

Санкт-Петербургский государственный университет

Прикладная математика и информатика

Отчет по учебной практике 1 (научно-исследовательской работе) (семестр 1)

Задача классификации с использованием логистической регрессии

Выполнил:

Погребников Николай Вадимович, группа

21.Б05-мм *подпись*

Научный руководитель:

Кандидат наук, доцент

• Петр Валерьевич.

а статистического моделирования

Работа выполнена  
в удовлетворительном  
объеме

Отметка о зачете:

<< Работа выполнена на среднем уровне и может быть зачтена с оценкой С>>

Санкт-Петербург

2021

# Введение

В современном мире всё чаще и чаще часто возникают задачи разделения одних данных от других. Это обусловлено повышающимся спросом людей на какие-либо онлайн сервисы. Количество самих данных тоже безмерно растёт, и никакой человек не сможет выполнить задачу классификации так же быстро, как и компьютер.

В данной работе передо мной была поставлена задача классификации с использованием логистической регрессии на основе языка программирования Python и библиотеки Scikit-Learn.

## Основная часть

### Теория

Перед рассмотрением конкретной задачи классификации мне предстояло ознакомиться с теоретической частью. В этом мне помогла книга Орельена Жерона "Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow". Мною была прочитана 4 глава, в которой рассматривались основы программирования на языке Python с использованием библиотеки Scikit-Learn и следующие понятия: линейная регрессия, градиентный спуск и его виды, полиномиальная регрессия, кривые обучения, регуляризированные линейные модели, логистическая регрессия. В данной книге был представлен наглядный пример использования логистической регрессии для выявления вида ириса по ширине и длине его лепестка, который послужил отправной точкой для моей работы.

### Практическая часть

Следующий шаг после изучения теории, состоял в том, чтобы выбрать данные для задачи. В этом мне помог ресурс UC Irvine Machine Learning Repository, на котором представлен набор данных португальского банка «Santander Totta». Меня интересовала следующая задача: на основе данных по 20000 клиентов (по каждому имеется 21 признак), спрогнозировать успех(столбец “у”, где 1 — успех, 0 — обратное) подписки других 20000 клиентов на депозит, оценить точность данного прогноза.

Первая подзадача состояла в том, чтобы перенести корректно данные из файла bank.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	eurobank3m	nr.employed	y
2	44	blue-collar	married	basic.4y	unknown	yes	no	cellular	aug	thu	210	1	999	0	nonexistent	1.4	93.444	-36.1	4.963	5228.1	0
3	53	technician	married	unknown	no	no	no	cellular	nov	fri	138	1	999	0	nonexistent	-0.1	93.2	-	-42.021	5195.8	0
4	28	management	single	university.degree	no	yes	no	cellular	jun	thu	339	3	6	2	success	-1.7	94.055	-39.8	0.729	4991.6	1
5	39	services	married	high.school	no	no	no	cellular	apr	fri	185	2	999	0	nonexistent	-1.8	93.075	-47.1	1.405	5099.1	0
6	55	retired	married	basic.4y	no	yes	no	cellular	aug	fri	137	1	3	1	success	-2.9	92.201	-31.4	0.863	5076.2	1
7	30	management	divorced	basic.4y	no	yes	no	cellular	jul	tue	68	8	999	0	nonexistent	1.4	93.918	-42.7	4.961	5228.1	0
8	37	blue-collar	married	basic.4y	no	yes	no	cellular	may	thu	204	1	999	0	nonexistent	-1.8	92.893	-46.2	1.327	5099.1	0
9	39	blue-collar	divorced	basic.9y	no	yes	no	cellular	may	fri	191	1	999	0	nonexistent	-1.8	92.893	-46.2	1.313	5099.1	0
10	36	admin.	married	university.degree	no	no	no	cellular	jun	mon	174	1	3	1	success	-2.9	92.963	-40.8	1.266	5076.2	1
11	27	blue-collar	single	basic.4y	no	yes	no	cellular	apr	thu	191	2	999	1	failure	-1.8	93.075	-47.1	1.41	5099.1	0

