда, когда матожидание оценки равно реальному, но неизвестному значению оцениваемого параметра:



Один из способов вычислить значения параметров модели является метод наименьших квадратов (МНК), который минимизирует среднеквадратичную ошибку между реальным значением зависимой переменной и прогнозом, выданным моделью:



Для решения данной оптимизационной задачи необходимо вычислить производные по параметрам модели, приравнять их к нулю и решить полученные уравнения относительно :





Итак, имея в виду все определения и условия описанные выше, мы можем утверждать, опираясь на теорему [Маркова-Гаусса](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B5%D0%BC%D0%B0_%D0%93%D0%B0%D1%83%D1%81%D1%81%D0%B0_%E2%80%94_%D0%9C%D0%B0%D1%80%D0%BA%D0%BE%D0%B2%D0%B0) (), что оценка МНК является лучшей оценкой параметров модели, среди всех линейных и несмещенных оценок, то есть обладающей наименьшей дисперсией.

***Метод максимального правдоподобия***

Могли возникнуть вопросы: например, почему мы минимизируем среднеквадратичную ошибку, а не что-то другое. Ведь можно минимизировать среднее абсолютное значение невязки или еще что-то. Единственное, что произойдёт в случае изменения минимизируемого значения, так это то, что мы выйдем из условий теоремы Маркова-Гаусса, и наши оценки перестанут быть лучшими среди линейных и несмещенных.

Давайте перед тем как продолжить, сделаем лирическое отступление, чтобы проиллюстрировать метод максимального правдоподобия на простом примере.

Как-то после школы я заметил, что все помнят формулу этилового спирта. Тогда я решил провести эксперимент: помнят ли люди более простую формулу метилового спирта: . Было опрошено 400 человек и оказалось, что формулу помнят всего 117 человек. Разумно предположить, что вероятность того, что следующий опрошенный знает формулу метилового спирта – . Покажем, что такая интуитивно понятная оценка не просто хороша, а еще и является оценкой максимального правдоподобия.

Разберемся, откуда берется эта оценка, а для этого вспомним [определение распределения Бернулли](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%91%D0%B5%D1%80%D0%BD%D1%83%D0%BB%D0%BB%D0%B8): случайная величина  имеет распределение Бернулли, если она принимает всего два значения ( и  с вероятностями  и  соответственно) и имеет следующую функцию распределения вероятности:



Похоже, это распределение – то, что нам нужно, а параметр распределения  и есть та оценка вероятности того, что человек знает формулу метилового спирта. Мы проделали 400 независимых экспериментов, обозначим их исходы как . Запишем правдоподобие наших данных (наблюдений), то есть вероятность наблюдать 117 реализаций случайной величины  и 283 реализации :



Далее будем максимизировать это выражение по , и чаще всего это делают не с правдоподобием , а с его логарифмом (применение монотонного преобразования не изменит решение, но упростит вычисления):





Теперь мы хотим найти такое значение , которое максимизирует правдоподобие, для этого мы возьмем производную по , приравняем к нулю и решим полученное уравнение:





Получается, что наша интуитивная оценка – это и есть оценка максимального правдоподобия. Применим теперь те же рассуждения для задачи линейной регрессии и попробуем выяснить, что лежит за среднеквадратичной ошибкой. Для этого нам придется посмотреть на линейную регрессию с вероятностной точки зрения. Модель, естественно, остается такой же:



но будем теперь считать, что случайные ошибки берутся из [центрированного нормального распределения](https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5):



Перепишем модель:



Так как примеры берутся независимо (ошибки не скоррелированы – одно из условий теоремы Маркова-Гаусса), то полное правдоподобие данных будет выглядеть как произведение функций плотности . Рассмотрим логарифм правдоподобия, что позволит нам перейти от произведения к сумме:



Мы хотим найти гипотезу максимального правдоподобия, т.е. нам нужно максимизировать выражение , а это то же самое, что и максимизация его логарифма. Обратите внимание, что при максимизации функции по какому-то параметру можно выкинуть все члены, не зависящие от этого параметра:



Таким образом, мы увидели, что максимизация правдоподобия данных – это то же самое, что и минимизация среднеквадратичной ошибки (при справедливости указанных выше предположений). Получается, что именно такая функция оценки является следствием того, что ошибка распределена нормально, а не как-то по-другому.

***Разложение ошибки на смещение и разброс (Bias-variance decomposition)***

Поговорим немного о свойствах ошибки прогноза линейной регрессии (в принципе эти рассуждения верны для всех алгоритмов машинного обучения). В свете предыдущего пункта мы выяснили, что:

* истинное значение целевой переменной складывается из некоторой детерминированной функции  и случайной ошибки : ;
* ошибка распределена нормально с центром в нуле и некоторым разбросом: ;
* истинное значение целевой переменной тоже распределено нормально:  мы пытаемся приблизить детерминированную, но неизвестную функцию  линейной функцией от регрессоров , которая, в свою очередь, является точечной оценкой функции  в пространстве функций (точнее, мы ограничили пространство функций параметрическим семейством линейных функций), т.е. случайной переменной, *у* которой есть среднее значение и дисперсия.

Тогда ошибка в точке  раскладывается следующим образом:



Для наглядности опустим обозначение аргумента функций. Рассмотрим каждый член в отдельности, первые два расписываются легко по формуле :



Пояснения:





И теперь последний член суммы. Мы помним, что ошибка и целевая переменная независимы друг от друга:



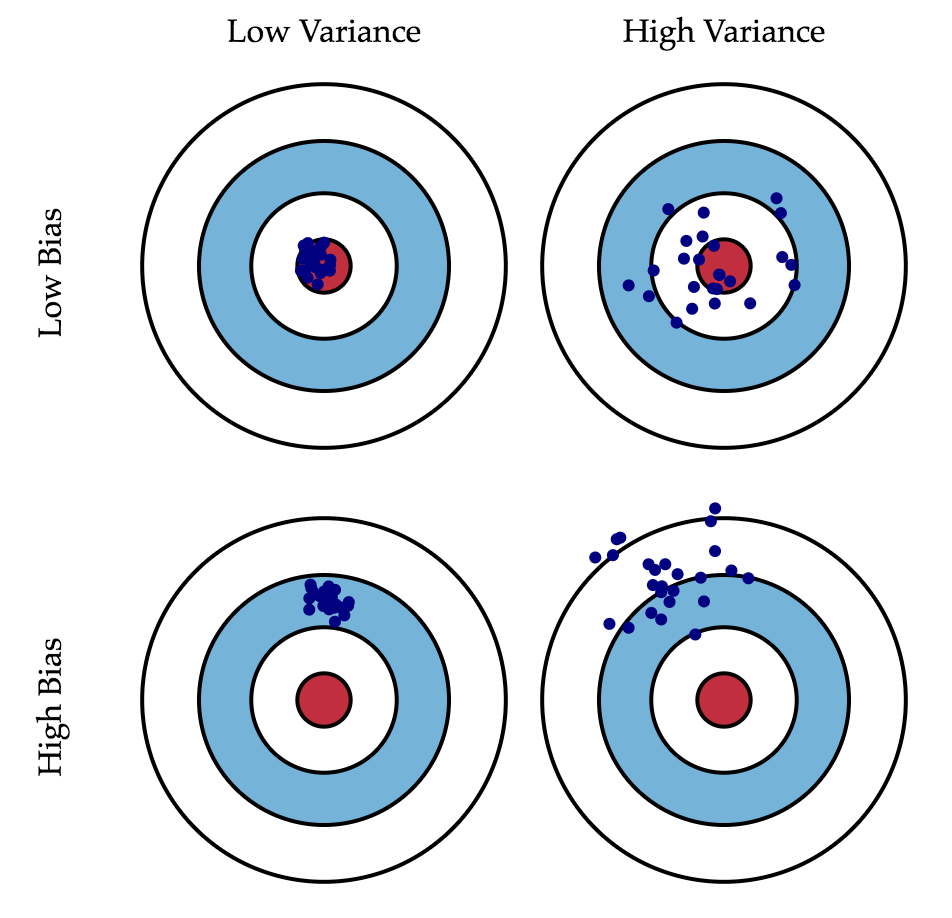
Наконец, собираем все вместе:



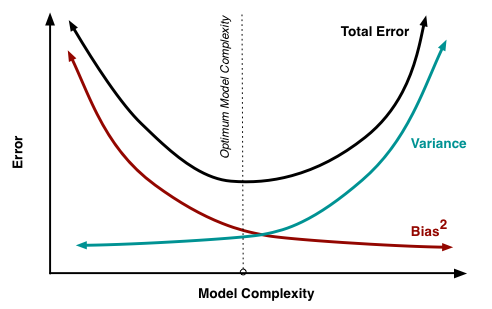
Итак, мы достигли цели всех вычислений, описанных выше, последняя формула говорит нам, что ошибка прогноза любой модели вида  складывается из:

* квадрата смещения:  – средняя ошибка по всевозможным наборам данных;
* дисперсии:  – вариативность ошибки, то, на сколько ошибка будет отличаться, если обучать модель на разных наборах данных;
* неустранимой ошибки: .

