**Лабораторная работа 4. Логистическая регрессия.**

Реализуем логистическую регрессию. Начнём с импорта библиотеки:

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

**import** pandas **as** pd

**from** sklearn.datasets **import** load\_boston

**from** sklearn.metrics **import** mean\_squared\_error, f1\_score, accuracy\_score, roc\_curve, roc\_auc\_score

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

**from** matplotlib **import** pyplot **as** plt

В случае с логистической регрессией мы можем использовать только**градиентный спуск**, так как нет явного матричного способа найти оптимальные коэффициенты. В качестве функции потерь будем использовать **бинарную кросс-энтропию**, Log Loss. Она записывается так:

Изображение выглядит как текст, Шрифт, белый, рукописный текст

Автоматически созданное описание

Будем использовать другой датасет с задачей **классификации**, где нужно определить зарплату меньше и больше определённого значения. Убираем в данных лишние признаки, конвертируем целевой столбец в бинарные значения и нормализуем данные.

Реализуем функцию sigmoid и функцию, вычисляющую **градиент бинарной кросс-энтропии**:

def sigmoid(X, theta):

return 1. / (1. + np.exp(-X.dot(theta)))

def calc\_binary\_cross\_entropy\_grad(X, y, theta):

n = X.shape[0]

grad = 1. / n \* X.transpose().dot(sigmoid(X, theta) - y)

return grad

def gradient\_step(theta, theta\_grad, alpha):

return theta - alpha \* theta\_grad

def optimize(X, y, grad\_func, start\_theta, alpha, n\_iters):

theta = start\_theta.copy()

for i in range(n\_iters):

theta\_grad = grad\_func(X, y, theta)

theta = gradient\_step(theta, theta\_grad, alpha)

return theta

Сделаем **предсказания** на тренировочной выборке и посчитаем значение метрики Accuracy  и  F1-score:

y\_pred = sigmoid(X, theta) > 0.5

print\_logisitc\_metrics(y, y\_pred)

Мы уже знаем, что этой выборке нельзя доверять, поэтому разбиваем данные и оптимизируем:

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X, y, test\_size=0.2)

theta = optimize(X\_train, y\_train, calc\_binary\_cross\_entropy\_grad, np.ones(m), 1., 300)

y\_pred = sigmoid(X\_valid, theta) > 0.5

print\_logisitc\_metrics(y\_valid, y\_pred)

Результат практически тот же. Отрисуем **ROC-кривую**, посчитаем её значения и значение площади под кривой AUC.

Для борьбы с **переобучением** добавим **регуляризацию**. Обернём линейную регрессию в **класс**:

**class** **RegOptimizer**():

**def** **\_\_init\_\_**(self, alpha, n\_iters):

self.theta = None

self.\_alpha = alpha

self.\_n\_iters = n\_iters

**def** **gradient\_step**(self, theta, theta\_grad):

**return** theta - self.\_alpha \* theta\_grad

**def** **grad\_func**(self, X, y, theta):

**raise** NotImplementedError()

**def** **optimize**(self, X, y, start\_theta, n\_iters):

theta = start\_theta.copy()

**for** \_ **in** range(n\_iters):

theta\_grad = self.grad\_func(X, y, theta)

theta = self.gradient\_step(theta, theta\_grad)

**return** theta

**def** **fit**(self, X, y):

m = X.shape[1]

start\_theta = np.ones(m)

self.theta = self.optimize(X, y, start\_theta, self.\_n\_iters)

**def** **predict**(self, X):

**raise** NotImplementedError()

Проделаем ту же операцию с логистичекой регрессией:

**class** **LogReg**(**RegOptimizer**):

**def** **sigmoid**(self, X, theta):

**return** 1. / (1. + np.exp(-X.dot(theta)))

**def** **grad\_func**(self, X, y, theta):

n = X.shape[0]

grad = 1. / n \* X.transpose().dot(self.sigmoid(X, theta) - y)

**return** grad

**def** **predict\_proba**(self, X):

**return** self.sigmoid(X, self.theta)

**def** **predict**(self, X):

**if** self.theta is **None**:

**raise** **Exception**('You should train the model first')

y\_pred = self.predict\_proba(X) > 0.5

**return** y\_pred

Избавлимся от лишних признаков, нормализуем данные. С переобучением боремся с помощью **регуляризации**.

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

В качестве экспериментальных данных возьмем датасет о доходах граждан в различных странах Adult Income (Далее сделаем необходимую предобработку:

adult = pd.read\_csv('./data/adult.data',

names=['age', 'workclass', 'fnlwgt', 'education',

'education-num', 'marital-status', 'occupation',

'relationship', 'race', 'sex', 'capital-gain',

'capital-loss', 'hours-per-week', 'native-country', 'salary'])

**Задание 3B.6.1**

Постройте модель логистической регрессии при помощи [sklearn](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html" \t "[object Object]). Используйте параметры по умолчанию, обучите на всей выборке и посчитайте *F1 score*.

**Задание 3B.6.2**

Посчитайте *confusion matrix* для классификатора из задачи 3.6.1. Для получения матрицы можно воспользоваться методом sklearn.metrics.confusion\_matrix(y\_true, y\_pred), либо посчитать каждый элемент вручную.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Введите значения получившейся матрицы в соответствующие ячейки.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  | | --- | --- | | **True Negative** | **False Positive** | | **False Negative** | **True Positive** | |

**Задание 3B.6.3**

Постройте ROC-кривую и посчитайте  для классификатора из задачи 3.6.1.

**Задание 3B.6.4**

Постройте модель логистической регрессии при помощи *sklearn* без регуляризации. Чему равен *F1 score* ?

**Задание 3B.6.5**

Переберите коэффициенты -регуляризации от  до  с шагом  и определите, на каком из них модель логистической регрессии из *sklearn* даёт наибольший *F1 score* .

**Задание 3B.6.6**

Замените в столбце native-country страны, у которых меньше ста записей, на other, поменяйте этот столбец на dummy-переменные, обучите классификатор на всей выборке и посчитайте *F1 score* .