Лабораторная работа №2

Задачи регрессии и классификации. Линейные модели: линейная и логистическая регрессии

Машинное обучение (ML) — это область искусственного интеллекта, характерной чертой которой является не прямое решение задачи, а обучение за счет применения решений множества сходных задач. Для построения таких методов используются средства математической статистики, численных методов, математического анализа, методов оптимизации, теории вероятностей, теории графов, различные техники работы с данными в цифровой форме. Модель машинного обучения учится выявлять закономерности в данных и делать верные прогнозы. В зависимости от наличия размеченных данных и способа обучения моделей выделяют несколько типов машинного обучения:

- Обучение с учителем (Supervised Learning)
- Обучение без учителя (Unsupervised Learning)
- Обучение с частичным привлечением учителя (Semi-supervised Learning)
- Обучение с подкреплением (Reinforcement Learning)

Каждый из этих подходов используется для решения различных задач и имеет свои особенности.

Обучение с учителем (Supervised Learning)

В этом типе обучения модели подаются **размеченные данные**: для каждого входного примера есть соответствующий ему правильный выход (метка класса или числовое значение). Модель анализирует примеры и учится предсказывать правильные ответы, для входных образцов имеющих схожее распределение признаков, что и обучающая выборка.

Решаемые задачи:

- **Регрессия** — предсказание числовых значений (например, прогнозирование цен на недвижимость, температуру). Используемые алгоритмы:
    - Линейная и полиномиальная регрессии
    - Деревья решений

    - Нейронные сети

- **Классификация** — определение класса объекта. Используемые алгоритмы:
    - Логистическая регрессия
    - Деревья решений
    - Метод случайного леса
    - Методы на основе формуле Байеса
    - Метод опорных векторов
    - Нейронные сети
- **Ранжирование** — присвоение рангов объектам в системе, по их признакам.

Используемые библиотеки

Разделение выборки

Чаще всего данные представлены в виде одного большого файла, содержащего все данные. Однако для предотвращения переобучения модели датасет разбивают на обучающую, тестовую и валидационную выборки:

- ***Обучающая выборка*** составляет обычно 60% от общей выборки и используется для обучения модели, что делает модель зависимой от этих данных и “предвзятой” к ним.

- ***Тестовая выборка*** — 20-25% общей выборки. Состоит из данных, которые отличны от обучающей выборки, но статистические характеристики этих двух выборок практически одинаковы. Тестовая выборка выполняет задачу проверки работы обученной модели для выявления факта недообученности или переобученности, о чем будет написано далее.
- ***Валидационная выборка*** — 15-20% общей выборки. Применяется во время обучения для проверки качества процесса обучения. С её помощью мы отвечаем на вопрос: “В правильном ли направлении идет обучение? Улучшается ли работа модели для данных из обучающей выборки?” На основе полученных оценок корректируются ***гиперпараметры модели*** — параметры модели, которые контролирую процесс обучения — его скорость, точность и т.д., в простых моделях выбираются до начала обучения и не изменяются до его окончания.

В Python для этой задачи можно использовать инструменты из библиотеки scikit-learn:

```
from sklearn.model_selection import train_test_split

X = df.drop(['Weight'], axis=1)
y = df['Weight']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.4, random_state=42)
```

У функции два выходных значения — обучающая и тестовая выборки, полученные из поданного в качестве первого параметра датасета. Параметр *test_size* определяет долю данных, которые попадут в тестовую выборку; *random_state* используется в качестве начального числа для генератора псевдослучайных чисел на основе которого данные будут перемешаны, это способствует избежанию переобучения модели.

Задачи решаемые в обучении с учителем

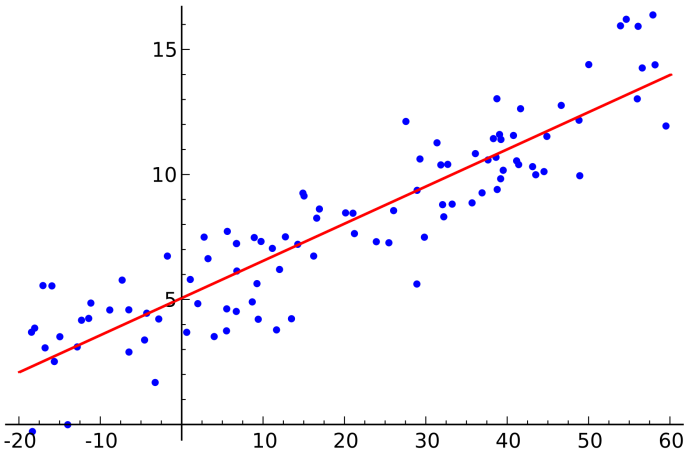
Данные, с которыми работает модель машинного обучения, могут быть двух видов: дискретные и непрерывные. Соответственно и выход модели может принимать вид как дискретных, так и непрерывных значений (или в виде вектора таких значений). Из этого для обучения с учителем вводятся два основных типа решаемых задач: если модель обучается находить непрерывные значения для входного вектора признаков — то это задача регрессии; если же обучается определять дискретный выход (класс, метку, категорию и т.п.) — это задача классификации. Остальные задачи решаемые в обучении с учителем “вытекают” из двух вышеперечисленных.

