Задание № Б-2.1.1. Мультиклассовая классификация

1. Введение

В этом упражнении вы реализуете мультиклассовую[1] логистическую регрессию для распознавания рукописных цифр.

Описание файлов задания:

- ex3.m – главный скрипт для выполнения упражнения,
- ex3data1.mat – обучающая выборка с картинками рукописных цифр,
- displayData.m – функция для визуализации обучающей выборки,
- fmincg.m – минимизирующий функционал,
- sigmoid.m - сигмоида,
- * lrCostFunction.m – функция стоимости для логистической регрессии,
- * oneVsAll.m – обучение мультиклассовой логистической регрессии,
- * predictOneVsAll.m – функция прогноза мультиклассовой логистической регрессии.

(*) – файлы, с которыми вам необходимо работать.

2. Логистическая регрессия

Мы будем использовать логистическую регрессию и нейронную сеть для распознавания рукописных цифр (от 0 до 9). Автоматическое распознавание рукописных символов используется сегодня в большом классе задач: для распознавания почтовых индексов на почтовых конвертах, для распознавания ответов на бланках ЕГЭ и иных формах и т.д. Это упражнение покажет вам, как методы, которые вы изучили, могут быть использованы для решения этой задачи классификации.

В первой части упражнения вы расширите свою предыдущую реализацию логистической регрессии и примените ее для решения задачи классификации по принципу «один ко многим, т.е. для мультиклассовой классификации.

2.1 Обучающая выборка

Вам дана обучающая выборка в файле ex3data1.mat, который содержит 5000 картинок рукописных цифр[2]. Формат .mat означает, что данные были сохранены в собственном матричном формате Octave / MATLAB вместо текстового (ASCII) формата, такого как csv-файл. Эти файлы могут быть считаны с помощью команды load, после чего в памяти вашей программы появятся уже заполненные матрицы правильных размеров. Матрицы уже будут названы, поэтому вам не нужно отдельно присваивать им имена.

```
% загружаем сохраненные матрицы из файла
load('ex3data1.mat');
% теперь матрицы X и y будут в вашем программном окружении Octave/Matlab
```

В файле ex3data1.mat содержится 5000 обучающих примеров, каждый из которых представляет собой изображение цифры в оттенках серого размером 20 на 20 пикселей. Каждый пиксель представлен числом с плавающей точкой, указывающим интенсивность оттенка серого. Сетка пикселей размером 20 на 20 «разворачивается» в 400-мерный вектор. В результате мы получаем матрицу X размером 5000 на 400, где каждая строка является обучающим примером рукописного изображения цифры.

---

[1] Напомним, что мультиклассовая регрессия (классификатор) относит входной объект к одному из K классов.

[2] Это подмножество датасета с рукописными цифрами MNIST (http://yann.lecun.com/exdb/mnist/).

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

Вторая часть обучающей выборки представляет собой 5000-мерный вектор y с метками. В целях удобства работы с индексацией Octave / MATLAB, начинающейся с единицы, мы присвоили цифре ноль метку со значением десять. Следовательно, цифра "0" помечается как "10", в то время как цифры от "1" до "9" помечаются как "1" до "9" в их естественном порядке.

2.2 Визуализация данных

Вы начнете с визуализации подмножества обучающей выборки. Код скрипта ex3.m случайным образом выбирает 100 строк из матрицы X и передает их функции displayData. Эта функция отображает каждую строку в виде изображения в оттенках серого размером 20 на 20 пикселей, при этом все 100 цифр выводятся в виде сетки 10 на 10. Вам уже дана функция displayData – советуем изучить ее код, чтобы понять, как она работает. После выполнения этого шага вы должны увидеть изображение, подобное Рисунку 1.