Если с последней мы ничего сделать не можем, то на первые два слагаемых мы можем как-то влиять. В идеале, конечно же, хотелось бы свести на нет оба этих слагаемых (левый верхний квадрат рисунка), но на практике часто приходится балансировать между смещенными и нестабильными оценками (высокая дисперсия).

Как правило, при увеличении сложности модели (например, при увеличении количества свободных параметров) увеличивается дисперсия (разброс) оценки, но уменьшается смещение. Из-за того, что тренировочный набор данных полностью запоминается вместо обобщения, небольшие изменения приводят к неожиданным результатам (переобучение). Если же модель слабая, то она не в состоянии выучить закономерность, в результате выучивается что-то другое, смещенное относительно правильного решения.

Теорема Маркова-Гаусса как раз утверждает, что МНК-оценка параметров линейной модели является самой лучшей в классе несмещенных линейных оценок, то есть с наименьшей дисперсией. Это значит, что если существует какая-либо другая несмещенная модель  тоже из класса линейных моделей, то мы можем быть уверены, что .



***Регуляризация линейной регрессии***

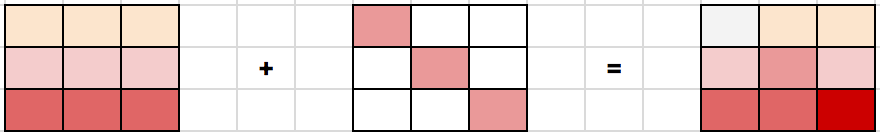
Иногда бывают ситуации, когда мы намеренно увеличиваем смещенность модели ради ее стабильности, т.е. ради уменьшения дисперсии модели . Одним из условий теоремы Маркова-Гаусса является полный столбцовый ранг матрицы . В противном случае решение МНК  не существует, т.к. не будет существовать обратная матрица  Другими словами, матрица  будет сингулярна, или вырождена. Такая задача называется некорректно поставленной. Задачу нужно скорректировать, а именно, сделать матрицу  невырожденной, или регулярной (именно поэтому этот процесс называется регуляризацией). Чаще в данных мы можем наблюдать так называемую мультиколлинеарность — когда два или несколько признаков сильно коррелированы, в матрице  это проявляется в виде "почти" линейной зависимости столбцов. Например, в задаче прогнозирования цены квартиры по ее параметрам "почти" линейная зависимость будет у признаков "площадь с учетом балкона" и "площадь без учета балкона". Формально для таких данных матрица  будет обратима, но из-за мультиколлинеарности у матрицы  некоторые собственные значения будут близки к нулю, а в обратной матрице  появятся экстремально большие собственные значения, т.к. собственные значения обратной матрицы – это . Итогом такого шатания собственных значений станет нестабильная оценка параметров модели, т.е. добавление нового наблюдения в набор тренировочных данных приведёт к совершенно другому решению. Одним из способов регуляризации является [регуляризация Тихонова](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D1%80%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8_%D0%A2%D0%B8%D1%85%D0%BE%D0%BD%D0%BE%D0%B2%D0%B0), которая в общем виде выглядит как добавление нового члена к среднеквадратичной ошибке:



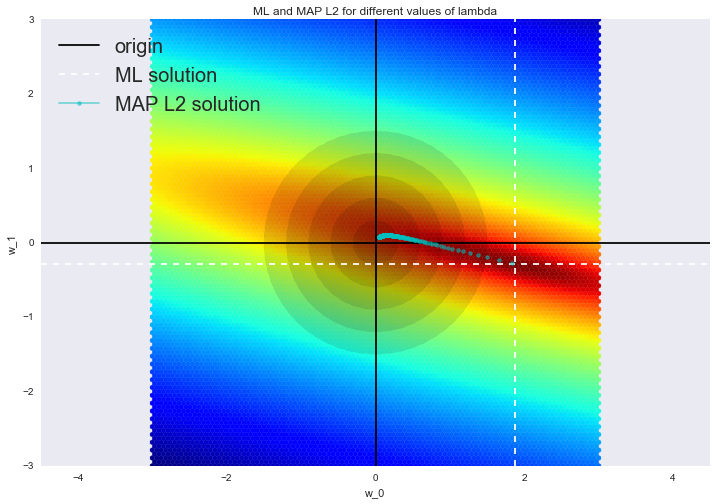
Часто матрица Тихонова выражается как произведение некоторого числа на единичную матрицу: . В этом случае задача минимизации среднеквадратичной ошибки становится задачей с ограничением на  норму. Если продифференцировать новую функцию стоимости по параметрам модели, приравнять полученную функцию к нулю и выразить , то мы получим точное решение задачи.



Такая регрессия называется гребневой регрессией (ridge regression). А гребнем является как раз диагональная матрица, которую мы прибавляем к матрице , в результате получается гарантированно регулярная матрица.



Такое решение уменьшает дисперсию, но становится смещенным, т.к. минимизируется также и норма вектора параметров, что заставляет решение сдвигаться в сторону нуля. На рисунке ниже на пересечении белых пунктирных линий находится МНК-решение. Голубыми точками обозначены различные решения гребневой регрессии. Видно, что при увеличении параметра регуляризации  решение сдвигается в сторону нуля.

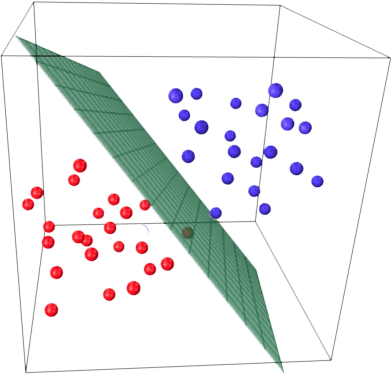


2. ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ

*Линейный классификатор*

Основная идея линейного классификатора заключается в том, что признаковое пространство может быть разделено гиперплоскостью на два полупространства, в каждом из которых прогнозируется одно из двух значений целевого класса.

Если это можно сделать без ошибок, то обучающая выборка называется линейно разделимой.



Мы уже знакомы с линейной регрессией и методом наименьших квадратов. Рассмотрим задачу бинарной классификации, причем метки целевого класса обозначим "+1" (положительные примеры) и "-1" (отрицательные примеры).

Один из самых простых линейных классификаторов получается на основе регрессии вот таким образом:



где

 – вектор признаков примера (вместе с единицей);

 – веса в линейной модели (вместе со смещением );

 – функция "сигнум", возвращающая знак своего аргумента;

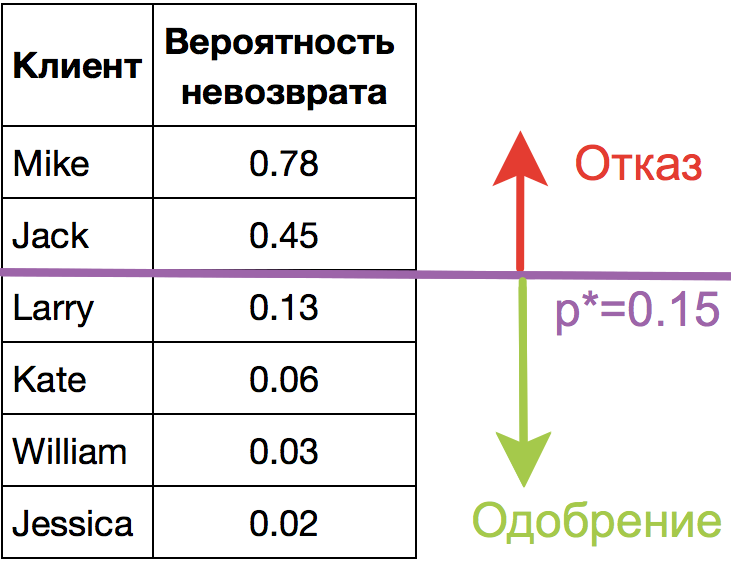
 – ответ классификатора на примере .