Задача регрессии

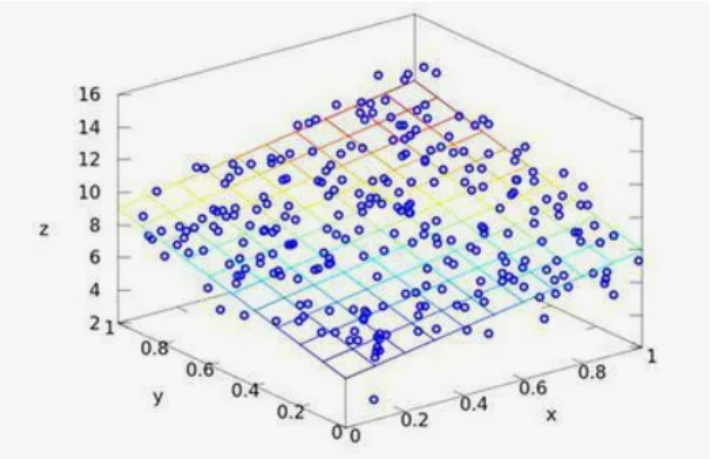
Первой рассмотрим задачу регрессии. Данные в ней являются размеченными, то есть в обучающей выборке уже есть набор векторов признаков X и набор соответствующих значений Y для каждого вектора из X. Таким образом, на основе этих данных, настраиваемая модель должна научиться определять непрерывное значение y для входного вектора x.

Задача регрессии очень похожа на задачу аппроксимации функции. Однако при аппроксимации целью является нахождение менее затратной по вычислительным ресурсам функции, которая с некоторой допустимой ошибкой повторяет значения более сложной известной заранее функции на некотором промежутке. В регрессии же заранее неизвестно какой функции соответствуют данные, а так как зачастую данные являются экспериментальным, то в них также присутствует и некая погрешность в виде шума. Поэтому в основе своей модель МО, решающая задачу регрессии, представляет собой настраиваемую функцию в гиперпространстве, называемая ***гипотезой***, которая пытается повторить некоторую неизвестную функцию, по которой распределены значения из обучающей выборки.

Графически — искомая гипотеза строит гиперплоскость, которая бы лучше всего описывала данные. Например, когда имеется один входной признак и один выходной результат, то данные, приведенные на графике ниже, можно описать простой линейной функцией. Отклонения же от предполагаемого распределения можно объяснить наличием шума при измерениях:



Для двух входных признаков и одного выходного результата гипотеза определяет плоскость:



При увеличении количества признаков задача перейдет в гиперпространство, а гипотеза будет описывать гиперплоскость.

Так как же найти подходящую функцию? Как было сказано ранее, гипотеза модели, как и сама модель, является “настраиваемой”: модель имеет вектор параметров w, который в данном случае определяет коэффициенты искомой функции. Вектор w также называют ***весами модели***. Исходя из этого ***линейную*** модель можно представить в виде

$$a(w, x) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n,$$

где n — количество признаков, w_0 — смещение.

Данная модель ***линейная*** в том смысле, что при вычислении выходного значения применяются только ***линейные операции***.

Можно также встретить название ***множественная линейная (мультилинейная) регрессия*** для такой модели, так как она определена для вектора x, имеющего более одного признака. Линейной же тогда называют самую простейшую модель вида

$$a(w, x) = w_0 + w_1 \cdot x_1,$$

или же знакомую функцию

$$y = kx + b.$$

В данном случае $k = w_1$, а смещение $b=w_0$.

Таким образом задача сводится к нахождению вектора параметров w, который позволял бы моделировать данные близкие к обучающей выборке.

Однако простой метод перебора значений коэффициентов в данном случае не поможет.

Начать стоит с введения некой функции, которая бы определяла насколько сильно значение, полученное моделью с текущим вектором параметров w для i-го входного вектора x^i, отличается от соответствующего значения y^i из обучающей выборки. Самым простым её определением может стать разница между значением полученным из модели и значением из выборки (назовем это ***ошибкой модели*** для i-го наблюдения)

$$a(w, x^i) - y^i$$

Однако такой вид ошибки не очень удобен в качестве оптимизируемой величины. Лучше бы подошла функция, значение которой уменьшалось бы при уменьшении ошибки и увеличивалось при росте ошибки. Две такие функции представлены ниже

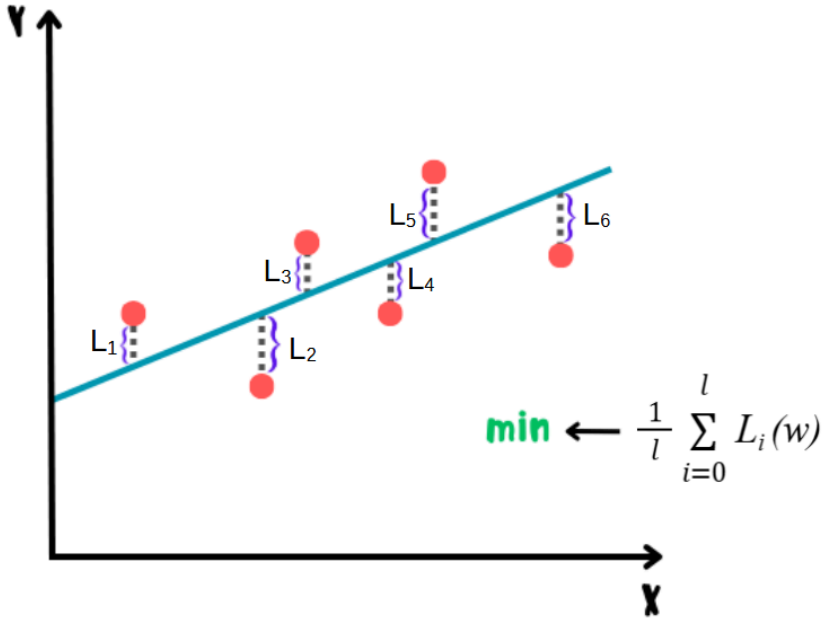
$$L_i(w) = |a(w, x^i) - y^i| \text{ — абсолютная ошибка}$$

$$L_i(w) = (a(w, x^i) - y^i)^2 \text{— квадратичная ошибка}$$

Данные функции называются ***функциями потерь (loss function)***. Но такая функция отражает ошибку модели только относительно одного наблюдения из обучающей выборки. Чтобы получить среднюю ошибку по всей обучающей выборке вводят ***функцию стоимости*** (также иногда — ***средний эмпирический риск***)

$$Q(w) = \frac{1}{l} \sum_{i=0}^{l} L_i(w) \rightarrow min$$

где l — размер обучающей выборки. Нахождение минимума такой функции приведет к получению вектора w, при котором значения модели $a(w, x)$ будут максимально близкими к значениям обучающей выборки. Графически это можно изобразить следующим образом:



В итоге задача обучения регрессионной модели сводится к задаче оптимизации *функции стоимости*, то есть поиска минимума функции *Q(w)*, зависящей от искомого нами вектора параметров *w*.

Стандартным способом решения задачи оптимизации является ***метод градиентного спуска***. Он имеет множество эвристических модификаций, которые позволяют быстрее находить минимум функции, не застревать в локальных минимумах и т.д.

Для решения регрессии в Python используйте ***LinearRegression*** из библиотеки scikit-learn.

```
from sklearn.linear_model import LinearRegression
linear_model = LinearRegression()
```

Как и для *MinMaxScaler* и *StandartScaler*, внутри полученного объекта *linear_model* храниться математическая модель — гипотеза с изначально не рассчитанными весами. Для обучения модели применяется метод *fit()*, в него передается матрица обучающих наблюдений *X* и соответствующие им выходные результаты *Y*

```
linear_model.fit(X_train, y_train)
```

Данная модель обучается *методом наименьших квадратов*. Предсказать же значения обученной моделью можно с помощью *predict()*, передав матрицу X для тестовой выборки

```
y_pred_test = linear_model.predict(X_test)
```

Получить больше примеров и информации о реализации линейных моделей из пакета *sklearn.linear_model* в *scikit-learn* можно здесь.

Оценка регрессионной модели

Для оценки качества модели, решающей задачу регрессии, можно применить функции стоимости, которые при подборе параметров модели минимизируются:

- MSE (Mean Square Error, среднеквадратичная ошибка):

$$MSE = \frac{1}{l} \sum_{i=0}^{l} (a(w, x^i) - y^i)^2$$

```
from sklearn.metrics import mean_squared_error
MSE = mean_squared_error(y_test, y_pred_test)
```

- RMSE (Root MSE, корень среднеквадратичной ошибки) — приводит MSE к той же размерности, что и исходные данные:

$$RMSE = \sqrt{MSE}$$

```
from sklearn.metrics import root_mean_squared_error
RMSE = root_mean_squared_error(y_test, y_pred_test)
```

- MAE (Mean Absolute Error, средняя абсолютная ошибка)

$$MAE = \frac{1}{l} \sum_{i=0}^{l} |a(w, x^i) - y^i|$$

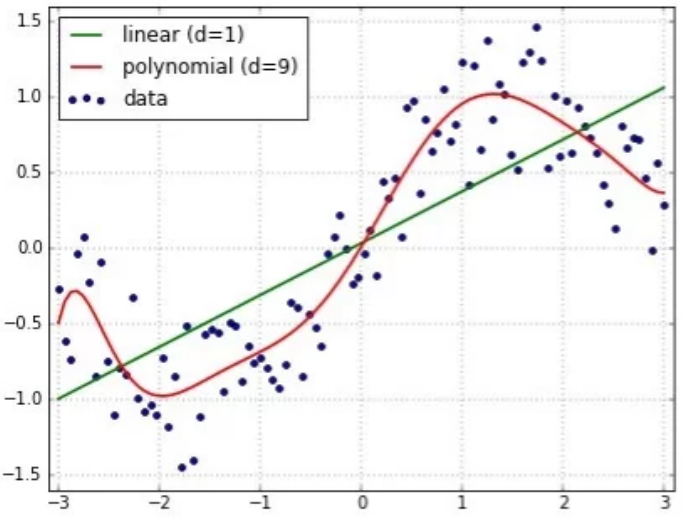
```
from sklearn.metrics import mean_absolute_error
MAE = mean_absolute_error(y_test, y_pred_test)
```

Полиномиальная регрессия

Стоит также упомянуть о ***полиномиальной регрессии***. Это форма регрессионного анализа, при которой взаимосвязь между независимой переменной x и зависимой переменной y моделируется как полином от x. То есть гипотеза модели с одним входным признаком x примет вид

$$a(w, x) = w_0 + w_1 \cdot x + w_2 \cdot x^2 + \ldots + w_n \cdot x^n$$

Ее применяют, когда данные выборки нельзя явно описать линейным образом, как например на рисунке ниже



Причем эта модель относится к *линейному типу моделей*. Можно сказать, что это та же линейная регрессия, для которой признаковое пространство выборки было расширено

На самом деле это все та же *линейная регрессия*, в которой однако расширили признаковое пространство: длина вектора признаков x увеличилась за счет добавления в него полиномиальных степеней изначальных признаков. Например, если мы хотим применить полиномиальную регрессию 2-ой степени и имеется выборка, где вектор признаков выглядит как $[x_1, x_2]$, то для каждого наблюдения будет получен новый вектор признаков вида $[1, x_1, x_2, x_1^2, x_1 x_2,, x_2^2]$. И уже преобразованные вектора пойдут на вход модели ***линейной регрессии*** с вектором параметров $w = [w_0, w_1, w_2, w_3, w_4,, w_5]$.