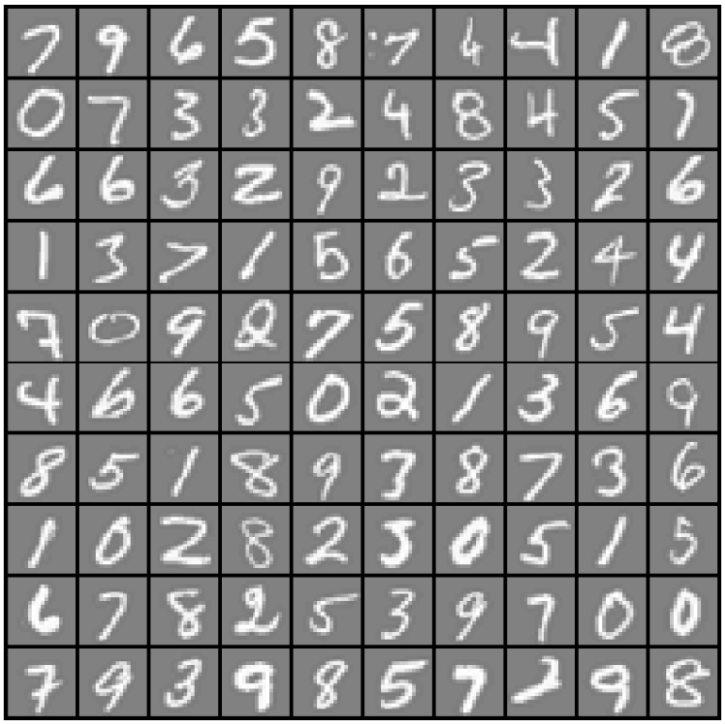

Рисунок 1: Подмножество обучающей выборки

2.3 Векторизация логистической регрессии

Для построения мультиклассового классификатора по модели «один ко многим» вы объедините несколько логистических регрессий в один классификатор. Поскольку существует 10 классов, вам нужно будет обучить 10 отдельных логистических регрессий. Чтобы процесс обучения был эффективным, ваш код должен быть хорошо векторизован. В этом разделе вы реализуете векторизованную версию логистической регрессии, которая не использует никаких циклов for.

2.3.1 Векторизация функции стоимости

Напомним, что нерегуляризованная функция стоимости логистической регрессии выглядит следующим образом

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right].$$

Для того, чтобы вычислить каждый элемент суммы, мы должны сначала вычислить значение гипотезы $h_\theta\left(x^{(i)}\right)$ для каждого i, где $h_\theta\left(x^{(i)}\right) = g(\theta^T x^{(i)})$, а

$$g(z) = \frac{1}{1+e^{-z}}.$$

Мы можем вычислить это очень быстро, используя матричное умножение. Пусть

$$X = \begin{bmatrix} — & (x^{(1)})^T & — \\ — & (x^{(2)})^T & — \\ & \vdots & \\ — & (x^{(m)})^T & — \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

и .

Тогда, вычислив матричное произведение $X\theta$, мы получим

$$X\theta = \begin{bmatrix} — & (x^{(1)})^T\theta & — \\ — & (x^{(2)})^T\theta & — \\ & \vdots & \\ — & (x^{(m)})^T\theta & — \end{bmatrix} = \begin{bmatrix} — & \theta^T(x^{(1)}) & — \\ — & \theta^T(x^{(2)}) & — \\ & \vdots & \\ — & \theta^T(x^{(m)}) & — \end{bmatrix}$$

В последнем уравнении мы использовали тот факт, что $a^T b = b^T a$, если a и b – вектора. Это позволяет нам подсчитать все значения $\theta^T x^{(i)}$ для всех обучающих примеров с помощью одной строки кода (которая в системах матричных вычислений будет благодаря параллельным вычислениям и использованию эффективных матричных алгоритмов вычисляться на несколько порядков быстрее, чем то же самое, написанное с помощью циклов).

Ваша задача – написать функцию стоимости нерегуляризованной логистической регрессии в файле lrCostFunction.m, используя полностью векторизованный подход, при котором в файле lrCostFunction.m не должно быть никаких циклов. После этого запустите скрипт ex3.m и убедитесь, что ваш ответ («Стоимость») совпадает с корректным («Ожидаемая стоимость»).

2.3.2 Векторизация градиента

Напомним, что градиент нерегуляризованной функции стоимости логистической регрессии – это вектор, в котором j-тая компонента определяется по формуле:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}.$$

Выпишите вектор значений градиента в явном виде

$$\begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} \sum_{i=1}^{m} \left((h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)} \right) \\ \sum_{i=1}^{m} \left((h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)} \right) \\ \sum_{i=1}^{m} \left((h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)} \right) \\ \vdots \\ \sum_{i=1}^{m} \left((h_\theta(x^{(i)}) - y^{(i)}) x_n^{(i)} \right) \end{bmatrix}$$

и попробуйте выразить его через матричные операции над имеющимися векторами и матрицами. В вашем коде не должно быть никаких циклов. Напишите полученные формулы в файле lrCostFunction.m. После этого запустите скрипт ex3.m, чтобы протестировать вашу реализацию. Сравните ваши значения градиента («Градиенты») с правильными («Ожидаемые градиенты»).

