Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Кафедра информатики

Лабораторная работа №2

«Линейная регрессия. Многоклассовая классификация.»

Выполнил: Чёрный Родион Павлович
магистрант кафедры информатики
группа №858642

Проверил: доцент кафедры информатики
Стержанов Максим Валерьевич

Минск 2019

Постановка задачи

Набор данных **ex2data1.txt** представляет собой текстовый файл, содержащий информацию об оценке студента по первому экзамену (первое число в строке), оценке по второму экзамену (второе число в строке) и поступлении в университет (0 - не поступил, 1 - поступил).

Набор данных **ex2data2.txt** представляет собой текстовый файл, содержащий информацию о результате первого теста (первое число в строке) и результате второго теста (второе число в строке) изделий и результате прохождения контроля (0 - контроль не пройден, 1 - контроль пройден).

Набор данных **ex2data3.mat** представляет собой файл формата *.mat (т.е. сохраненного из Matlab). Набор содержит 5000 изображений 20x20 в оттенках серого. Каждый пиксель представляет собой значение яркости (вещественное число). Каждое изображение сохранено в виде вектора из 400 элементов. В результате загрузки набора данных должна быть получена матрица 5000x400. Далее расположены метки классов изображений от 1 до 9 (соответствуют цифрам от 1 до 9), а также 10 (соответствует цифре 0).

1. Загрузите данные **ex2data1.txt** из текстового файла.
2. Постройте график, где по осям откладываются оценки по предметам, а точки обозначаются двумя разными маркерами в зависимости от того, поступил ли данный студент в университет или нет.
3. Реализуйте функции потерь J(θ) и градиентного спуска для логистической регрессии с использованием векторизации.
4. Реализуйте другие методы (как минимум 2) оптимизации для реализованной функции стоимости (например, Метод Нелдера — Мида, Алгоритм Бройдена — Флетчера — Гольдфарба — Шанно, генетические методы и т.п.). Разрешается использовать библиотечные реализации методов оптимизации (например, из библиотеки scipy).
5. Реализуйте функцию предсказания вероятности поступления студента в зависимости от значений оценок по экзаменам.
6. Постройте разделяющую прямую, полученную в результате обучения модели. Совместите прямую с графиком из пункта 2.
7. Загрузите данные **ex2data2.txt** из текстового файла.
8. Постройте график, где по осям откладываются результаты тестов, а точки обозначаются двумя разными маркерами в зависимости от того, прошло ли изделие контроль или нет.

9. Постройте все возможные комбинации признаков x_1 (результат первого теста) и x_2 (результат второго теста), в которых степень полинома не превышает 6, т.е. 1, x_1, x_2, x_1^2, x_1x_2, x_2^2, …, $x_1x_2^5$, x_2^6 (всего 28 комбинаций).
10. Реализуйте L2-регуляризацию для логистической регрессии и обучите ее на расширенном наборе признаков методом градиентного спуска.
11. Реализуйте другие методы оптимизации.
12. Реализуйте функцию предсказания вероятности прохождения контроля изделием в зависимости от результатов тестов.
13. Постройте разделяющую кривую, полученную в результате обучения модели. Совместите прямую с графиком из пункта 7.
14. Попробуйте различные значения параметра регуляризации λ. Как выбор данного значения влияет на вид разделяющей кривой? Ответ дайте в виде графиков.
15. Загрузите данные **ex2data3.mat** из файла.
16. Визуализируйте несколько случайных изображений из набора данных. Визуализация должна содержать каждую цифру как минимум один раз.
17. Реализуйте бинарный классификатор с помощью логистической регрессии с использованием векторизации (функции потерь и градиентного спуска).
18. Добавьте L2-регуляризацию к модели.
19. Реализуйте многоклассовую классификацию по методу “один против всех”.
20. Реализуйте функцию предсказания класса по изображению с использованием обученных классификаторов.
21. Процент правильных классификаций на обучающей выборке должен составлять около 95%.