Функционал преобразования признакового пространства в scikit-learn реализован с помощью ***PolynomialFeatures*.** При его создании требуется указать степень полинома n, и затем использовать знакомый по другим объектам из scikit-learn метод fit_transform(), с передаче в него матрицы признаков. На выходе получим матрицу с признаками соответствующим членам полинома n-ой степени, который можно будет передать в модель ***LinearRegression***:

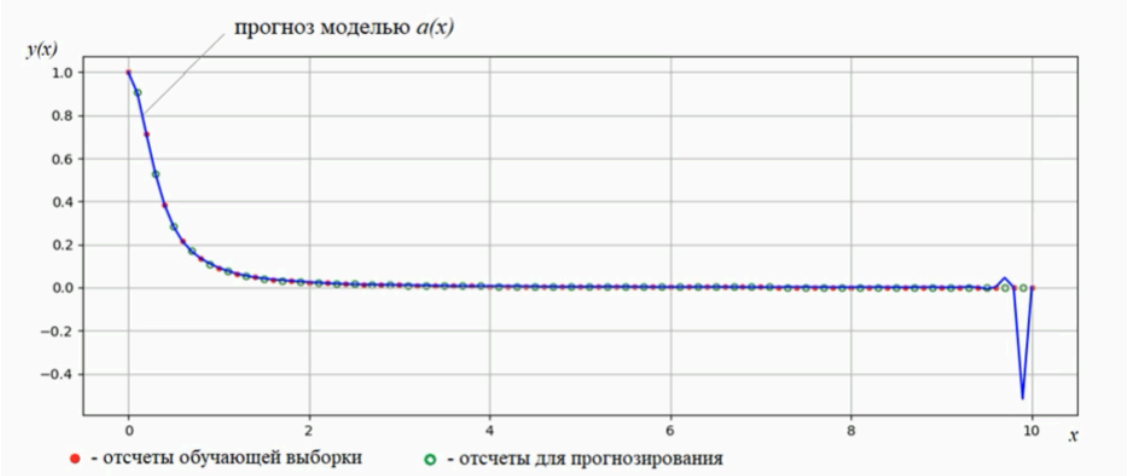
```
from sklearn.preprocessing import PolynomialFeatures
n = 9
poly_features = PolynomialFeatures(n)
```

```
X_train = poly_features.fit_transform(X_train)

linear_model.fit(X_train, y_train)
```

На рисунке показанном чуть ранее, данные описывались полиномом 9-ой степени. В целом, повышение степени полинома модели будет приводить к всё лучшему описанию обучающих данных со стороны модели. Графически это будет выглядеть как эффект интерполяции, когда график функции проходит через каждую точку выборки.

Но в машинном обучении такой исход является нежелательным и называется ***переобучением модели***. То есть в данном случае модель будет выдавать хорошие значения для входных данных из обучающей выборки, однако для данных не из этой выборки выходные значения будут далеки от истинного распределения. Такой эффект можно увидеть на следующем рисунке



Здесь красные точки — значения из обучающей выборки, зеленые — из тестовой, а синий график отражает используемый в обученной модели полином 54-ой степени. Из рисунка видно как значения, полученные моделью, резко расходятся со значениями из тестовой выборки для последних наблюдений.

Поэтому переобучение противоречит одной из целей методов машинного обучения — предсказание выходных значений для входных данных, для которых выходные значения нам неизвестны. Характеристика, из-за которой модель хорошо предсказывает как обучающую, так и тестовую выборки, называется ***обобщающей способностью***.

Способы улучшения обобщающей способности модели

Разделение изначального набора данных на обучающую и тестовую выборки: шаг, который не столько поможет избежать переобучения, сколько выявить факт его наличия.

Для предотвращения же переобучения перед разделением изначальные данные могут проанализировать и составить обучающую и тестовую выборки так, чтобы их статистические характеристики были схожи. Однако самым простым и зачастую действенным способом является рандомизация порядка наблюдений в изначальной выборке и ее дальнейшее разбиение.

L1 и L2 регуляризации: введение ограничений на вектор параметров модели *w*.

Если провести эксперимент и переобучить модель, то можно обнаружить, что значения в её векторе параметров несбалансированны: некоторые значения отличаются на порядоки от других и являются слишком большими.

L1 и L2 регуляризации решают эту проблему путем наложения штрафа на вектор *w*, чтобы его значения не были огромными. Делается это посредством добавления слагаемого в *функцию стоимости*

$$Q_{L1}(w) = \sum_{i=0}^{l} L_i(w) + \lambda \, ||w||_1 = \sum_{i=0}^{l} L_i(w) + \lambda \sum_{i=1}^{n} |w_i| \rightarrow min$$

$$Q_{L2}(w) = \sum_{i=0}^{l} L_i(w) + \lambda \, ||w||_2^2 = \sum_{i=0}^{l} L_i(w) + \lambda \sum_{i=1}^{n} w_i^2 \rightarrow min$$

где $||w||_1$ — L1-норма вектора *w*, $||w||_2$ — L2-норма, а λ — коэффициент регуляризации.

Из формул видно, что для минимизации *функции стоимости*, потребуется также минимизировать веса модели. Параметр λ регулирует величину накладываемого штрафа. Чем он больше, тем меньше должны быть окончательные веса. Чем меньше — тем у модели есть “больше пространства” при подборе параметров.

Обратите также особое внимание, что на смещение, он же параметр w_0, не накладываются ограничения при регуляризации.

Не вдаваясь в строгие подробности, итогом работы L2 регуляризатора будут небольшие веса одного порядка (результат зависит от λ), L1 — также небольшие по значению параметры, часть из которых примет значение 0, если они имеют линейную зависимость от других параметров.