в программу, с чем мне помогла библиотека pandas, которую я обозначил как pd

```
7     data = pd.read_csv('bank.csv', header=0)
8     data = data.dropna()
```

Так выглядит их представление в языке Python:

	age	job	marital	...	euribor3m	nr_employed	y
0	44	blue-collar	married	...	4.963	5228.1	0
1	53	technician	married	...	4.021	5195.8	0
2	28	management	single	...	0.729	4991.6	1
3	39	services	married	...	1.405	5099.1	0
4	55	retired	married	...	0.869	5076.2	1

Кроме того, было необходимо привести все данные к числовому виду, поскольку для использования линейной регрессии неприемлем строковый тип. Для этого я принял решение использования индикаторных переменных. Суть данного метода в том, чтобы преобразовать столбцы из не числовых данных в множество других столбцов, состоящих из нулей и единиц. Пример: столбец «job» был преобразован в столбцы: «job\_admin.», «job\_blue-collar», ..., «job\_unemployed», «job\_unknown»

	job_admin.	job_blue-collar	...	job_unemployed	job_unknown
0	0		1	...	0
1	0		0	...	0
2	0		0	...	0
3	0		0	...	0
4	0		0	...	0

Далее данные были разделены на обучающие и тестовые (в соотношении 1:1):

```
30     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
```

Теперь с этими данными можно работать.

Второй подзадачей являлась построение логистической регрессии. Логистическая регрессия (также называемая логит-регрессией (logit regression)) обычно используется для оценки вероятности того, что образец принадлежит к определенному классу (например, какова вероятность того, что заданное почтовое сообщение является спамом?). Если оценочная вероятность больше 50%, тогда модель прогнозирует, что образец принадлежит к данному классу (называемому положительным классом, помеченным "1"), а иначе - что не

принадлежит (т.е. относится к отрицательному классу, помеченному "0"). Это делает ее двоичным классификатором.

Логистическая регрессионная модель подсчитывает взвешенные суммы входных признаков (плюс член смещения), выдаёт вероятность принадлежности к классу по данной формуле:

$$\hat{p} = h_{\theta}(\mathbf{x}) = \sigma(\theta^T \cdot \mathbf{x})$$

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

- $\theta$  — *вектор параметров* модели, содержащий член смещения  $\theta_0$  и веса признаков от  $\theta_1$  до  $\theta_n$ .
- $\theta^T$  — транспонированный  $\theta$  (вектор-строка вместо вектора-столбца).
- $\mathbf{x}$  — *вектор признаков* образца, содержащий от  $x_0$  до  $x_n$ , где  $x_0$  всегда равно 1.
- $\theta^T \cdot \mathbf{x}$  — скалярное произведение  $\theta^T$  и  $\mathbf{x}$ .
- $h_{\theta}$  — функция гипотезы, использующая параметры модели  $\theta$ .

Если вероятность меньше 0.5, тогда модель прогнозирует 0, иначе — 1.

Для обучения модели необходимо задать функцию издержек, которая на полном обучающем наборе представляет собой средние издержки на всех обучающих образцах, она выпуклая и поэтому, например, градиентный спуск гарантированно отыщет глобальный минимум:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[ y^{(i)} \log(\hat{p}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{p}^{(i)}) \right]$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^m (\sigma(\theta^T \cdot \mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

По аналогии с примером из книги, основанным на ирисах, я использовал лишь функцию `LogisticRegression()` из библиотеки `sklearn.linear_model`,

```
30      log_reg = LogisticRegression()
31      log_reg.fit(X_train, y_train)
```

что привело меня к данной ошибке

```
Increase the number of iterations (max_iter) or scale the data as shown in:  
https://scikit-learn.org/stable/modules/preprocessing.html
```