Описание реализации

1. Метод для чтения данных из файла в формате csv:

```
def read_csv(filename):
    csv_data = pd.read_csv(filename, header=None)
    cols = len(csv_data.columns)
    X = np.array([np.array(csv_data[col]) for col in range(cols - 1)]).T
    y = np.array(csv_data[cols - 1], dtype=float)
    return X, y
```

2. Изобразим данные из файла ex2data1.txt в виде точек на графике: синяя - если по результатам оценок за экзамен было зачисление, оранжевое - если нет. График изображен на рисунке 1:

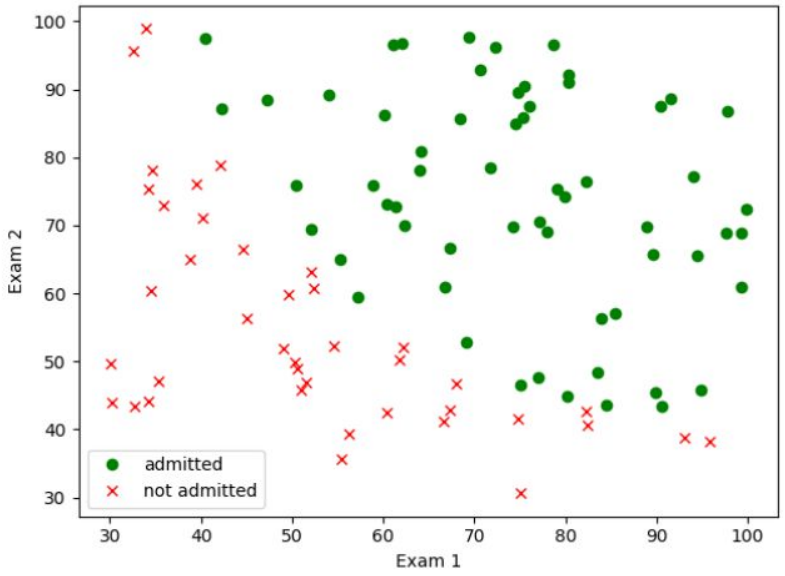

Рисунок 1 - данные о поступлении из файла ex2data1.txt

3. Векторная реализация градиентного спуска и функции потерь для логистической регрессии:

```
@staticmethod
def sigmoid(v):
    return 1.0 / (1 + np.exp(-v))

def cost(self, theta):
    m = self.X.shape[0]
    h = LogisticRegression.sigmoid(np.matmul(self.X, theta))
     cost = np.matmul(-self.Y.T, np.log(h)) - np.matmul((1 - self.Y.T), np.log(1
- h)) / m
    return cost
```

```
def cost_gradient(self, theta):
    m = self.X.shape[0]
    h = self.sigmoid(np.matmul(self.X, theta))
    grad = np.matmul(self.X.T, (h - self.Y)) / m
    return grad
```

4. Добавим возможность производить обучение модели одним из трех методов: методом градиентного спуска, методом Недлера-Мида, алгоритмом Бройдена — Флетчера — Гольдфарба — Шанно. Воспользуемся реализацией двух последних методов из библиотеки sklearn:

```
def train(self, mode='gradient', n_iterations=1000):
    self.model_parameters.fill(0)
    if mode == 'gradient':
        for _ in range(n_iterations):
            grad = self.cost_gradient(self.model_parameters)
            self.model_parameters = self.model_parameters - self.learning_rate * grad
    elif mode == 'nedler':
        res = minimize(self.cost, self.model_parameters, method='nelder-mead',
                       options={'xtol': 1e-4, 'disp': True})
        self.model_parameters = np.array(res.x)
    elif mode == 'bro':
        res=minimize(self.cost,self.model_parameters,method='BFGS',
        jac=self.cost_gradient, options={'disp': True})
        self.model_parameters = np.array(res.x)
```

5. Вычислив параметры модели сможем предсказать вероятность поступления студента в зависимости от его оценок за оба экзамена:

```
def predict(self, x):
    t = self.model_parameters.dot(np.insert(x, 0, 1))
    pred = 1 / (1 + np.exp(-t))
    return pred
```

6. Методом градиентного спуска найдем оптимальные параметры модели. График разделяющей прямой изображен вместе с исходными данными на рисунке 2:

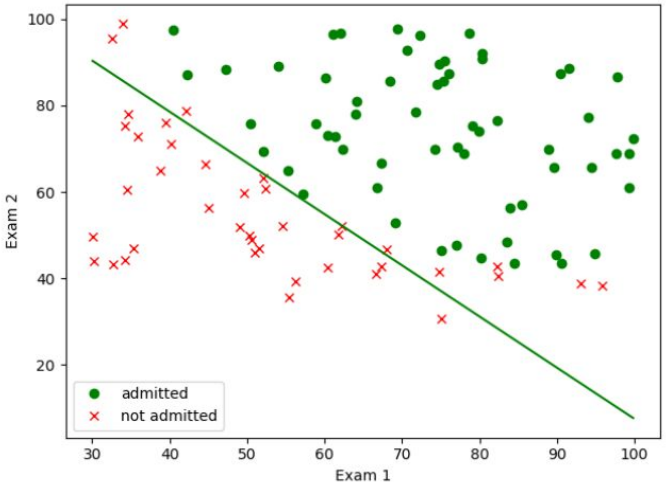

Рисунок 2 - разделяющая прямая и исходные данные

7. Загружаем данные из файла ex2data2.txt используя уже объявленный метод:

```
X, y = read_csv("ex2data2.txt")
```

8. На рисунке 3 изображен график, где по осям откладываются результаты тестов, а точки обозначаются двумя разными маркерами в зависимости от того, прошло ли изделие контроль или нет:

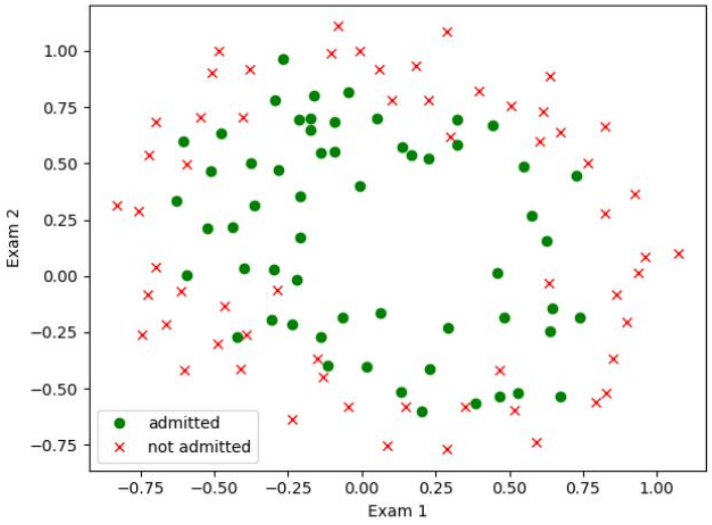

Рисунок 3 - график прохождения тестов изделиями

9. Вычислим все комбинации признаков, в которых степень полинома не превышает 6:

```
def compute_polynom(powers, x_vector):
    p_len = len(powers)
    x_vector = np.repeat(np.array([x_vector]), p_len, axis=0)
    return np.prod(x_vector**powers, axis=1)

powers = []
for i in range(0, polynom + 1):
    for j in range(0, polynom + 1):
        if i + j <= polynom:
            powers.append((i, j))
powers = sorted(powers, key=lambda p: p[0] + p[1])
X = np.array([compute_polynom(powers, x[1:]) for x in X])
```

10. Реализуем L2 регуляризацию для логистической регрессии в функцию вычисления градиента и в функцию стоимости:

```
def cost_gradient(self, theta):
    m = self.X.shape[0]
    h = self.sigmoid(np.dot(self.X, theta))
    grad = np.dot(self.X.T, (h - self.Y)) / m
    grad[1:] = grad[1:] + (self.reg_coeff / m) * theta[1:]
    return grad

def cost(self, theta):
    m = self.X.shape[0]
    h = LogisticRegression.sigmoid(np.matmul(self.X, theta))
    cost = np.matmul(-self.Y.T, np.log(h)) - np.matmul((1 - self.Y.T), np.log(1 - h))
    cost += (1/2) * self.reg_coeff * theta.dot(theta)
    return cost / m
```

11. Проведем тренировку модели методами градиентного спуска (с шагом обучения 15), методом Недлера-Мида и алгоритмом Бройдена:

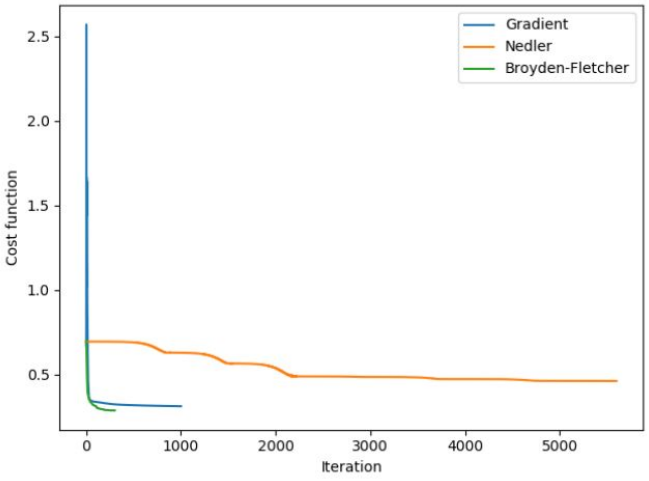

Рисунок 4 - кривые обучения логистической регрессии

12. Ниже приведена реализация функции предсказания вероятности прохождения контроля изделием, в зависимости от результатов тестов:

```
def predict(self, x):
    t = self.model_parameters.dot(self.compute_polynom(x))
    pred = 1 / (1 + np.exp(-t))
    return pred
```

13. Разделяющая кривая, полученная в результате обучения модели с коэффициентом регуляризации 0.0001, приведена на рисунке 5, совместно с исходными данными, которые использовались для обучения.

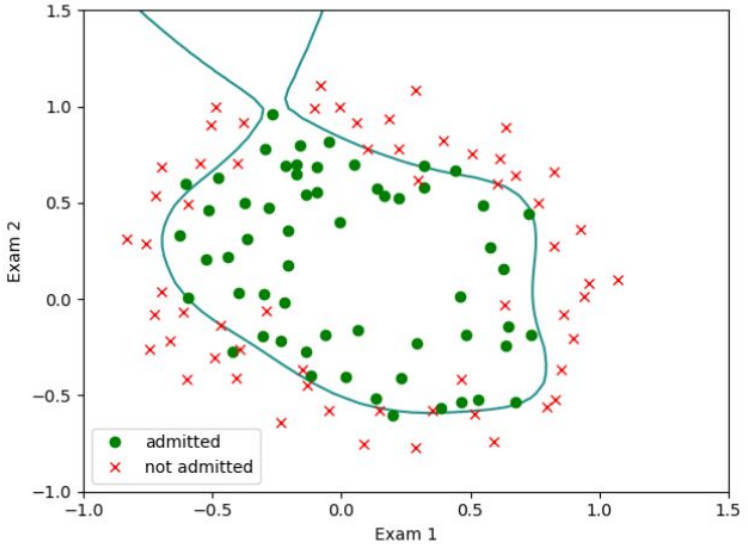

Рисунок 5 - Разделяющая кривая и исходные данные

14. Попробуем тренировать нашу модель, используя различные значения параметра регуляризации: 0.001, 0.01, 1, 10. Сравнение разделяющих кривых приведено на рисунке 6 - 10.