Стоит упомянуть также ***ElasticNet*** — объединение L1 и L2 регуляризаций

$$Q_{ElasticNet}(w) = \sum_{i=0}^{l} L_i(w) + \lambda_1 ||w||_1 + \lambda_2 ||w||_2^2 \rightarrow min$$

В Python линейные модели с функционалом регуляризациями также представлены в пакете *sklearn.linear_model*. Импортировать их можно по альтернативным названиям для регуляризаций: L1 — ***Lasso***, L2 — ***Ridge***, и ***ElasticNet***. При создании модели можно указать коэффициент регуляризации как параметр *alpha* (для *ElasticNet* их два). В остальном работа модели идентична работе *LinearRegression* и также применяет *метод наименьших квадратов* для решения оптимизации:

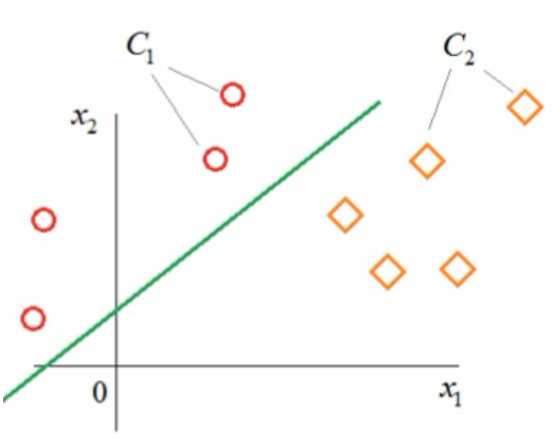
```
from sklearn.linear_model import Ridge
l2_linear_model = Ridge(alpha=2.0)

l2_linear_model.fit(X_train, y_train)
y_pred_test = l2_linear_model.predict(X_test)
```

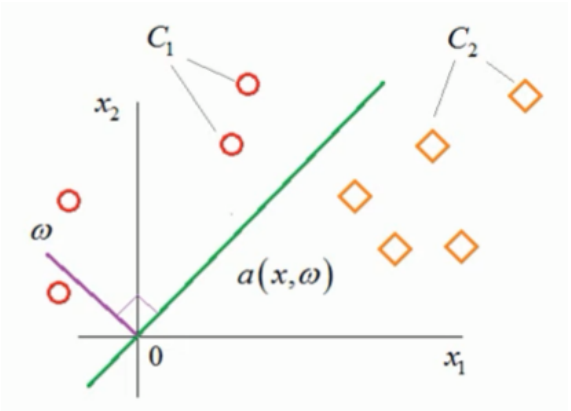
Задача классификации

Классификация — это задача машинного обучения, в которой модель предсказывает категориальную переменную (класс) на основе входных данных. Самая простая классификация относит объект к одному из двух классов, например: определение спама в электронной почте («спам» или «не спам»), диагностика заболеваний («здоров» или «болен»), прогноз оттока клиентов (“уйдет” или “останется” клиент) и т. д.

Решение задачи классификации похоже на решение регрессии — в ней также подбираются параметры функции, которая описывает гиперплоскость. Однако при классификации эта гиперплоскость не повторяет распределение данных, а разделяет гиперпространство на области, которые соответствуют разным классам объектов. На рисунке ниже представлен пример одного из решений классификации, когда имеется два признака и две метки класса:



После разрешения некоторых математический уравнения, для ***линейных классификаторов*** (т.е. для тех, в которых функция имеет только ***линейные операции***) вектор параметров *w* оказывается ***перпендикуляром*** к плоскости, делящей пространство:



Если для классификации на основе двух классов мы примем, что одному классу соответствует метка +1, а другому — -1, тогда модель *a(w, x)* можно представить в виде

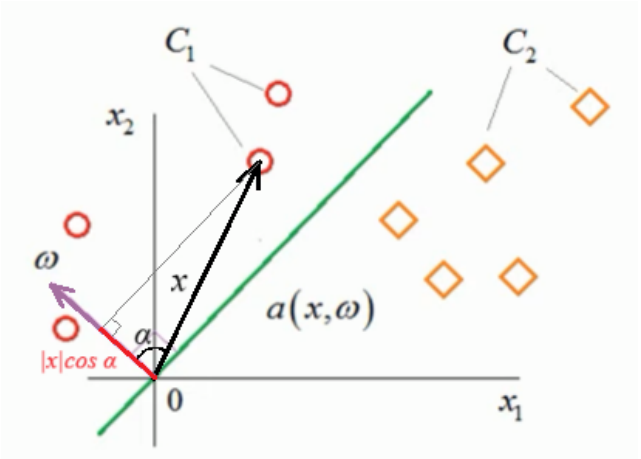
$$a(w, x) = sign(\langle w, x \rangle) = \{1, x \in C_1; -1, x \in C_2\}$$

где $\langle w, x \rangle$ — скалярное произведение векторов *w* и *x*, а C_1 и C_2 — множества классов, представленных на графике выше.

Почему же это выражение верно для примера выше? Распишем скалярное произведение векторов

$$\langle w, x \rangle = |w||x| \cos \alpha$$

где α — угол между векторами *w* и *x*. Если из произведения взять отдельно $|x|\cos \alpha$, то можно увидеть что это есть отображение вектора *x* на вектор *w*:



Из этого следует, что если угол между векторами острый, то их произведение имеет положительное значение, если тупой — отрицательное. Функция же *sign* приводит это значение к +1 или -1.

Как и в задаче регрессии, для нахождения вектора w введем *функцию стоимости*. Для задачи классификации он будет выглядеть следующим образом:

$$Q(w) = \sum_{i=0}^{l} [a(w, x^i) \neq y^i] \rightarrow min$$

где квадратные скобки соответствуют *нотации Айверсона* — обозначению в математике, согласно которому истинное или ложное утверждение, заключенное в квадратные скобки, равно 1, если данное утверждение истинно, и 0, если данное утверждение ложно. Таким образом функционал $Q(w)$ подсчитывает ***количество неверных классификаций***.