*Логистическая регрессия как линейный классификатор*

Логистическая регрессия является частным случаем линейного классификатора, но она обладает хорошим "умением" – прогнозировать вероятность  отнесения примера  к классу "+":

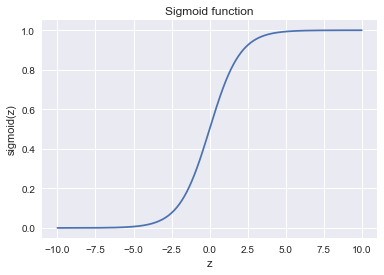


Прогнозирование не просто ответа ("+1" или "-1"), а именно вероятности отнесения к классу "+1" во многих задачах является очень важным бизнес-требованием. Например, в задаче кредитного скоринга, где традиционно применяется логистическая регрессия, часто прогнозируют вероятность невозврата кредита (). Клиентов, обратившихся за кредитом, сортируют по этой предсказанной вероятности (по убыванию), и получается скоркарта — по сути, рейтинг клиентов от плохих к хорошим. Ниже приведен игрушечный пример такой скоркарты.



Банк выбирает для себя порог  предсказанной вероятности невозврата кредита (на картинке – 0.15) и начиная с этого значения уже не выдает кредит. Более того, можно умножить предсказанную вероятность на выданную сумму и получить матожидание потерь с клиента, что тоже будет хорошей бизнес-метрикой (Далее в комментариях специалисты по скорингу могут поправить, но главная суть примерно такая).

Итак, мы хотим прогнозировать вероятность , а пока умеем строить линейный прогноз с помощью МНК: . Каким образом преобразовать полученное значение в вероятность, пределы которой – [0, 1]? Очевидно, для этого нужна некоторая функция  В модели логистической регрессии для этого берется конкретная функция: . И сейчас разберемся, каковы для этого предпосылки.



Обозначим  вероятностью происходящего события . Тогда отношение вероятностей  определяется из , а это — отношение вероятностей того, произойдет ли событие или не произойдет. Очевидно, что вероятность и отношение шансов содержат одинаковую информацию. Но в то время как  находится в пределах от 0 до 1,  находится в пределах от 0 до .

Если вычислить логарифм  (то есть называется логарифм шансов, или логарифм отношения вероятностей), то легко заметить, что . Его-то мы и будем прогнозировать с помощью МНК.

Посмотрим, как логистическая регрессия будет делать прогноз  (пока считаем, что веса  мы как-то получили (т.е. обучили модель), далее разберемся, как именно).

Шаг 1. Вычислить значение . (уравнение  задает гиперплоскость, разделяющую примеры на 2 класса);

Шаг 2. Вычислить логарифм отношения шансов: .

Шаг 3. Имея прогноз шансов на отнесение к классу "+" – , вычислить  с помощью простой зависимости:



В правой части мы получили как раз сигмоид-функцию.

Итак, логистическая регрессия прогнозирует вероятность отнесения примера к классу "+" (при условии, что мы знаем его признаки и веса модели) как сигмоид-преобразование линейной комбинации вектора весов модели и вектора признаков примера:



Следующий вопрос: как модель обучается? Тут мы опять обращаемся к принципу максимального правдоподобия.

*Принцип максимального правдоподобия и логистическая регрессия*

Теперь посмотрим, как из принципа максимального правдоподобия получается оптимизационная задача, которую решает логистическая регрессия, а именно – минимизация логистической функции потерь.

Только что мы увидели, что логистическая регрессия моделирует вероятность отнесения примера к классу "+" как



Тогда для класса "-" аналогичная вероятность:



Оба этих выражения можно ловко объединить в одно (следите за моими руками – не обманывают ли вас):



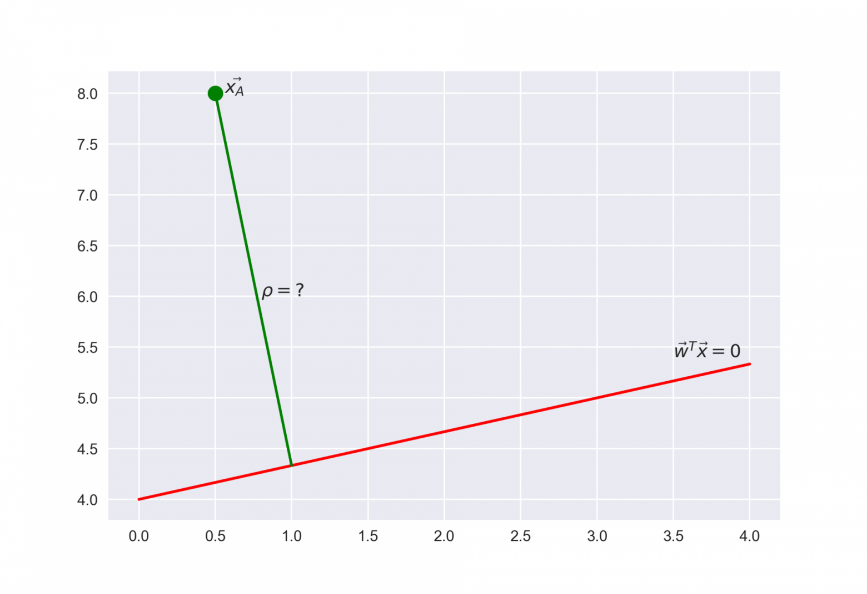
Выражение  называется отступом (margin) классификации на объекте  (не путать с зазором (тоже margin), про который чаще всего говорят в контексте SVM). Если он неотрицателен, модель не ошибается на объекте , если же отрицателен – значит, класс для  спрогнозирован неправильно.

Заметим, что отступ определен для объектов именно обучающей выборки, для которых известны реальные метки целевого класса .

Чтобы понять, почему мы сделали такие выводы, обратимся к геометрической интерпретации линейного классификатора. Подробно про это можно почитать в [материалах](https://github.com/esokolov/ml-course-hse) Евгения Соколова.

Рекомендую решить почти классическую задачу из начального курса линейной алгебры: найти расстояние от точки с радиус-вектором  до плоскости, которая задается уравнением 

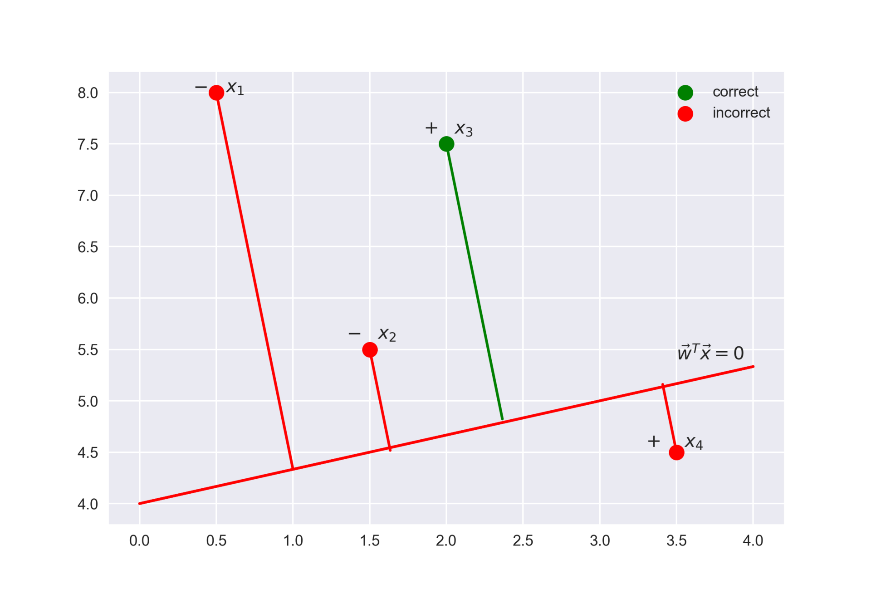
Ответ: 



Когда получим (или посмотрим) ответ, то поймем, что чем больше по модулю выражение , тем дальше точка  находится от плоскости 

Значит, выражение  – это своего рода "уверенность" модели в классификации объекта :

* если отступ большой (по модулю) и положительный, это значит, что метка класса поставлена правильно, а объект находится далеко от разделяющей гиперплоскости (такой объект классифицируется уверенно). На рисунке –  .
* если отступ большой (по модулю) и отрицательный, значит метка класса поставлена неправильно, а объект находится далеко от разделяющей гиперплоскости (скорее всего такой объект – аномалия, например, его метка в обучающей выборке поставлена неправильно). На рисунке – .
* если отступ малый (по модулю), то объект находится близко к разделяющей гиперплоскости, а знак отступа определяет, правильно ли объект классифицирован. На рисунке –  и .