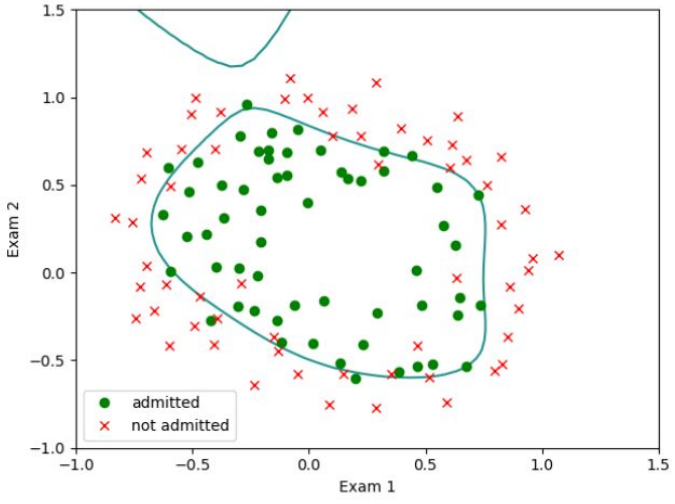

Рисунок 6 - Разделяющая кривая, коэффициент регуляризации 0.001

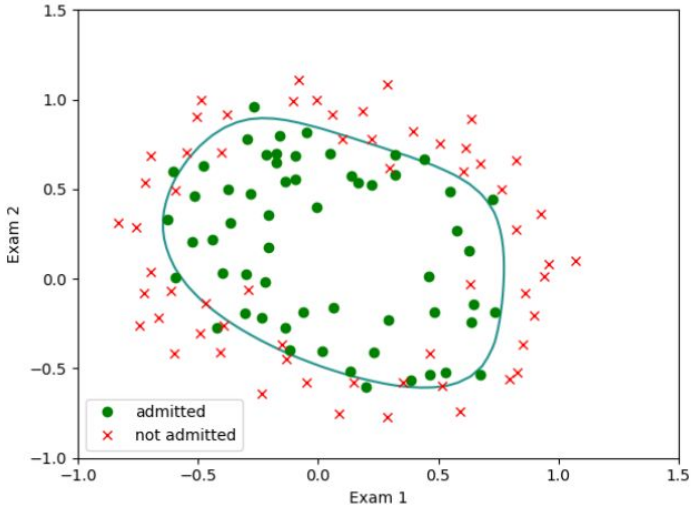

Рисунок 7 - Разделяющая кривая, коэффициент регуляризации 0.01

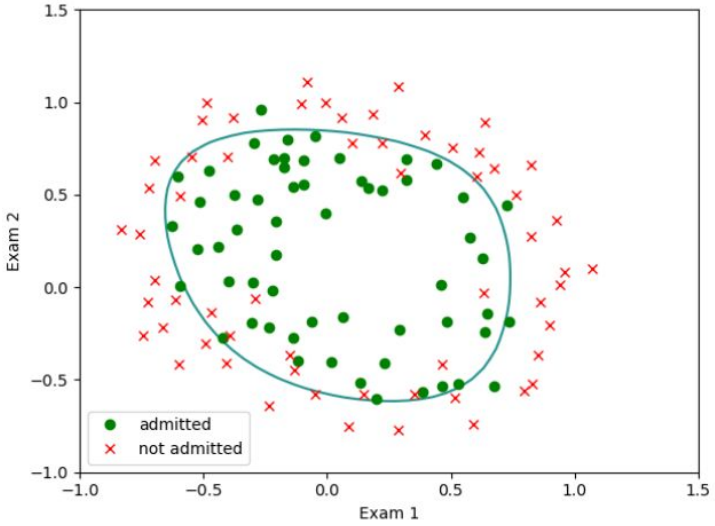

Рисунок 8 - Разделяющая кривая, коэффициент регуляризации 1

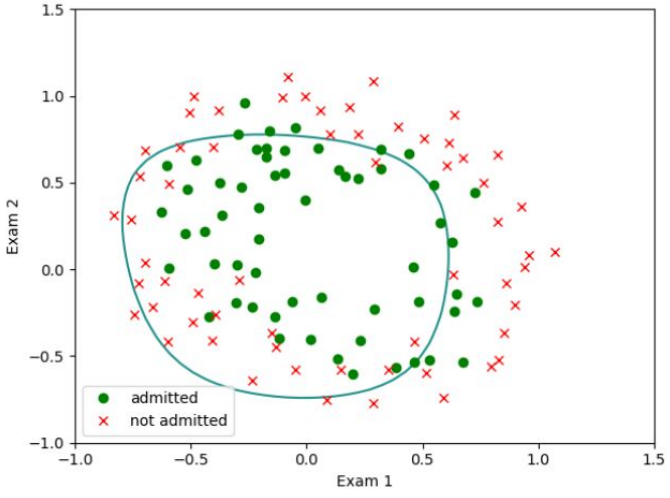

Рисунок 9 - Разделяющая кривая, коэффициент регуляризации 10

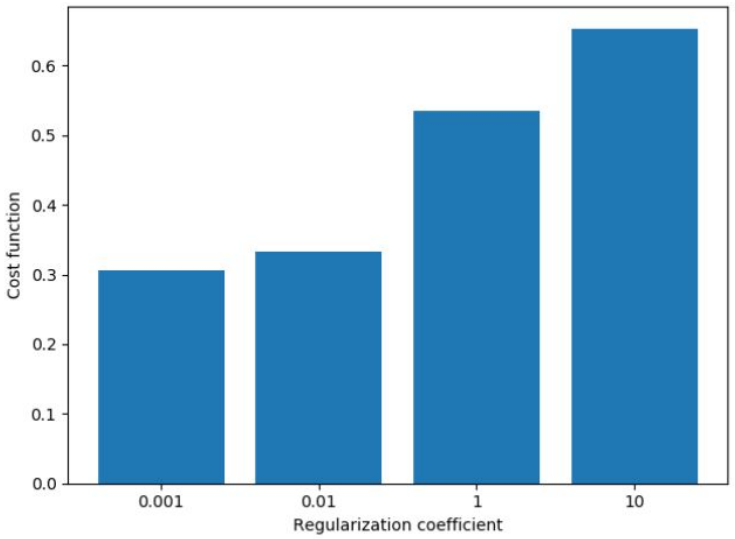

Рисунок 10 - Сравнение значений функции стоимости при различных значениях параметра регуляризации

15. Загружаем данные из файла ex2data3.mat используя метод loadmat:

```
mat = scipy.io.loadmat('ex2data3.mat')
X = mat['X']
y = mat['y']
```

16. На рисунке 11 визуализировано 20 случайных изображений из входного набора данных.

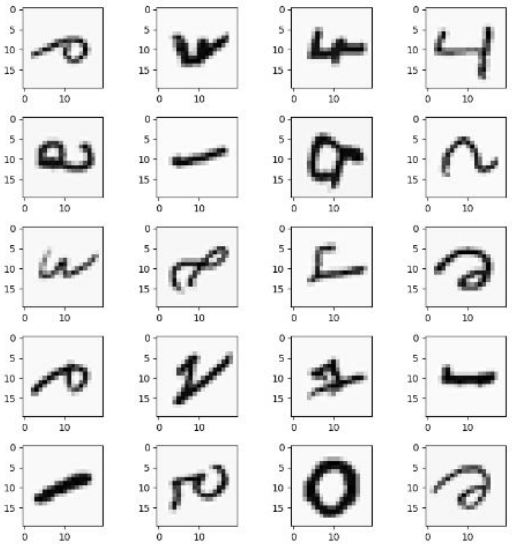

Рисунок 11 - 20 случайных изображений из набора данных

17, 18. Для бинарной классификации мы будем использовать уже реализованные функции стоимости и градиентного спуска, так как они уже были реализованы с использованием векторизации и регуляризации.

```
LogisticRegression(X, sample_y, reg_coeff=0, learning_rate=15)
```

19. Натренируем 10 классификаторов, каждый из которых будет предсказывать вероятность принадлежности изображения к одному определенному классу.

```
classes = list(range(1, 11))
predictors = []
for cls in classes:
    mask = y == cls
    sample_y = np.zeros(len(y))
    sample_y[mask] = 1