Перейдя по ссылке, я понял, что мои признаки не масштабированы. Во второй главе книги Орельена Жерона было представлено два вида масштабирования: StandardScaler и MinMaxScaler. Я решил использовать StandardScaler, поскольку стандартизация гораздо менее подвержена влиянию выбросов.

```
29     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)  
30     pipe = make_pipeline(StandardScaler(), LogisticRegression())  
31     pipe.fit(X_train, y_train)
```

Просмотрев тестовые данные, программа вывела следующую информацию:

```
Метод решения: lbfgs; Регрессия: l2  
Время: 0.3402578830718994  
Точность: 0.9131785957074876
```

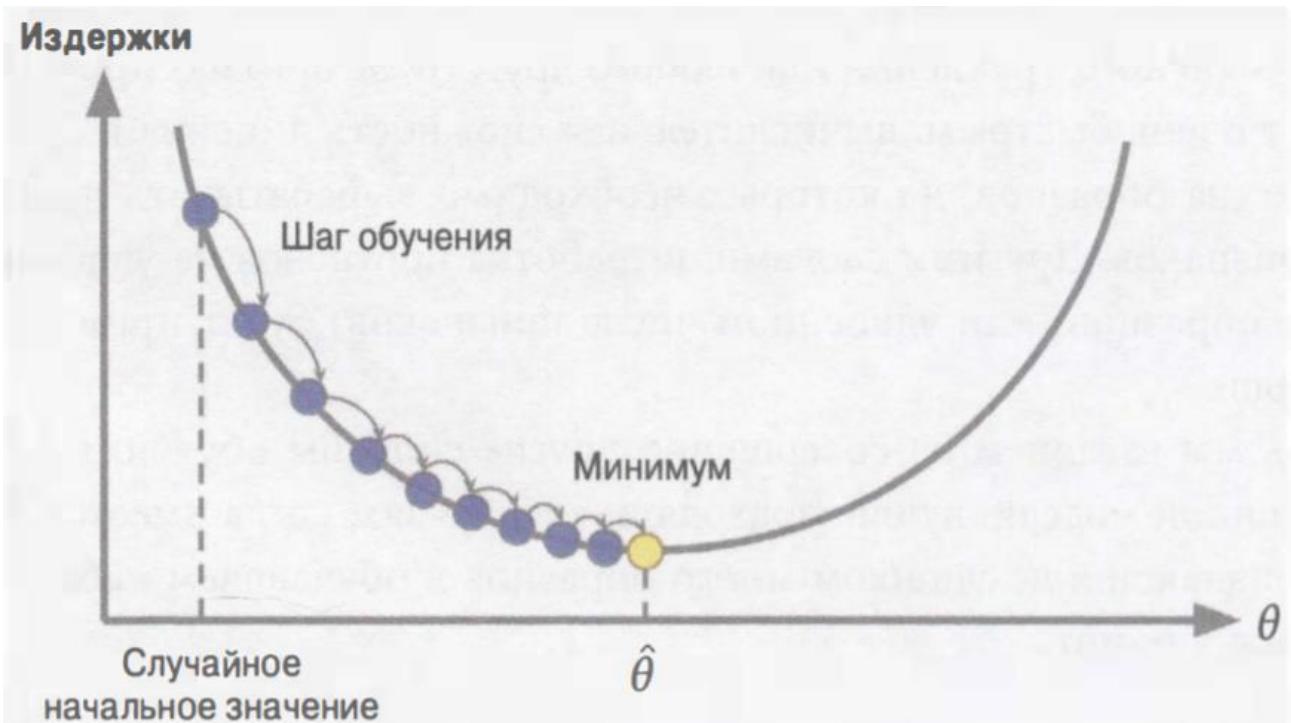
Однако была рассмотрена только модель с применением метода решения lbfgs, поскольку она используется по умолчанию при применении функции LogisticRegression().