Теперь распишем правдоподобие выборки, а именно, вероятность наблюдать данный вектор  у выборки . Делаем сильное предположение: объекты приходят независимо, из одного распределения (i.i.d. Independent and identically distributed random variables). Тогда



где  – длина выборки  (число строк).

Как водится, возьмем логарифм данного выражения (сумму оптимизировать намного проще, чем произведение):

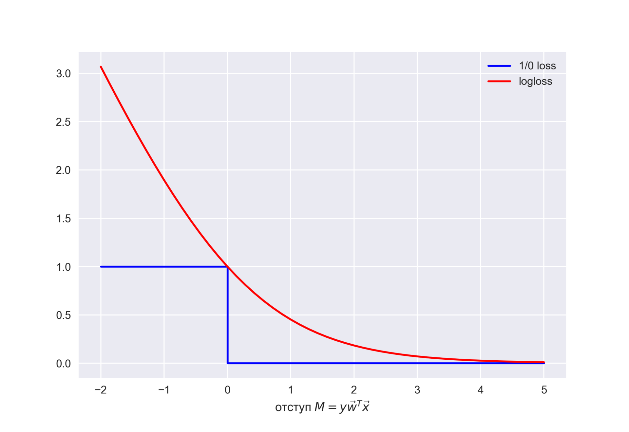


То есть в данном случае принцип максимизации правдоподобия приводит к минимизации выражения



Это логистическая функция потерь, просуммированная по всем объектам обучающей выборки.

Посмотрим на новую фунцию как на функцию от отступа: . Нарисуем ее график, а также график 1/0 функциий потерь (zero-one loss), которая просто штрафует модель на 1 за ошибку на каждом объекте (отступ отрицательный): .



Картинка отражает общую идею, что в задаче классификации, не умея напрямую минимизировать число ошибок (по крайней мере, градиентными методами это не сделать – производная 1/0 функциий потерь в нуле обращается в бесконечность), мы минимизируем некоторую ее верхнюю оценку. В данном случае это логистическая функция потерь (где логарифм двоичный, но это не принципиально), и справедливо



где  – попросту число ошибок логистической регрессии с весами  на выборке .

То есть уменьшая верхнюю оценку  на число ошибок классификации, мы таким образом надеемся уменьшить и само число ошибок.

-*регуляризация логистических потерь*

*L*2-регуляризация логистической регрессии устроена почти так же, как и в случае с гребневой (Ridge регрессией). Вместо функционала  минимизируется следующий:



В случае логистической регрессии принято введение обратного коэффициента регуляризации . И тогда решением задачи будет



Далее рассмотрим пример, позволяющий интуитивно понять один из смыслов регуляризации.

3. НАГЛЯДНЫЙ ПРИМЕР РЕГУЛЯРИЗАЦИИ ЛОГИСТИЧЕСКОЙ РЕГРЕССИИ

Полиномиальные признаки (замена переменных) позволяют линейным моделям строить нелинейные разделяющие поверхности. Покажем это в картинках.

Посмотрим, как регуляризация влияет на качество классификации на наборе данных по тестированию микрочипов из курса Andrew Ng по машинному обучению.

Будем использовать логистическую регрессию с полиномиальными признаками и варьировать параметр регуляризации C.

Сначала посмотрим, как регуляризация влияет на разделяющую границу классификатора, интуитивно распознаем переобучение и недообучение.

Потом численно установим близкий к оптимальному параметр регуляризации с помощью кросс-валидации (cross-validation) и перебора по сетке (GridSearch).

***Подключение библиотек***

from \_\_future\_\_ import division, print\_function

# отключим всякие предупреждения Anaconda

import warnings

warnings.filterwarnings('ignore')

%matplotlib inline

from matplotlib import pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LogisticRegression, LogisticRegressionCV

from sklearn.model\_selection import cross\_val\_score, StratifiedKFold

from sklearn.model\_selection import GridSearchCV

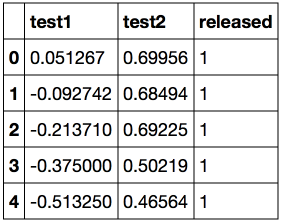
data = pd.read\_csv('../../data/microchip\_tests.txt',

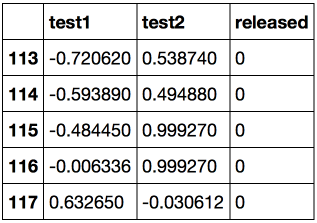
header=None, names = ('test1','test2','released'))

# информация о наборе данных

data.info()

Посмотрим на первые и последние 5 строк.





Сохраним обучающую выборку и метки целевого класса в отдельных массивах NumPy. Отобразим данные. Красный цвет соответствует бракованным чипам, зеленый – нормальным.

***Код***

X = data.ix[:,:2].values

y = data.ix[:,2].values

plt.scatter(X[y == 1, 0], X[y == 1, 1], c='green', label='Выпущен')

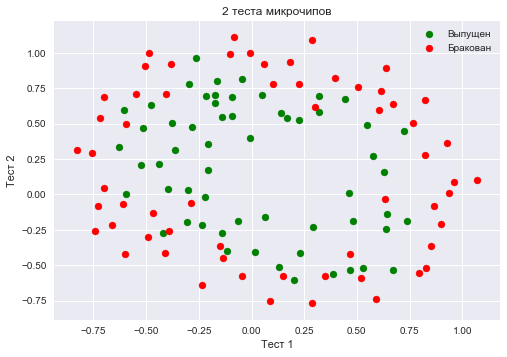
plt.scatter(X[y == 0, 0], X[y == 0, 1], c='red', label='Бракован')

plt.xlabel("Тест 1")

plt.ylabel("Тест 2")

plt.title('2 теста микрочипов')

plt.legend();



Определяем функцию для отображения разделяющей кривой классификатора

***Код***

def plot\_boundary(clf, X, y, grid\_step=.01, poly\_featurizer=None):

x\_min, x\_max = X[:, 0].min() - .1, X[:, 0].max() + .1

y\_min, y\_max = X[:, 1].min() - .1, X[:, 1].max() + .1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, grid\_step),

np.arange(y\_min, y\_max, grid\_step))

# каждой точке в сетке [x\_min, m\_max]x[y\_min, y\_max]

# ставим в соответствие свой цвет

Z = clf.predict(poly\_featurizer.transform(np.c\_[xx.ravel(), yy.ravel()]))

Z = Z.reshape(xx.shape)

plt.contour(xx, yy, Z, cmap=plt.cm.Paired)

Полиномиальными признаками до степени  для двух переменных  и  мы называем следующие:



Например, для  это будут следующие признаки:



Нарисовав треугольник Пифагора, Вы сообразите, сколько таких признаков будет для  и вообще для любого .

Попросту говоря, таких признаков экспоненциально много, и строить, скажем, для 100 признаков полиномиальные степени 10 может оказаться затратно (а более того, и не нужно).

Создадим объект sklearn, который добавит в матрицу  полиномиальные признаки вплоть до степени 7 и обучим логистическую регрессию с параметром регуляризации . Изобразим разделяющую границу.

Также проверим долю правильных ответов классификатора на обучающей выборке. Видим, что регуляризация оказалась слишком сильной, и модель "недообучилась". Доля правильных ответов классификатора на обучающей выборке оказалась равной 0.627.