Совет по отладке: Векторизация кода с непривычки может быть сложной задачей. Одна из распространенных стратегий отладки кода – выписать размеры матриц, с которыми вы работаете. Например, если матрица X имеет размеры 100×20, а вектор θ – размеры 20×1, то мы сразу видим, что $X\theta$ является допустимой операцией умножения, а θX нет. Прежде чем приступать к написанию кода, **изучите размерности входных и выходных данных**. Для вашего удобства все выходные переменные также объявляются и инициализируется в начале каждого файла.

2.3.3 Векторизация регуляризованной логистической регрессии

Теперь добавим регуляризацию. Вспомним, что функция стоимости в этом случае будет выглядеть так

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2.$$

Помните, что вам **не надо** регуляризовать параметр θ_0.

Соответственно, частные производные функции стоимости в этом случае принимают вид

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)},$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j, \qquad \text{для } j \geq 1.$$

Измените свой код в lrCostFunction, чтобы учесть регуляризацию. По-прежнему, вы не должны использовать никаких циклов. После этого запустите скрипт ex3.m и убедитесь, что вы получаете корректные значения регуляризованной версии стоимости и градиента.

Подсказка по программированию в Octave/Matlab. При векторизации регуляризованной логистической регрессии часто может потребоваться суммировать и обновлять только определенные элементы θ (например, чтобы не трогать θ_0). Матричный язык Octave / MATLAB позволяет вам с помощью операции индексирования обновлять только определенные элементы матриц. Например, A(:, 3:5) = B(:, 1:3) заменит столбцы с 3 по 5 в A столбцами с 1 по 3 из B. Одно специальное ключевое слово, которое вы можете использовать при индексации: end – оно означает индекс последнего элемента. Это позволяет вам обращаться к столбцам (или строкам) с какой-то позиции и до конца матрицы.

2.4 Классификация «один ко многим»

В этой части упражнения вы реализуете классификацию по принципу «один ко многим» путем обучения нескольких регуляризованных логистических регрессий, по одному для каждого из K классов в нашей обучающей выборке. В нашем конкретном случае K = 10, но ваш код должен работать для любого значения K.

Допишите код в oneVsAll.m, чтобы обучить по одному классификатору на каждый класс. В частности, ваш код должен возвращать все параметры класса в матрице $\theta \in \mathbb{R}^{K\times(N+1)}$, где каждая строка θ соответствует найденным параметрам логистической регрессии для одного класса. Вы можете сделать это с помощью цикла for, обучая каждый класс независимо.

Обратите внимание, что аргумент y для этой функции представляет собой вектор меток от 1 до 10, где мы сопоставили цифру 0 с меткой 10 (чтобы избежать путаницы с индексацией). В то время как при обучении классификатора для класса $k \in \{1, \dots, K\}$ вам понадобится m-мерный вектор меток y, где $y_j \in \{0,1\}$ указывает, принадлежит ли j-й обучающий пример классу k ($y_j = 1$) или нет ($y_j = 0$). Возможно, вы найдете полезными в этой связи логические матрицы.

Подсказка по программированию в Octave/Matlab. Логические матрицы в Octave/Matlab – это матрицы из нулей и единиц. Результатом вычисления выражения $a == b$, где a – вектор, а b – скаляр, является вектор той же размерности, что и a, в котором на i-том месте будет 1, если $a[i] = = b$, и 0 в противном случае. Попробуйте выполнить следующий код, чтобы понять, как они работают

```
a = 1:10; % создаем a и b
b = 3;
a == b % попробуйте разные значения b
```

Кроме того, в этом упражнении вы будете использовать fmincg (вместо fminunc). Функция fmincg работает аналогично fminunc, но она более эффективна при работе с большим количеством параметров.

После того, как вы завершите работу с функцией oneVsAll, скрипт ex3.m использует ее для обучения мультиклассового классификатора.

2.4.1 Работа классификатора «один ко многим»

После обучения вашего мультиклассового классификатора вы можете использовать его для распознавания цифр на изображениях. Для этого вы должны, используя обученные логистические регрессии, вычислить для каждого входного вектора «вероятность» того, что он принадлежит к каждому из классов. Ваша функция гипотезы выберет класс, для которого соответствующая логистическая регрессия выдаст наибольшую вероятность, и вернет метку этого класса (1, 2,... или K) в качестве ответа.

Допишите код в predictOneVsAll.m соответствующим образом. Когда вы закончите, скрипт ex3.m вызовет вашу функцию predictOneVsAll и вы должны получить точность работы классификатора около 94,9% (т.е. ваша модель правильно классифицирует 94,9% примеров из обучающей выборки).