## Методы решения

Рассмотрим два вида решений: lbfgs(limited Broyden, Fletcher, Goldfarb, Shanno) и saga(Stochastic Average Gradient).

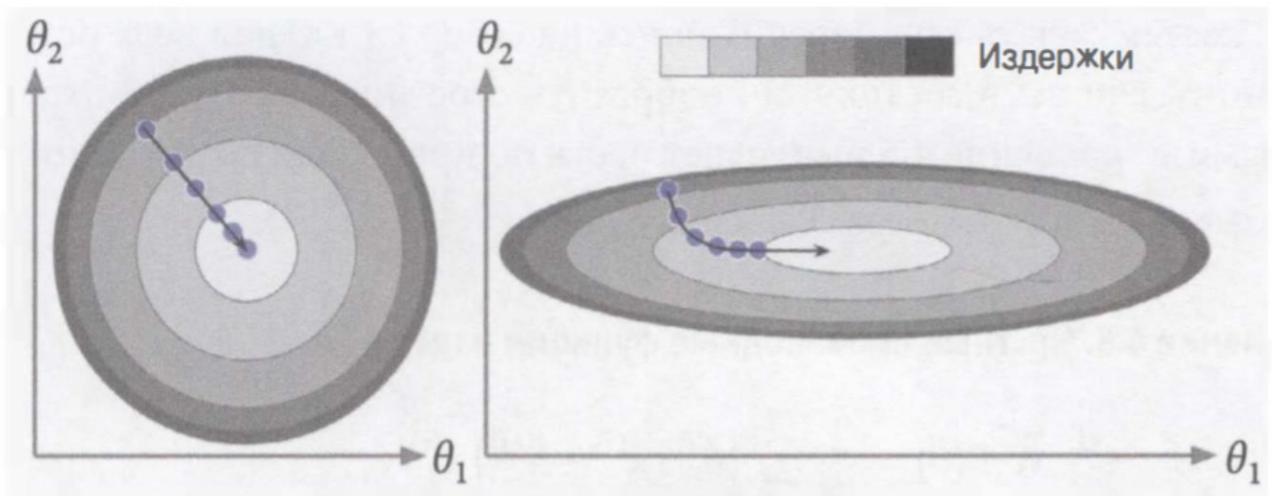
### Стохастический градиентный спуск

Градиентный спуск представляет собой самый общий алгоритм оптимизации, способный находить оптимальные решения широкого диапазона задач. Основная идея градиентного спуска заключается в том, чтобы итеративно подстраивать параметры для сведения к минимуму функции издержек. Предположим, вы потерялись в горах в густом тумане; вы способны прощупывать только крутизну поверхности под ногами. Хорошая стратегия быстро добраться до dna долины предусматривает движение вниз по самому кругому спуску (не поступайте так в реальных горах, потому что быстро - не значит безопасно; ходите по указанным на карте тропам - примеч. пер.). Это в точности то, что делает градиентный спуск: он измеряет локальный градиент функции ошибок применительно к вектору параметров  $\theta$  и двигается в направлении убывающего градиента. Как только градиент становится нулевым, вы достигли минимума.



Выражаясь более конкретно, вы начинаете с наполнения вектора случайными значениями (случайная инициализация). Затем вы постепенно улучшаете его, предпринимая по одному маленькому шагу за раз и на каждом шаге пытаясь снизить функцию издержек (например, MSE) до тех пор, пока алгоритм не сойдется в минимуме.

Кроме того, очень важно масштабировать признаки при использовании градиентного спуска:



Градиентный спуск с масштабированием    Градиентный спуск без масштабирования

Слева алгоритм градиентного спуска устремляется прямо к минимуму, из-за чего достигает его быстро, а справа он сначала двигается в

направлении, которое практически перпендикулярно направлению к глобальному минимуму, и заканчивает длинным маршем по почти плоской долине. Минимум в итоге достигается, но за долгое время.