***Код***

poly = PolynomialFeatures(degree=7)

X\_poly = poly.fit\_transform(X)

C = 1e-2

logit = LogisticRegression(C=C, n\_jobs=-1, random\_state=17)

logit.fit(X\_poly, y)

plot\_boundary(logit, X, y, grid\_step=.01, poly\_featurizer=poly)

plt.scatter(X[y == 1, 0], X[y == 1, 1], c='green', label='Выпущен')

plt.scatter(X[y == 0, 0], X[y == 0, 1], c='red', label='Бракован')

plt.xlabel("Тест 1")

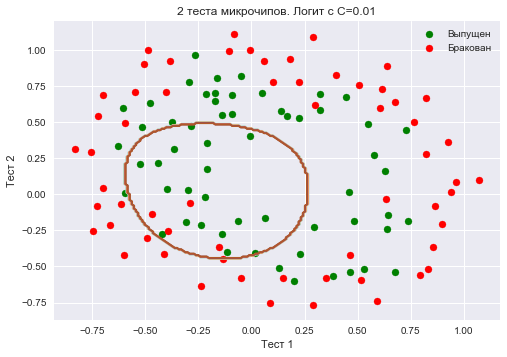
plt.ylabel("Тест 2")

plt.title('2 теста микрочипов. Логит с C=0.01')

plt.legend();

print("Доля правильных ответов классификатора на обучающей выборке:",

round(logit.score(X\_poly, y), 3))



Увеличим *C* до 1. Тем самым мы *ослабляем* регуляризацию, теперь в решении значения весов логистической регрессии могут оказаться больше (по модулю), чем в прошлом случае. Теперь доля правильных ответов классификатора на обучающей выборке – 0.831.

***Код***

C = 1

logit = LogisticRegression(C=C, n\_jobs=-1, random\_state=17)

logit.fit(X\_poly, y)

plot\_boundary(logit, X, y, grid\_step=.005, poly\_featurizer=poly)

plt.scatter(X[y == 1, 0], X[y == 1, 1], c='green', label='Выпущен')

plt.scatter(X[y == 0, 0], X[y == 0, 1], c='red', label='Бракован')

plt.xlabel("Тест 1")

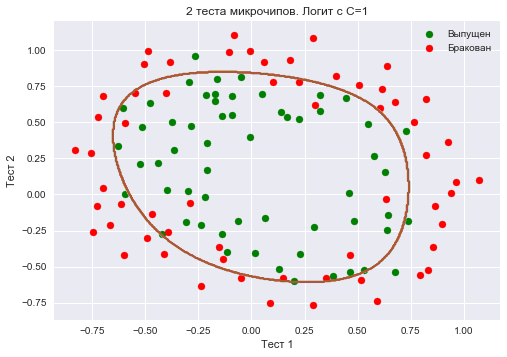
plt.ylabel("Тест 2")

plt.title('2 теста микрочипов. Логит с C=1')

plt.legend();

print("Доля правильных ответов классификатора на обучающей выборке:",

round(logit.score(X\_poly, y), 3))



Еще увеличим *C* – до 10 тысяч. Теперь регуляризации явно недостаточно, и мы наблюдаем переобучение. Можно заметить, что в прошлом случае (при *C* =1 и "гладкой" границе) доля правильных ответов модели на обучающей выборке не намного ниже, чем в 3 случае, зато на новой выборке, можно себе представить, 2 модель сработает намного лучше. Доля правильных ответов классификатора на обучающей выборке – 0.873.

**Код**

C = 1e4

logit = LogisticRegression(C=C, n\_jobs=-1, random\_state=17)

logit.fit(X\_poly, y)

plot\_boundary(logit, X, y, grid\_step=.005, poly\_featurizer=poly)

plt.scatter(X[y == 1, 0], X[y == 1, 1], c='green', label='Выпущен')

plt.scatter(X[y == 0, 0], X[y == 0, 1], c='red', label='Бракован')

plt.xlabel("Тест 1")

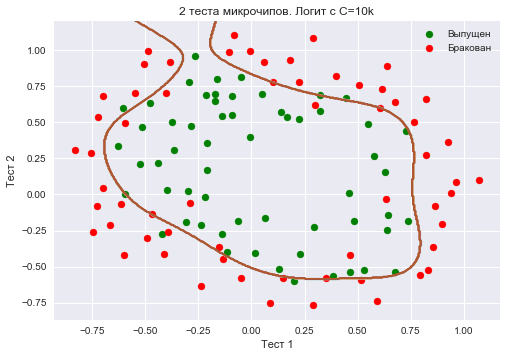
plt.ylabel("Тест 2")

plt.title('2 теста микрочипов. Логит с C=10k')

plt.legend();

print("Доля правильных ответов классификатора на обучающей выборке:",

round(logit.score(X\_poly, y), 3))



Чтоб обсудить результаты, перепишем формулу для функционала, который оптимизируется в логистической регрессии, в таком виде:



где

 – логистическая функция потерь, просуммированная по всей выборке

 – обратный коэффициент регуляризации (тот самый  в sklearn-реализации LogisticRegression)

**Промежуточные выводы:**

чем больше параметр , тем более сложные зависимости в данных может восстанавливать модель (интуитивно  соответствует "сложности" модели (model capacity))

если регуляризация слишком сильная (малые значения ), то решением задачи минимизации логистической функции потерь может оказаться то, когда многие веса занулились или стали слишком малыми. Еще говорят, что модель недостаточно "штрафуется" за ошибки (то есть в функционале  "перевешивает" сумма квадратов весов, а ошибка  может быть относительно большой). В таком случае модель окажется недообученной (1 случай)

наоборот, если регуляризация слишком слабая (большие значения ), то решением задачи оптимизации может стать вектор  с большими по модулю компонентами. В таком случае больший вклад в оптимизируемый функционал  имеет  и, вольно выражаясь, модель слишком "боится" ошибиться на объектах обучающей выборки, поэтому окажется переобученной (3 случай)

то, какое значение  выбрать, сама логистическая регрессия "не поймет" (или еще говорят "не выучит"), то есть это не может быть определено решением оптимизационной задачи, которой является логистическая регрессия (в отличие от весов ). Так же точно, дерево решений не может "само понять", какое ограничение на глубину выбрать (за один процесс обучения). Поэтому  – это гиперпараметр модели, который настраивается на кросс-валидации, как и max\_depth для дерева.

**Настройка параметра регуляризации**

Теперь найдем оптимальное (в данном примере) значение параметра регуляризации . Сделать это можно с помощью LogisticRegressionCV – перебора параметров по сетке с последующей кросс-валидацией. Этот класс создан специально для логистической регрессии (для нее известны эффективные алгоритмы перебора параметров), для произвольной модели мы бы использовали GridSearchCV, RandomizedSearchCV или, например, специальные алгоритмы оптимизации гиперпараметров, реализованные в hyperopt.

***Код***

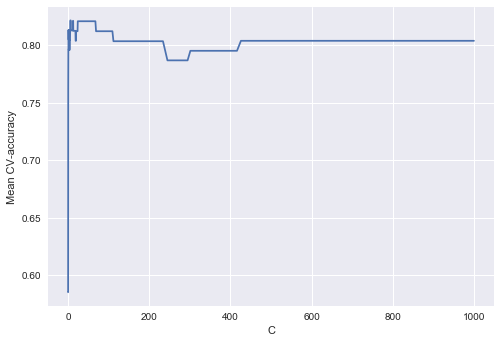
skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=17)

c\_values = np.logspace(-2, 3, 500)

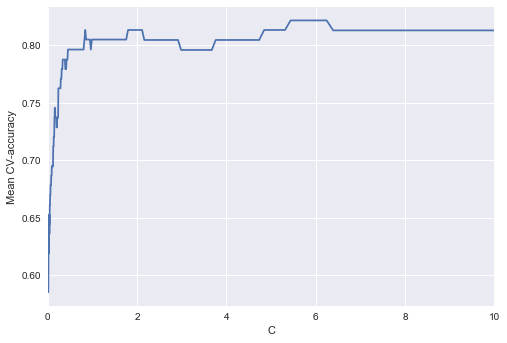
logit\_searcher = LogisticRegressionCV(Cs=c\_values, cv=skf, verbose=1, n\_jobs=-1)

logit\_searcher.fit(X\_poly, y)

Посмотрим, как качество модели (доля правильных ответов на обучающей и валидационной выборках) меняется при изменении гиперпараметра .



Выделим участок с "лучшими" значениями *C*.

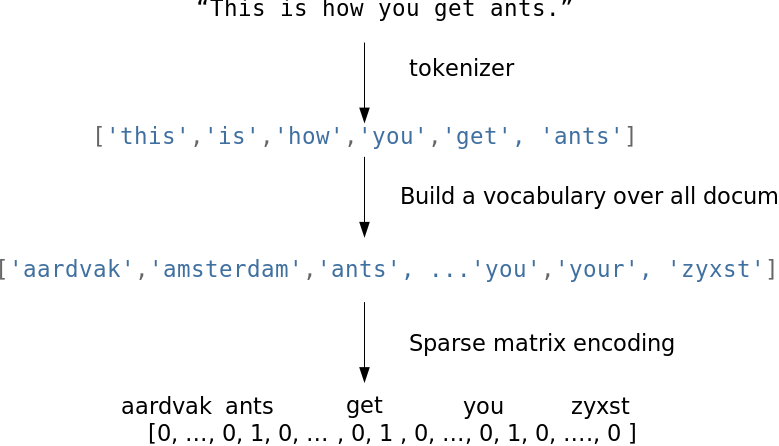


Как мы помним, такие кривые называются *валидационными*, раньше мы их строили вручную, но в sklearn для них их построения есть специальные методы, которые мы тоже сейчас будем использовать.