Однако, так как метки классов принимают значения +1 и -1, то *функцию стоимости* можно переписать в виде

$$Q(w) = \sum_{i=0}^{l} [a(w, x^i) \cdot y^i < 0] \rightarrow min$$

При совпадении классов произведение будет положительным, при несовпадении — отрицательным. Заменим функцию $a(w, x_i)$ в этом выражении на $\langle w, x \rangle$ и введем понятие ***отступа (margin)***

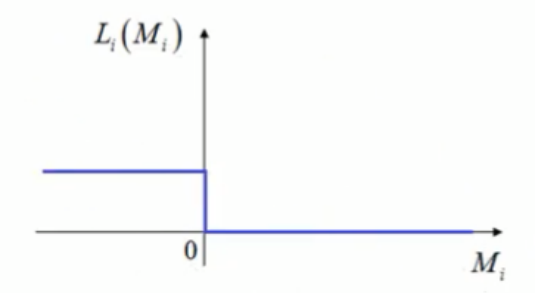
$$M_i = \langle w, x^i \rangle \cdot y^i$$

$$Q(w) = \sum_{i=0}^{l} [a(w, x^i) \cdot y^i < 0] = \sum_{i=0}^{l} [\langle w, x^i \rangle \cdot y^i < 0] = \sum_{i=0}^{l} [M_i < 0] \to min$$

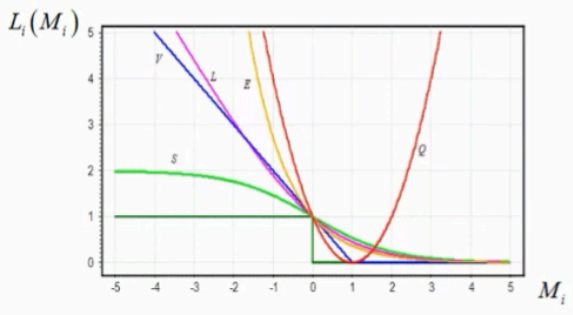
Можно сказать, что *отступ* — эвристика, оценивающая то, *насколько объект "погружен" в свой истинный класс, насколько эталонным представителем он является.* Чем меньше значение отступа, тем ближе объект находится к границе класса, построенной по вектору *w*, соответственно тем выше вероятность ошибочного прогноза. Если же отступ отрицательный, то это значит, что он находиться в области противоположного класса.

В результате, решение задачи классификации также сводится к задаче оптимизации *функции стоимости Q(w).*

Но в тоже время представленная функция потерь $L(w) = [M < 0]$ не лучшим образом подходит для решения задачи оптимизации, так как она не дифференцируема в точке 0, а на остальном диапазоне произвольная — константа. График этой функции представлен ниже:



Поэтому данную функцию потерь аппроксимируют другими похожими на нее функциями, которые на всем диапазоне по значению больше либо равны ей. Примеры таких функций и их графиков изображены на следующем рисунке:



- $V(M) = (1 - M)_+$ - кусочно-линейная (SVM);
- $H(M) = (-M)_+$ - кусочно-линейная (Hebb's rule);
- $L(M) = \log_2(1 + e^{-M})$ - логарифмическая (LR);
- $Q(M) = (1 - M)^2$ - квадратичная (FLD);
- $S(M) = 2 \cdot (1 + e^M)^{-1}$ - сигмоидная (ANN);
- $E(M) = e^{-M}$ - экспоненциальная (AdaBoost).

Вышеперечисленные функции всюду дифференцируемы и каждая из них задает свой дальнейший алгоритм вычисления минимума $Q(w)$.

Логистическая регрессия

В описанном выше подходе классификации, гипотеза модели выстраивалась в признаковом пространстве, а ее выход определялся по знаку векторного произведения w и x. Но есть ли возможность определять класс объекта не по знаку, а по некоторому непрерывному значению, которое также будет нести в себе информацию об объекте?

Предположим, что в задаче бинарной классификации, классы имеют метки 0 и 1. Тогда найдем такую гипотезу $a(w, x)$, что если для объекта с вектором характеристик x гипотеза дает $a(w, x) > 0.5$ то мы отнесем его к классу 1, если же $a(w, x) < 0.5$ — то к классу 0.

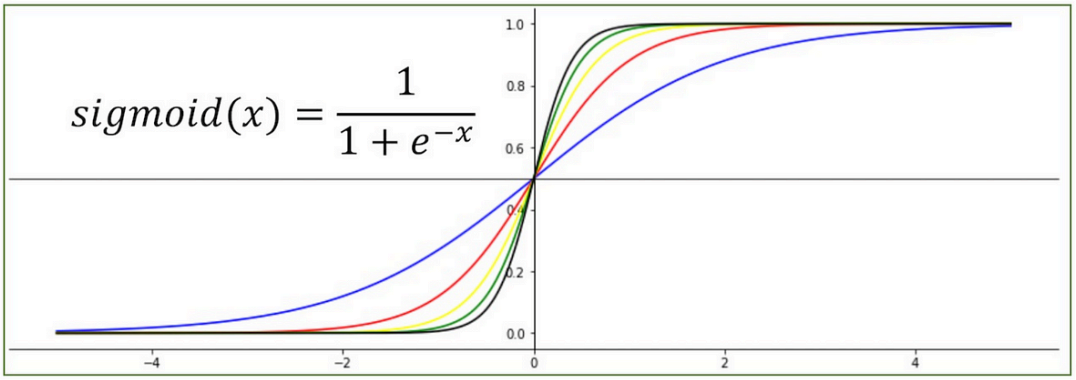
В качестве такой гипотезы можно взять линейную регрессию и при ее значениях больше 0.5 будем относить объект к классу 1.