Теперь рассмотрим стохастический градиентный спуск. Главная проблема с градиентным спуском - тот факт, что он использует полный обучающий набор для вычисления градиентов на каждом шаге, который делает его очень медленным в случае крупного обучающего набора. Как противоположная крайность, стохастический градиентный спуск на каждом шаге просто выбирает из обучающего набора случайный образец и вычисляет градиенты на основе только этого единственного образца. Очевидно, алгоритм становится гораздо быстрее, так как на каждой операции ему приходится манипулировать совсем малым объемом данных. В библиотеке Scikit-Learn реализован усреднённый стохастический градиентный спуск(saga) — модификация обычного стохастического градиентного спуска, которую мы в последующем и будем использовать.

### **Алгоритм Бройдена — Флетчера — Гольдфарба — Шанно (BFGS)**

Метод BFGS, итерационный метод численной оптимизации, назван в честь его исследователей: Broyden, Fletcher, Goldfarb, Shanno. Относится к классу так называемых квазиньютоновских методов. В отличие от ньютоновских методов в квазиньютоновских не вычисляется напрямую гессиан функции, т.е. нет необходимости находить частные производные второго порядка. Вместо этого гессиан вычисляется приближенно, исходя из сделанных до этого шагов.

Модификация L-BFGS(ограниченное использование памяти) — используется в случае большого количества неизвестных.

## **Разновидности регуляризации и их реализации с применением усреднённого градиентного спуска**

Для линейной модели регуляризация обычно достигается путем ограничения весов модели. Мы рассмотрим на примере saga гребневую регрессию(12), лассо-регрессию(11) и эластичную сеть(elastic net), которые реализуют три разных способа ограничения весов.

### **Гребневая регрессия**

Гребневая регрессия работает следующим образом: к функции издержек добавляется член регуляризации. Это заставляет алгоритм обучения не только приспосабливаться к данным, но также удерживать веса модели насколько возможно небольшими. Гиперпараметр  $\alpha$  управляет тем, насколько необходимо регуляризовать модель. Таким образом, функция издержек принимает вид:

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^n \theta_i^2$$

В случае использования библиотеки Scikit-Learn функции «LogisticRegression()» и « LogisticRegression(penalty="l2")» ничем не отличаются. Гиперпараметр `penalty` устанавливает используемый тип члена регуляризации. Указание "l2" обозначает то, что мы желаем, чтобы алгоритм добавлял к функции издержек член регуляризации, равный одной второй квадрату нормы весового вектора.

Результаты программы при использовании гребневой регрессии следующие:

```
Метод решения: saga; Регрессия: l2
Время: 37.01811623573303
Точность: 0.9133728270370011
```

### Лассо-регрессия

Регрессия методом наименьшего абсолютного сокращения и выбора, называемая лассо-регрессией, в точности как гребневая регрессия добавляет к функции издержек член регуляризации, но вместо одной второй квадрата нормы весового вектора использует просто норму весового вектора. Важной характеристикой лассо-регрессии является то, что она стремится полностью исключить веса наименее важных признаков.

Функция издержек:

$$J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^n |\theta_i|$$

Результаты программы при использовании лассо-регрессии:

```
Метод решения: saga; Регрессия: l1
Время: 46.69146776199341
Точность: 0.9133728270370011
```

### Эластичная сеть

Эластичная сеть - это серединная точка между гребневой регрессией и лассо-регрессией. Член регуляризации представляет собой просто смесь членов регуляризации гребневой регрессии и лассо-регрессии, к тому же можно

управлять отношением смеси  $r$ . При  $r = 0$  эластичная сеть эквивалентна гребневой регрессии, а при  $r = 1$  она эквивалентна лассо-регрессии.