**4. Где логистическая регрессия хороша и где не очень**

**Анализ отзывов IMDB к фильмам**

Будем решать задачу бинарной классификации отзывов IMDB к фильмам. Имеется обучающая выборка с размеченными отзывами, по 12500 отзывов известно, что они хорошие, еще про 12500 – что они плохие. Здесь уже не так просто сразу приступить к машинному обучению, потому что готовой матрицы нет – ее надо приготовить. Будем использовать самый простой подход – мешок слов ("Bag of words"). При таком подходе признаками отзыва будут индикаторы наличия в нем каждого слова из всего корпуса, где корпус – это множество всех отзывов. Идея иллюстрируется картинкой



***Импорт библиотек и загрузка данных***

from \_\_future\_\_ import division, print\_function

# отключим всякие предупреждения Anaconda

import warnings

warnings.filterwarnings('ignore')

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

import numpy as np

from sklearn.datasets import load\_files

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer, TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import LinearSVC

Загрузим данные [отсюда](http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz) (краткое описание — [тут](http://ai.stanford.edu/~amaas/data/sentiment/)). В обучающей и тестовой выборках по 12500 тысяч хороших и плохих отзывов к фильмам.

reviews\_train = load\_files("YOUR PATH")

text\_train, y\_train = reviews\_train.data, reviews\_train.target

print("Number of documents in training data: %d" % len(text\_train))

print(np.bincount(y\_train))

# поменяйте путь к файлу

reviews\_test = load\_files("YOUR PATH")

text\_test, y\_test = reviews\_test.data, reviews\_test.target

print("Number of documents in test data: %d" % len(text\_test))

print(np.bincount(y\_test))

***Пример плохого отзыва:***

'Words can\'t describe how bad this movie is. I can\'t explain it by writing only. You have too see it for yourself to get at grip of how horrible a movie really can be. Not that I recommend you to do that. There are so many clich\xc3\xa9s, mistakes (and all other negative things you can imagine) here that will just make you cry. To start with the technical first, there are a LOT of mistakes regarding the airplane. I won\'t list them here, but just mention the coloring of the plane. They didn\'t even manage to show an airliner in the colors of a fictional airline, but instead used a 747 painted in the original Boeing livery. Very bad. The plot is stupid and has been done many times before, only much, much better. There are so many ridiculous moments here that i lost count of it really early. Also, I was on the bad guys\' side all the time in the movie, because the good guys were so stupid. "Executive Decision" should without a doubt be you\'re choice over this one, even the "Turbulence"-movies are better. In fact, every other movie in the world is better than this one.'

***Пример хорошего отзыва:***

'Everyone plays their part pretty well in this "little nice movie". Belushi gets the chance to live part of his life differently, but ends up realizing that what he had was going to be just as good or maybe even better. The movie shows us that we ought to take advantage of the opportunities we have, not the ones we do not or cannot have. If U can get this movie on video for around $10, it\xc2\xb4d be an investment!'

**Простой подсчет слов**

Составим словарь всех слов с помощью CountVectorizer. Всего в выборке 74849 уникальных слов. Если посмотреть на примеры полученных "слов" (лучше их называть токенами), то можно увидеть, что многие важные этапы обработки текста мы тут пропустили (автоматическая обработка текстов – это могло бы быть темой отдельной серии статей).

***Код***

cv = CountVectorizer()

cv.fit(text\_train)

print(len(cv.vocabulary\_)) #74849

print(cv.get\_feature\_names()[:50])

print(cv.get\_feature\_names()[50000:50050])

['00', '000', '0000000000001', '00001', '00015', '000s', '001', '003830', '006', '007', '0079', '0080', '0083', '0093638', '00am', '00pm', '00s', '01', '01pm', '02', '020410', '029', '03', '04', '041', '05', '050', '06', '06th', '07', '08', '087', '089', '08th', '09', '0f', '0ne', '0r', '0s', '10', '100', '1000', '1000000', '10000000000000', '1000lb', '1000s', '1001', '100b', '100k', '100m']  
['pincher', 'pinchers', 'pinches', 'pinching', 'pinchot', 'pinciotti', 'pine', 'pineal', 'pineapple', 'pineapples', 'pines', 'pinet', 'pinetrees', 'pineyro', 'pinfall', 'pinfold', 'ping', 'pingo', 'pinhead', 'pinheads', 'pinho', 'pining', 'pinjar', 'pink', 'pinkerton', 'pinkett', 'pinkie', 'pinkins', 'pinkish', 'pinko', 'pinks', 'pinku', 'pinkus', 'pinky', 'pinnacle', 'pinnacles', 'pinned', 'pinning', 'pinnings', 'pinnochio', 'pinnocioesque', 'pino', 'pinocchio', 'pinochet', 'pinochets', 'pinoy', 'pinpoint', 'pinpoints', 'pins', 'pinsent']

Закодируем предложения из текстов обучающей выборки индексами входящих слов. Используем разреженный формат. Преобразуем так же тестовую выборку.

X\_train = cv.transform(text\_train)

X\_test = cv.transform(text\_test)

Обучим логистическую регрессию и посмотрим на доли правильных ответов на обучающей и тестовой выборках. Получается, на тестовой выборке мы правильно угадываем тональность примерно 86.7% отзывов.

***Код***

%%time

logit = LogisticRegression(n\_jobs=-1, random\_state=7)

logit.fit(X\_train, y\_train)

print(round(logit.score(X\_train, y\_train), 3), round(logit.score(X\_test, y\_test), 3))

Коэффициенты модели можно красиво отобразить.

***Код визуализации коэффициентов модели***

def visualize\_coefficients(classifier, feature\_names, n\_top\_features=25):

# get coefficients with large absolute values

coef = classifier.coef\_.ravel()

positive\_coefficients = np.argsort(coef)[-n\_top\_features:]

negative\_coefficients = np.argsort(coef)[:n\_top\_features]

interesting\_coefficients = np.hstack([negative\_coefficients, positive\_coefficients])

# plot them

plt.figure(figsize=(15, 5))

colors = ["red" if c < 0 else "blue" for c in coef[interesting\_coefficients]]

plt.bar(np.arange(2 \* n\_top\_features), coef[interesting\_coefficients], color=colors)

feature\_names = np.array(feature\_names)

plt.xticks(np.arange(1, 1 + 2 \* n\_top\_features), feature\_names[interesting\_coefficients], rotation=60, ha="right");

def plot\_grid\_scores(grid, param\_name):

plt.plot(grid.param\_grid[param\_name], grid.cv\_results\_['mean\_train\_score'],

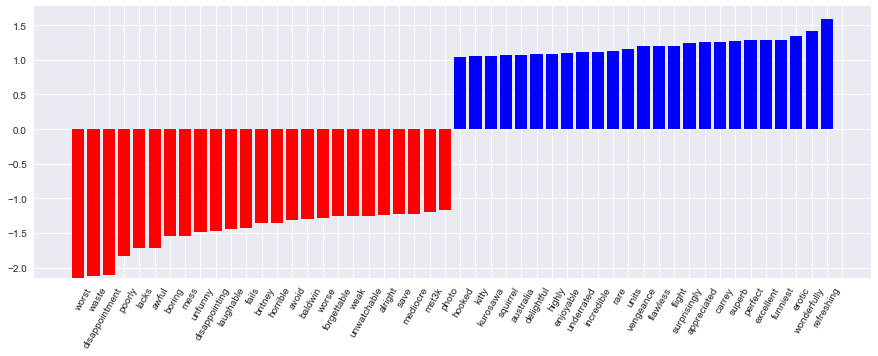
color='green', label='train')

plt.plot(grid.param\_grid[param\_name], grid.cv\_results\_['mean\_test\_score'],

color='red', label='test')

plt.legend();

visualize\_coefficients(logit, cv.get\_feature\_names())



Подберем коэффициент регуляризации для логистической регрессии. Используем sklearn.pipeline, поскольку CountVectorizer правильно применять только на тех данных, на которых в текущий момент обучается модель (чтоб не "подсматривать" в тестовую выборку и не считать по ней частоты вхождения слов). В данном случае pipeline задает последовательность действий: применить CountVectorizer, затем обучить логистическую регрессию. Так мы поднимаем долю правильных ответов до 88.5% на кросс-валидации и 87.9% – на отложенной выборке.