    predictor = LogisticRegression(X, sample_y, reg_coeff=0, learning_rate=15)
    predictor.train('gradient', n_iterations=2000)
    predictors.append(predictor)
    print("Class " + str(cls) + " trained.")
```

20. Реализация функции принадлежности к классу:

```
def predict_class(predictors, image):
    predictions = [p.predict(image) for p in predictors]
    index_of_max = np.argmax(predictions)
    return index_of_max + 1
```

21. С шагом обучения 10 и количеством итераций 1500 удалось добиться точности определения класса изображения 93.24%

Выводы

Большое преимущество логистической регрессии в том, что в качестве прогноза модель выдает не бинарное значение, а вероятность принадлежности к определенному классу. Это может иметь большое значение на практике, то есть мы сможем сами определять границу, после которой будем решать относить данные к классу или нет: например, выдавать или не выдавать кредит в зависимости от рейтинга клиента.

Другим полезным свойством является то, что подавая на вход алгоритму полиноминальные признаки, мы можем построить нелинейную границу между классами. И чем лучше натренирована модель, тем более точно граница будет разделять классы. Однако, это может привести к переобучению.

Регуляризация позволяет нам избежать переобучения. Чем больше параметр регуляризации - тем менее точный прогноз на исходном наборе данных мы получаем, но при этом повышаем вероятность того, что предсказания будут верными с другими данными.