Функция издержек:

$$J(\theta) = \text{MSE}(\theta) + r\alpha \sum_{i=1}^n |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^n \theta_i^2$$

Результаты программы при использовании лассо-регрессии:

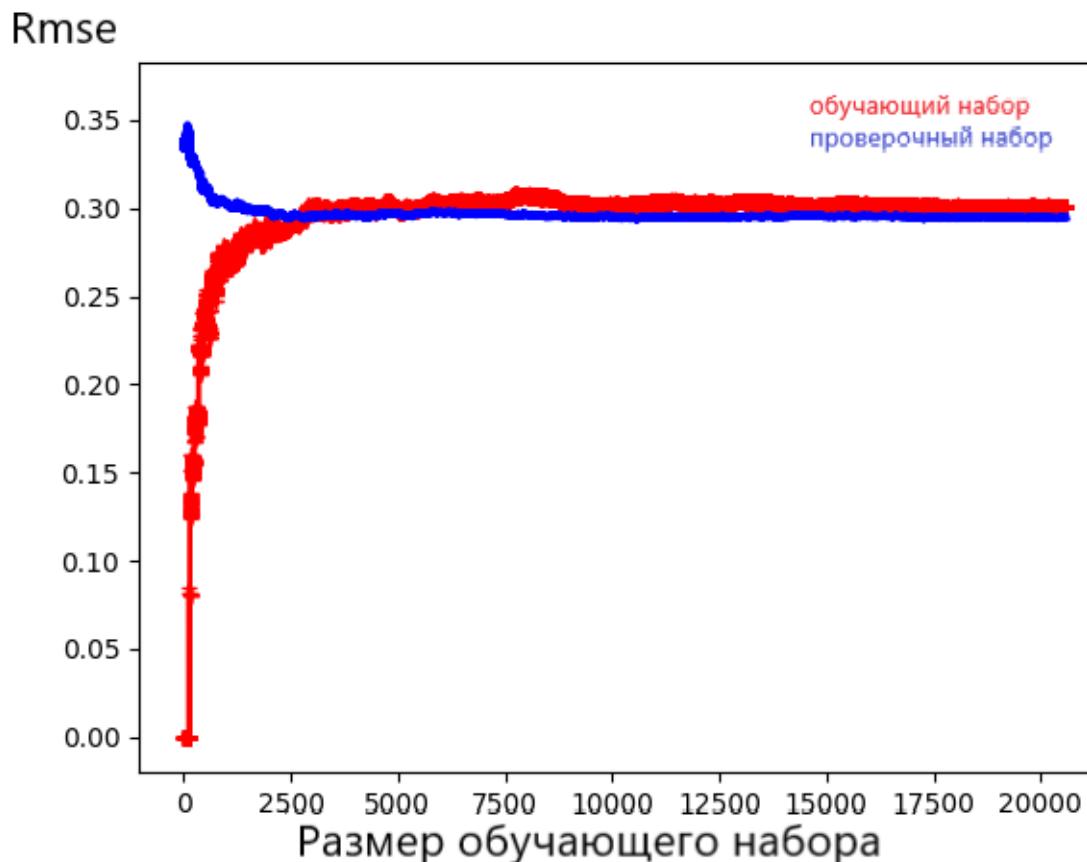
```
Метод решения: saga; Регрессия: elasticnet
Время: 47.0902533531189
Точность: 0.9132757113722444
```

## Оценка результатов

Сравнивая методы решений, мы можем заметить огромную разницу во времени работы программы: использование метода `lbfgs` тратит меньше памяти, в следствие чего время так же понижается, однако метод стохастического градиентного спуска работает несколько точнее.

## Кривые обучения

Рассмотрим кривые обучения на примере метода lbfgs. Они представляют собой графики производительности модели на обучающем наборе и проверочном наборе как функции от размера обучающего набора (или итерации обучения). Обучая модель несколько раз получаем данный график:



Можно заметить, что когда образцов в обучающем наборе мало(до 2000), модель подогнана к нему, однако по мере добавления образцов в обучающий набор идеальная подгонка модели к обучающим данным становится невозможной. С проверочным набором происходит обратное: с каждой итерацией ошибка медленно снижается, что улучшает работу модели.