$$a(w, x) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n$$

Однако множество значений линейной регрессии определено на отрезке от -∞ до +∞. С помощью сигмоидной функции (частный случай логистической функции) значения можно ограничить в диапазоне от 0 до 1, тогда гипотеза примет вид

$$a(w, x) = \sigma(w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n) = (1 + exp\{-(w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n)\})^{-1}$$

Из формулы можно заметить, что здесь именно к линейной регрессии применяется сигмоидная функция. Её график выглядит следующим образом



Таким образом, логистическая регрессия — это обобщение **линейной регрессии** для задач классификации.

Также возникает вопрос о смысловой нагрузке значения, получаемого из гипотезы. Так как значения находятся в диапазоне от 0 до 1 и имеют непрерывный характер, то их можно использовать как вероятность отнесения объекта к классу $y = 1$ при условии, что объект имеет вектор признаков x

$$a(w, x) = P(y = 1 \mid x, w) = (1 + exp\{-(w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n)\})^{-1}$$

Это один из способов перехода к вероятностному типу задач. В них, для вычисления вектора параметров w используют ***функцию правдоподобия***

$$\prod_{i=0}^{l} P(y^i \mid x^i, w) = \prod_{i=0}^{l} p_i^y \, (1 - p_i)^{1-y} \rightarrow max$$

где $p_i = a(w, x^i)$. Каждый отдельный множитель этого произведения характеризует то, с какой вероятностью модель отнесет наблюдение x^i из обучающей выборки к его классу y^i.

Произведение вероятностей, при большом количестве множителей, дает очень малый результат, с которым неудобно работать. Поэтому *функцию правдоподобия* логарифмируют для перехода к сумме, а также умножают на -1 для сведения задачи к поиску минимума. Полученную функцию можно использовать как *функцию стоимости*

$$Q(w) = - log(\prod_{i=0}^{l} P(y^i \mid x^i, w)) = - \sum_{i=0}^{l} log(P(y^i \mid x^i, w)) =$$

$$= - \sum_{i=0}^{l} log(p_i^y \, (1 - p_i)^{1-y}) = - \sum_{i=0}^{l} (y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i)) \rightarrow min$$

Из нее можно выделить функцию потерь и для вероятностного представления задач она имеет название ***logloss***:

$$L_{log}{}^i (w) = -(y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i))$$

Как и ранее, решение задачи классификации *логистической регрессией* свелось к задаче оптимизации *функции стоимости Q(w),* которая является суммой *logloss* для всех объектов обучающей выборки.

Модель логистической регрессии в *scikit-learn* имеет название ***LogisticRegression*** и находится в пакете *sklearn.linear_model*. Процесс взаимодействия такой же как и для других моделей

```
from sklearn.linear_model import LogisticRegression
logreg_model = LogisticRegression()

logreg_model.fit(X_train, y_train)
y_pred_test = logreg_model.predict(X_test)
```

По умолчанию *LogisticRegression* использует L2 регуляризацию. Для смены регуляризации, при создании модели нужно указать желаемый тип в параметре *penalty* (*'l1', 'l2', 'elasticnet' или None*).

Оценка классификационной модели

Самая простая оценка классификации — доля правильных классификаций моделью, то есть отношение суммы правильных ответов модели к общему числу объектов. Такая метрика называется ***accuracy*** и считается следующим образом

$$Accuracy = \frac{1}{l} \sum_{i=0}^{l} [a(w, x^i) = y^i]$$

Доля ошибок в таком случае

$$Error\ Rate = 1 - Accuracy$$

Для расчета *accuracy* примените *accuracy_score*:

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred_test)
```

Но данные оценки имеют свои существенные недостатки в рамках некоторых задач классификации:

1. **Дисбаланс классов** — выборка содержит много больше объектов одного класса по сравнению с противоположным (в случае бинарной классификации).

   Для примера возьмем модель классифицирующую банковские транзакции на мошеннические и нет. Обучающая выборка скорее всего будет иметь дисбаланс классов, так как мошеннических транзакций явно происходит меньше.

   Допустим, что тестовый набор данных состоит из 1000 транзакций, только 50 из которых являются мошенническими. Таким образом, даже

если модель будет определять все транзакции как валидные, то метрика *accuracy* даст значение 0.95, что является хорошим результатом. Однако такая модель явно не отвечает поставленной задаче выявления действий мошенников.

2. **Разная цена ошибок**. В бинарной классификации (метки классов — 0 и 1) возникают два типа ошибок: когда модель выдала для объекта класс 0, а на самом деле он имел метку 1, и когда модель обозначила объект как 1, но у его верная метка 0.

Теперь представим, что стоит задача классификации клиентов банка, которые хотят взять кредит. Модель должна предсказывать — сможет ли конкретный клиент выплатить кредит в будущем (“да”/“нет”).

Пусть имеется выборка основанная прошлых кредитных историях банка, дисбаланс классов отсутствует и в ней есть 500 клиентов выплативших кредит с меткой 1, и 500 — не оплативших с меткой 0. Представим, что мы обучили две модели, основанные на разных методах, и получили для них следующие результаты:

- Первая модель правильно классифицирует все объекты с классом 0, но для объектов класса 1 ошибается в 150 случаях (то есть выдает метку 0, вместо 1).
- Вторая модель правильно классифицирует все 500 объектов с классом 1, а для объектов класса 0 количество ошибок — 150.

Метрика *accuracy* для двух моделей окажется одинаковой — 0.85. Однако по факту результаты работы моделей отличаются и выбор модели для использования в будущем зависит от предпочтений банка: если банк нацелен на экономию денег и хочет, чтобы кредит точно был выплачен, то выбор падет на первую модель; если банк хочет заработать, несмотря на риски невыплаты со стороны некоторых клиентов, то можно использовать вторую модель.

Тогда для оценки модели классификации вводят ***матрицу ошибок (confusion matrix)***, которая считает количество правильных и неправильных классификаций для каждого класса по всей выборке. Для бинарной классификации она выглядит следующим образом:

	$y = 1$	$y = 0$
$a(w, x) = 1$	*True Positive (TP)*	*False Positive (FP)*
$a(w, x) = 0$	*False Negative (FN)*	*True Negative (TN)*

Целевой класс с меткой 1 в таком случае условно называют *Positive* (в этом случае говорят, что *"объект несет целевой признак"*). Противоположный класс с меткой 0 — *Negative* (т.е. *"не несущий целевой признак"*). В данной матрице *Positive* и *Negative* показывают какую метку выдала модель для подсчитываемых объектов. *True* и *False* означают верность классификации полученной моделью. То есть значение *True Positive* показывает сколько объектов с классом 1 модель правильно классифицировала как класс 1, а *False Positive* — сколько объектов с классом 0 модель ошибочно предсказала как класс 1. Такая же логика применяется и для нижней строки матрицы, только для класса *Negative*.

За вычисление матрицы ошибок в *scikit-learn* отвечает *confusion_matrix*:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred_test)
```