***Код***

from sklearn.pipeline import make\_pipeline

text\_pipe\_logit = make\_pipeline(CountVectorizer(),

LogisticRegression(n\_jobs=-1, random\_state=7))

text\_pipe\_logit.fit(text\_train, y\_train)

print(text\_pipe\_logit.score(text\_test, y\_test))

from sklearn.model\_selection import GridSearchCV

param\_grid\_logit = {'logisticregression\_\_C': np.logspace(-5, 0, 6)}

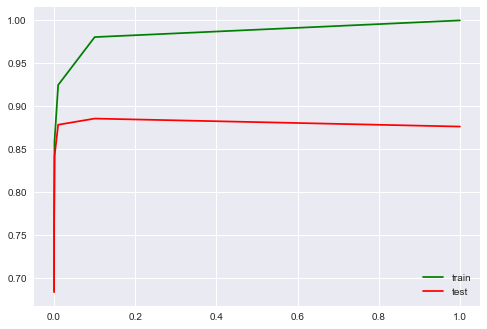
grid\_logit = GridSearchCV(text\_pipe\_logit, param\_grid\_logit, cv=3, n\_jobs=-1)

grid\_logit.fit(text\_train, y\_train)

grid\_logit.best\_params\_, grid\_logit.best\_score\_

plot\_grid\_scores(grid\_logit, 'logisticregression\_\_C')

grid\_logit.score(text\_test, y\_test)



Теперь то же самое, но со случайным лесом. Видим, что с логистической регрессией мы достигаем большей доли правильных ответов меньшими усилиями. Лес работает дольше, на отложенной выборке 85.5% правильных ответов.

***Код для обучения случайного леса***

from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n\_estimators=200, n\_jobs=-1, random\_state=17)

forest.fit(X\_train, y\_train)

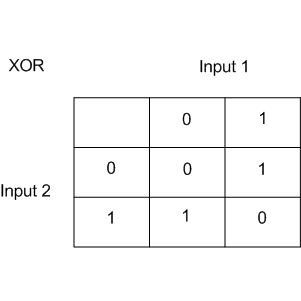
print(round(forest.score(X\_test, y\_test), 3))

**XOR-проблема**

Теперь рассмотрим пример, где линейные модели справляются хуже.

Линейные методы классификации строят все же очень простую разделяющую поверхность – гиперплоскость. Самый известный игрушечный пример, в котором классы нельзя без ошибок поделить гиперплоскостью (то есть прямой, если это 2D), получил имя "the XOR problem".

XOR – это "исключающее ИЛИ", булева функция со следующей таблицей истинности:



XOR дал имя простой задаче бинарной классификации, в которой классы представлены вытянутыми по диагоналям и пересекающимися облаками точек.

***Код, рисующий следующие 3 картинки***

# порождаем данные

rng = np.random.RandomState(0)

X = rng.randn(200, 2)

y = np.logical\_xor(X[:, 0] > 0, X[:, 1] > 0)

plt.scatter(X[:, 0], X[:, 1], s=30, c=y, cmap=plt.cm.Paired);

def plot\_boundary(clf, X, y, plot\_title):

xx, yy = np.meshgrid(np.linspace(-3, 3, 50),

np.linspace(-3, 3, 50))

clf.fit(X, y)

# plot the decision function for each datapoint on the grid

Z = clf.predict\_proba(np.vstack((xx.ravel(), yy.ravel())).T)[:, 1]

Z = Z.reshape(xx.shape)

image = plt.imshow(Z, interpolation='nearest',

extent=(xx.min(), xx.max(), yy.min(), yy.max()),

aspect='auto', origin='lower', cmap=plt.cm.PuOr\_r)

contours = plt.contour(xx, yy, Z, levels=[0], linewidths=2,

linetypes='--')

plt.scatter(X[:, 0], X[:, 1], s=30, c=y, cmap=plt.cm.Paired)

plt.xticks(())

plt.yticks(())

plt.xlabel(r'$<!-- math>$inline$x\_1$inline$</math -->$')

plt.ylabel(r'$<!-- math>$inline$x\_2$inline$</math -->$')

plt.axis([-3, 3, -3, 3])

plt.colorbar(image)

plt.title(plot\_title, fontsize=12);

plot\_boundary(LogisticRegression(), X, y,

"Logistic Regression, XOR problem")

from sklearn.preprocessing import PolynomialFeatures

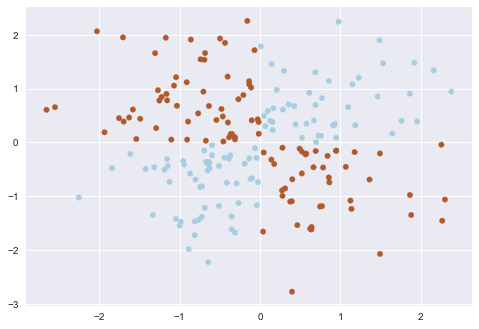
from sklearn.pipeline import Pipeline

logit\_pipe = Pipeline([('poly', PolynomialFeatures(degree=2)),

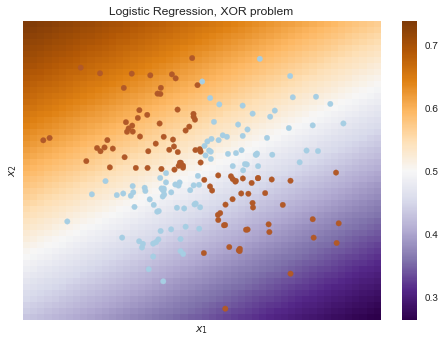
('logit', LogisticRegression())])

plot\_boundary(logit\_pipe, X, y,

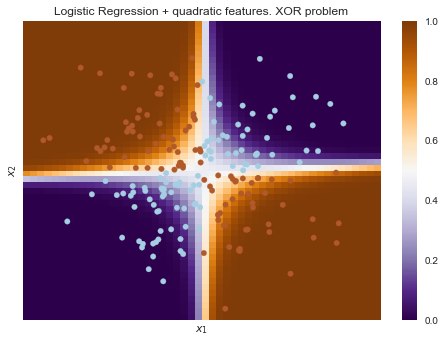
"Logistic Regression + quadratic features. XOR problem")



Очевидно, нельзя провести прямую так, чтобы без ошибок отделить один класс от другого. Поэтому логистическая регрессия плохо справляется с такой задачей.



А вот если на вход подать полиномиальные признаки, в данном случае до 2 степени, то проблема решается.



Здесь логистическая регрессия все равно строила гиперплоскость, но в 6-мерном пространстве признаков  и .. В проекции на исходное пространство признаков  граница получилась нелинейной.

На практике полиномиальные признаки действительно помогают, но строить их явно – вычислительно неэффективно. Гораздо быстрее работает SVM с ядровым трюком. При таком подходе в пространстве высокой размерности считается только расстояние между объектами (задаваемое функцией-ядром), а явно плодить комбинаторно большое число признаков не приходится. Про это подробно можно почитать в [курсе](https://github.com/esokolov/ml-course-hse) Евгения Соколова (математика уже серьезная).

**5. Кривые валидации и обучения**

Мы уже получили представление о проверке модели, кросс-валидации и регуляризации.  
Теперь рассмотрим главный вопрос:

Если качество модели нас не устраивает, что делать?

* Сделать модель сложнее или упростить?
* Добавить больше признаков?
* Или нам просто нужно больше данных для обучения?

Ответы на данные вопросы не всегда лежат на поверхности. В частности, иногда использование более сложной модели приведет к ухудшению показателей. Либо добавление наблюдений не приведет к ощутимым изменениям. Способность принять правильное решение и выбрать правильный способ улучшения модели, собственно говоря, и отличает хорошего специалиста от плохого.

Будем работать со знакомыми данными по оттоку клиентов телеком-оператора.