## Вывод

Использование логистической регрессии привело к удовлетворительным результатам классификации данных.

## **Заключение**

В процессе выполнения научно-исследовательской работы я познакомился с понятиями линейной регрессии, масштабирования, освоил базовые навыки написания кода на языке Python и решил классическую задачу классификации с использованием линейной регрессии и сравнил различные методы реализации логистической регрессии.

## Приложение

### Обработка входных данных

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from time import time
from sklearn.metrics import mean_squared_error

data = pd.read_csv('bank.csv', header=0)
data = data.dropna()

#нахождение индикаторных переменных
no_indicate_vars = []
for i in range(len(data.columns)):
    if type(data.values[0][i]) == str:
        no_indicate_vars.append(data.columns[i])

#создание индикаторных переменных
for var in no_indicate_vars:
    no_indicate_list = pd.get_dummies(data[var], prefix=var)
    data = data.join(no_indicate_list)
    data.pop(var)

#разделение данных
X = data.loc[:, data.columns != 'y']
y = data.loc[:, data.columns == 'y']

X = X.values.tolist()
y = y.values.tolist()
y = np.array(y).ravel()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
```

## Различные реализации обучения модели

```
#lbfgs
t = time()
pipe = make_pipeline(StandardScaler(), LogisticRegression(max_iter=40000))
pipe.fit(X_train, y_train)
print("Метод решения: lbfgs; Регрессия: l2")
print('Время:', time()-t)
print('Точность:', pipe.score(X_test, y_test))
print()

#gребневая регрессия
t = time()
pipe = make_pipeline(StandardScaler(), LogisticRegression(solver="saga",
penalty="l2", max_iter=40000))
pipe.fit(X_train, y_train)
print("Метод решения: saga; Регрессия: l2")
print('Время:', time()-t)
print('Точность:', pipe.score(X_test, y_test))
print()

#лассо-регрессия
t = time()
pipe = make_pipeline(StandardScaler(), LogisticRegression(solver="saga",
penalty="l1", max_iter=40000))
pipe.fit(X_train, y_train)
print("Метод решения: saga; Регрессия: l1")
print('Время:', time()-t)
print('Точность:', pipe.score(X_test, y_test))
print()

#эластичная сеть
t = time()
pipe = make_pipeline(StandardScaler(), LogisticRegression(random_state=0,
solver="saga", penalty="elasticnet", max_iter=40000, l1_ratio=0.2))
pipe.fit(X_train, y_train)
print("Метод решения: saga; Регрессия: elasticnet")
print('Время:', time()-t)
print('Точность:', pipe.score(X_test, y_test))
```

## Кривые обучения

```
#вывод кривых обучения
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.5, random_state=0)
train_errors, val_errors = [], []
log_reg = LogisticRegression(max_iter=40000)
print(len(X_train))
for m in range(10, len(X_train)):
    log_reg.fit(X_train[:m], y_train[:m])
    print(m)
    y_train_predict = log_reg.predict(X_train[:m])
    y_val_predict = log_reg.predict(X_val)
    train_errors.append(mean_squared_error(y_train_predict, y_train[:m]))
    val_errors.append(mean_squared_error(y_val_predict, y_val))
plt.plot(np.sqrt(train_errors), "r-+", linewidth=2, label="train")
plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="val")
plt.show()
```

## **Источники информации**

- Орельен Жерон Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем. Пер. с англ. - СпБ.: ООО "Альфа-книга": 2018. - 688 с.: ил. - Парал. тит. англ.
- <https://archive.ics.uci.edu/ml/index.php>
- <https://scikit-learn.org/stable/index.html>
- <https://pandas.pydata.org>