Лабораторная работа №6. Множественная классификация.

В данной работе вам предстоит реализовать множественную классификацию (на 10 классов,) с использованием регуляризованной логистической регрессии.

Мы будем использовать обучающий набор изображений рукописных цифр (от 0 до 9) из базы данных MNIST (https://ru.wikipedia.org/wiki/MNIST). База данных содержит изображения размером 20×20 пикселей, каждый пиксель принимает целочисленные значения 0-255 (оттенки серого). Задачей нашей работы является создать классификатор, который получает на вход изображение и должен выдать ответ, какая цифра на нем изображена.

1. Загрузка и отображение данных

На этом шаге вам не потребуется писать собственного кода. В нем загружается файл ex3data1.npy, который содержит обучающий набор данных в виде матрицы *X* и вектора *y* в формате NumPy.

Матрица *X* размером 5000×400 содержит 5000 изображений. Строка матрицы *X* длиной 400 элементов хранит изображение 20×20 (всего 400 пикселей) в следующем виде: сначала 20 чисел значений яркости первой строки изображение, затем второй и т.д. до последней строки. Таким образом вместо матрицы 20×20 один обучающий пример задается вектором из 400 параметров.

Вектор *y* длиной 5000 элементов для каждого обучающего примера X_i хранит индекс класса. Класс y_i=0 соответствует цифре «0», y_i=1 соответствует цифре «1» и т.д., y_i=9 соответствует цифре «9».

После загрузки выбирается случайных 100 изображений из обучающего набора и показываются на экране. Для показа используется функция display_data. Вам не нужно изменять содержимое функции, но рекомендуется ознакомиться с принципом ее работы.

Запустите программу, вы должны увидеть пример обучающих данных. Дальнейшие шаги в программе пока не реализованы.

Удалите оператор return после первого шага в функции main, чтобы перейти ко второму шагу.

2. Регуляризованная логистическая регрессия

На данном шаге вам необходимо написать функции, реализующие регуляризованную логистическую регрессию, аналогично тому, как это было во лабораторной работе №5.

Найдите в программе функцию sigmoid и напишите в ней код для вычисления логистической функции. На вход функция принимает вектор z (произвольной длины). Из функции должен быть возвращен вектор той же самой длины, значение каждого элемента которого должно быть результатом логистической функции от соответствующего элемента входного вектора.

Найдите в программе функцию cost_function и напишите в ней код для вычисления функции стоимости регуляризованной логистической регрессии. Входные параметры функции являются:

theta - вектор параметров функции гипотезы;

X – матрица элементов обучающей выборки;

y – вектор ответов;

lamb – параметр регуляризации.

Функция должна возвращать одно значение: величину функции стоимости для

заданных theta, X, y, lamb.

Найдите в программе функцию gradient_function и напишите в ней код для вычисления градиента (вектора частных производных) функции стоимости регуляризованной логистической регрессии. Входные параметры аналогичны функции cost_function. Функция gradient_function должна возвращать вектор градиента длиной, равной длине строки входной матрицы X.

Рекомендуется реализовывать все функции в матричном виде.

После того, как обе функции будут написаны, запустите программу. В командном окне выводятся тестовые значения стоимости и градиента после запуска написанной вами функции и значения, которые ожидается от нее получить. Если эти значения совпадают, переходите к следующему шагу.

3. Обучение классификатора «один-против-всех»

На этом шаге вам необходимо построить и обучить классификатор по принципу «один-против-всех» для идентификации цифры по ее изображению из обучающего набора. Найдите в программе функцию one_vs_all и дополните его необходимым кодом до рабочего варианта.

Функция принимает на вход следующие параметры:

X – матрица элементов обучающей выборки;

y – вектор ответов;

lamb – параметр регуляризации;

num_labels – количество различных классов в y (10 классов).

Выдавать функция должна обученные модели регуляризованной логистической регрессии для каждого класса (всего num_labels моделей). Напомним, что обученная модель логистической регрессии представляет собой набор параметров θ (вектор длиной n+1, где n – количество признаков в X, в наших данных n=400). В подходе «один-против-всех» нам необходимо обучить столько же моделей, сколько различных классов в задаче (num_labels классов). Таким образом, функция возвращает матрицу all_theta, в которой в каждой *k*-й строке записаны параметры θ отдельной модели для *k*-го класса, а количество столбцов равно количеству параметров модели. То есть, для наших обучающих данных all_theta будет матрицей с размером (num_labels)×(n+1).

Для реализации этой функции вам могут пригодится логические массивы. Например, обучая классификатор для класса 3 нам необходимо подготовить вектор ответов y_3, в котором для каждого элемента из X будут указаны: 1 – если элемент относится к классу 3, 0 – если он относится к любому другому классу. Исходный вектор y нашей обучающей выборки содержит значения от 0 до 9 и не годится для обучения модели. Выражение

```
y3=(y==3)
```

вернет как раз такой вектор, который нам нужен (поэкспериментируйте в консоли Python).

Для оптимизации функции стоимости и нахождения параметров θ каждого классификатора можно воспользоваться функцией minimize из библиотеки scipy.optimize (смотрите пример в работе №5).

Замечание: для ускорения процесса обучения моделей можно указать использование быстродействующего метода Ньютона с усечением (TNC). Для этого в параметрах функции нужно дополнительно указать method='TNC'. Например, так:

```
res = op.minimize(fun=cost_function, x0= theta0, args=(X, y, lamb), method='TNC', jac=gradient_function)
```

Точность результата будет немного меньше, чем у методов без округления, но достаточно высокой. Для сравнения, после того как выполните работу до конца, можете попробовать указать method='BFGS' и посмотреть, как изменились скорость обучения и точность классификации.

После того, как функция готова, запустите программу. Вы должны увидеть процесс обучения. Для оценки правильности обучения необходимо выполнить следующий шаг.

4. Предсказание «один-против-всех»

На данном шаге мы воспользуемся построенным классификатором «один-против-всех», который задан матрицей параметров гипотез all_theta, для того, чтобы предсказать для любого изображения символа, какая цифра на нем показана.

Найдите функцию predict_one_vs_all, необходимо дополнить ее до рабочего состояния. Функция принимает на вход следующие параметры:

X – матрица элементов тестовой выборки;

all_theta – матрица параметров обученных классификаторов, полученная на предыдущем шаге (количество строк в матрице равно num_labels – числу классов).

Возвращать функция должна вектор p с ответами, к какому классу относится каждый элемент из X. Размер p равен числу строк в матрице X, а значение каждого k-го элемента p соответствует идентификатору класса (целое число от 0 до num_labels–1).

Допишите функцию так, чтобы выдавать правильное значение p. Обратите внимание, что X необходимо дополнить слева единичным столбцом. Для определения индекса модели, которая выдает наибольшее значение логистической функции, можно использовать функцию argmax (подробнее в пособии «Практическая работа с NumPy»).

После того, как функция готова, запустите программу. В конце программы вычисляется точность на обучающем наборе. Если она близка к ожидаемой точности, значит работа выполнена правильно.

На этом выполнение лабораторной работы №6 завершается.