***Импорт библиотек и чтение данных***

from \_\_future\_\_ import division, print\_function

# отключим всякие предупреждения Anaconda

import warnings

warnings.filterwarnings('ignore')

%matplotlib inline

from matplotlib import pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

from sklearn.preprocessing import PolynomialFeatures

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression, LogisticRegressionCV, SGDClassifier

from sklearn.model\_selection import validation\_curve

data = pd.read\_csv('../../data/telecom\_churn.csv').drop('State', axis=1)

data['International plan'] = data['International plan'].map({'Yes': 1, 'No': 0})

data['Voice mail plan'] = data['Voice mail plan'].map({'Yes': 1, 'No': 0})

y = data['Churn'].astype('int').values

X = data.drop('Churn', axis=1).values

Логистическую регрессию будем обучать стохастическим градиентным спуском. Пока объясним это тем, что так быстрее, но далее в программе у нас отдельная статья про это дело. Построим валидационные кривые, показывающие, как качество (ROC AUC) на обучающей и проверочной выборке меняется с изменением параметра регуляризации.

***Код***

alphas = np.logspace(-2, 0, 20)

sgd\_logit = SGDClassifier(loss='log', n\_jobs=-1, random\_state=17)

logit\_pipe = Pipeline([('scaler', StandardScaler()), ('poly', PolynomialFeatures(degree=2)),

('sgd\_logit', sgd\_logit)])

val\_train, val\_test = validation\_curve(logit\_pipe, X, y,

'sgd\_logit\_\_alpha', alphas, cv=5,

scoring='roc\_auc')

def plot\_with\_err(x, data, \*\*kwargs):

mu, std = data.mean(1), data.std(1)

lines = plt.plot(x, mu, '-', \*\*kwargs)

plt.fill\_between(x, mu - std, mu + std, edgecolor='none',

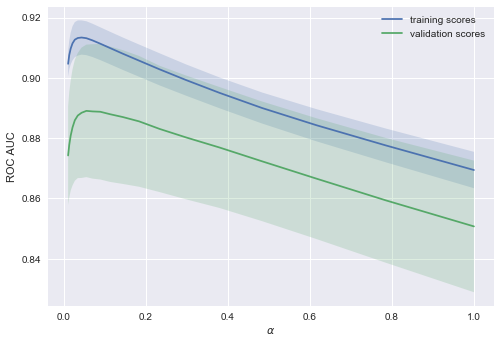
facecolor=lines[0].get\_color(), alpha=0.2)

plot\_with\_err(alphas, val\_train, label='training scores')

plot\_with\_err(alphas, val\_test, label='validation scores')

plt.xlabel(r'$\alpha$'); plt.ylabel('ROC AUC')

plt.legend();



Тенденция видна сразу, и она очень часто встречается.

* Для простых моделей тренировочная и валидационная ошибка находятся где-то рядом, и они велики. Это говорит о том, что модель **недообучилась**: то есть она не имеет достаточное кол-во параметров.
* Для сильно усложненных моделей тренировочная и валидационная ошибки значительно отличаются. Это можно объяснить **переобучением**: когда параметров слишком много либо не хватает регуляризации, алгоритм может "отвлекаться" на шум в данных и упускать основной тренд.

**Сколько нужно данных?**

Известно, что чем больше данных использует модель, тем лучше. Но как нам понять в конкретной ситуации, помогут ли новые данные? Скажем, целесообразно ли нам потратить N\$ на труд асессоров, чтобы увеличить выборку вдвое?

Поскольку новых данных пока может и не быть, разумно поварьировать размер имеющейся обучающей выборки и посмотреть, как качество решения задачи зависит от объема данных, на котором мы обучали модель. Так получаются **кривые обучения** (**learning curves**).

Идея простая: мы отображаем ошибку как функцию от количества примеров, используемых для обучения. При этом параметры модели фиксируются заранее.

Давайте посмотрим, что мы получим для линейной модели. Коэффициент регуляризации выставим большим.

***Код***

from sklearn.model\_selection import learning\_curve

def plot\_learning\_curve(degree=2, alpha=0.01):

train\_sizes = np.linspace(0.05, 1, 20)

logit\_pipe = Pipeline([('scaler', StandardScaler()), ('poly', PolynomialFeatures(degree=degree)),

('sgd\_logit', SGDClassifier(n\_jobs=-1, random\_state=17, alpha=alpha))])

N\_train, val\_train, val\_test = learning\_curve(logit\_pipe,

X, y, train\_sizes=train\_sizes, cv=5,

scoring='roc\_auc')

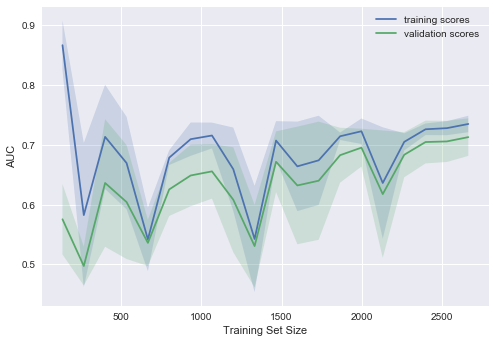
plot\_with\_err(N\_train, val\_train, label='training scores')

plot\_with\_err(N\_train, val\_test, label='validation scores')

plt.xlabel('Training Set Size'); plt.ylabel('AUC')

plt.legend()

plot\_learning\_curve(degree=2, alpha=10)



Типичная ситуация: для небольшого объема данных ошибки на обучающей выборке и в процессе кросс-валидации довольно сильно отличаются, что указывает на переобучение. Для той же модели, но с большим объемом данных ошибки "сходятся", что указывается на недообучение.

Если добавить еще данные, ошибка на обучающей выборке не будет расти, но с другой стороны, ошибка на тестовых данных не будет уменьшаться.

Получается, ошибки "сошлись", и добавление новых данных не поможет. Собственно, это случай – самый интересный для бизнеса. Возможна ситуация, когда мы увеличиваем выборку в 10 раз. Но если не менять сложность модели, это может и не помочь. То есть стратегия "настроил один раз – дальше использую 10 раз" может и не работать.

Что будет, если изменить коэффициент регуляризации (уменьшить до 0.05)?

Видим хорошую тенденцию – кривые постепенно сходятся, и если дальше двигаться направо (добавлять в модель данные), можно еще повысить качество на валидации.



А если усложнить модель ещё больше ()?

Проявляется переобучение – AUC падает как на обучении, так и на валидации.



Строя подобные кривые, можно понять, в какую сторону двигаться, и как правильно настроить сложность модели на новых данных.

***Выводы по кривым валидации и обучения***

* Ошибка на обучающей выборке сама по себе ничего не говорит о качестве модели
* Кросс-валидационная ошибка показывает, насколько хорошо модель подстраивается под данные (имеющийся тренд в данных), сохраняя при этом способность обобщения на новые данные
* **Валидационная кривая** представляет собой график, показывающий результат на тренировочной и валидационной выборке в зависимости от **сложности модели**:
* если две кривые располагаются близко, и обе ошибки велики, — это признак *недообучения*
* если две кривые далеко друг от друга, — это показатель *переобучения*
* **Кривая обучения** — это график, показывающий результаты на валидации и тренировочной подвыборке в зависимости от количества наблюдений:
* если кривые сошлись друг к другу, добавление новых данных не поможет – надо менять сложность модели
* если кривые еще не сошлись, добавление новых данных может улучшить результат.

**6. Плюсы и минусы линейных моделей в задачах машинного обучения**

***Плюсы:***

* Хорошо изучены
* Очень быстрые, могут работать на очень больших выборках
* Практически вне конкуренции, когда признаков очень много (от сотен тысяч и более), и они разреженные (хотя есть еще факторизационные машины)
* Коэффициенты перед признаками могут интерпретироваться (при условии, что признаки масштабированы) – в линейной регрессии как частные производные зависимой переменной от признаков, в логистической – как изменение шансов на отнесение к одному из классов в  раз при изменении признака  на 1 ед., подробнее [тут](https://www.unm.edu/~schrader/biostat/bio2/Spr06/lec11.pdf) (eng)
* Логистическая регрессия выдает вероятности отнесения к разным классам (это очень ценится, например, в кредитном скоринге)
* Модель может строить и нелинейную границу, если на вход подать полиномиальные признаки

***Минусы:***

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

В качестве закрепления изученного материала предлагаем следующее задание: разобраться с тем, как работает TfidfVectorizer и DictVectorizer, обучить и настроить модель линейной регрессии Ridge на данных о публикациях на Хабрахабре и воспроизвести бенчмарк в соревновании (https://www.kaggle.com/c/howpop-habrahabr-favs-lognorm).