Используя инструменты библиотек matplotlib.pyplot и seaborn можно более красиво визуализировать матрицу ошибок:

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(4, 3))
sns.heatmap(cm, annot=True, fmt='d', cmap='bwr')
plt.title('Confusion matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Из значений матрицы определяют две следующие метрики, изменяющие своё значение от 0 до 1:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Precision — точность целевой классификации. Показывает точность модели для класса *Positive*, то есть долю объектов, которые были *правильно отнесены моделью к классу Positive*, относительно всего количества объектов, которые модель *отметила как Positive*.

Recall — полнота. Демонстрирует отношение количества объектов, которые модель *правильно классифицировала как Positive*, к общему числу объектов, которые модель *должна была классифицировать как Positive.*

Какую из этих метрик устремлять к единице, зависит от условий задачи. Например, при аутентификации по биометрии в защищенную систему мы хотим, чтобы злоумышленник точно не попал в систему ($FP \rightarrow 0$). Тогда стоит максимизировать *precision*, возможно за счет того, что персоны имеющие доступ будут распознаны как злоумышленники.

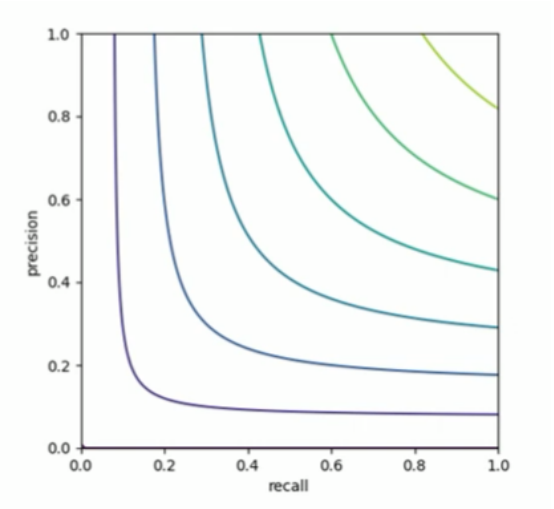
При задаче обнаружения болезни у пациента, целью является минимизация риска, что пациент с болезнью будет классифицирован как здоровый ($FN \rightarrow 0$). В этом случае важна метрика *recall*, хоть её улучшение и может привести к отнесению здорового пациента к больным.

В идеале же модель не ошибается (то есть *FP* и *FN* равны 0) и ее метрики *precision* и *recall* принимают значение 1. Однако на практике идеальную модель получить практически невозможно и хотелось бы иметь такую метрику для модели, которая бы отражала работоспособность относительно *precision* и *recall* одновременно.

Такой оценкой является гармоническое среднее *precision* и *recall*, называемое ***F-мерой (F-score, F_1)***

$$F_1 = 2 \times \frac{1}{\frac{1}{precision} + \frac{1}{recall}} = 2 \times \frac{precision \times recall}{precision + recall}$$

Графически *F-меру* представлена на графике ниже. Линии соответствуют одинаковому значению для получаемой *F-меры*, чем светлее линия тем больше *F-мера*:



Если для задачи какая-то из метрик *precision* или *recall* более важна, то вводят коэффициент β и получают ***F_β-меру***

$$F_\beta = (1 + \beta^2)\frac{precision \times recall}{\beta^2 \cdot precision + recall}$$

Вышеописанные оценки можно как посчитать вручную с помощью стандартных инструментов Python, взяв значения из *confusion_matrix*. Или же задействовать функционал из *sklearn.metrics* для соответствующих оценок: *precision_score*, *recall_score*, *f1_score* и *fbeta_score*. А также *classification_report* для комплексного отчета по работе классификационной модели:

```
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred_test)
```

Задание

1. Разделить датасет, подготовленный на первой лабораторной работе, на обучающую и тестовую выборки
2. Решить задачу регрессии для одного из непрерывных признаков в датасете
3. Оценить работу регрессионной модели. При плохих результатах подумать как можно его улучшить
4. Решить задачу классификации
5. Оценить работу классификационной модели. При плохих результатах подумать как можно его улучшить
6. Выгрузить результат работы на Github