Задание № Б-1.2. Логистическая регрессия

1. Введение

В этом упражнении вы реализуете логистическую регрессию и примените ее к двум различным наборам данных.

Описание файлов задания:

- ex2.m – главный скрипт для выполнения первой части упражнения,
- ex2_reg.m - скрипт для выполнения второй части упражнения,
- ex2data1.txt - датасет для первой части,
- ex2data2.txt - датасет для второй части,
- mapFeature.m – функция для генерации полиномиальных признаков,
- plotDecisionBoudary.m – функция для отрисовки решающей границы классификатора,
- * plotData.m - функция для отрисовки обучающей выборки,
- * sigmoid.m - сигмоида,
- * costFunction.m – функция стоимости для логистической регрессии,
- * predict.m – гипотеза логистической регрессии,
- * costFunctionReg.m – регуляризованная функция стоимости логистической регрессии.

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

На протяжении всего упражнения вы будете запускать только скрипты ex2.m и ex2_reg.m. Они подготавливают датасеты для разных подзадач и выполняют вызовы всех нужных функций. Изменять их содержимое не надо. От вас требуется только написать код функций в файлах, помеченных *.

2. Логистическая регрессия

В этой части упражнения вы построите модель логистической регрессии, позволяющую предсказывать, поступит ли студент в университет.

Предположим, что руководство факультета попросило вас определить шансы каждого абитуриента на поступление на основе их результатов за два экзамена – математике и информатике. У вас есть данные за предыдущие годы, которые вы можете использовать в качестве обучающей выборки для логистической регрессии. Для каждого учебного примера у вас есть баллы абитуриента по двум экзаменам и решение о приеме.

Ваша задача состоит в том, чтобы построить модель классификатора, который позволяет оценивать вероятность поступления абитуриента на основе его результатов за эти два экзамена.

2.1 Визуализация данных

Прежде чем приступить к выполнению какой-либо задачи, часто бывает полезно визуализировать данные, если это возможно. В начале своей работы скрипт ex2.m загрузит данные и отобразит их на двумерном графике, вызвав функцию plotData. Вам требуется дописать код в plotData.m так, чтобы он рисовал что-то похожее на Рисунок 1, где по осям заданы экзаменационные баллы, а положительные (“Admitted”) и отрицательные (“Not admitted”) примеры показаны разными маркерами.

Чтобы помочь вам попрактиковаться в написании кода для визуализации данных, файл plotData.m оставлен пустым. Попробуйте реализовать указанную функцию самостоятельно. После того, как вам это удастся сделать (или, наоборот, если вы застрянете в пути), можете посмотреть на представленную реализацию ниже. Если вы решите скопировать этот код, убедитесь, что вы понимаете, что делает каждая команда.

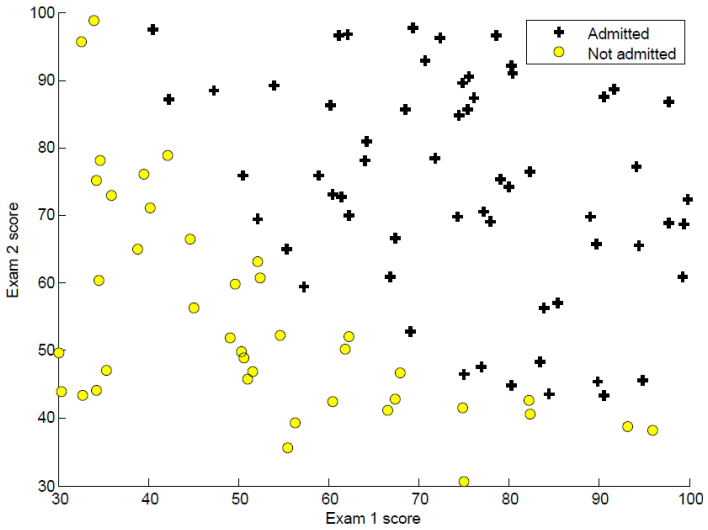

Рисунок 1: Точечный график обучающей выборки

```
% находим индексы положительных и отрицательных примеров
pos = find(y == 1);
neg = find(y == 0);

% визуализируем данные
plot(X(pos, 1), X(pos, 2), 'k+','LineWidth', 2, 'MarkerSize', 7);
plot(X(neg, 1), X(neg, 2), 'ko', 'MarkerFaceColor', 'y', 'MarkerSize', 7);
```

2.2 Реализация

2.2.1 Функция сигмоиды

Напомним, что гипотеза логистической регрессии определяется как:

$$h_\theta(x) = g(\theta^T x),$$

где g – это сигмоида:

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

Ваш первый шаг – реализовать эту функцию в sigmoid.m, чтобы она могла быть вызвана остальной частью вашей программы. Когда вы закончите, попробуйте ее протестировать на нескольких значениях, вызвав sigmoid(x) в командной строке Octave/MATLAB. При больших положительных значениях x сигмоида должна быть близка к 1, в то время как при больших отрицательных значениях она должна быть близка к 0. Значение sigmoid(0) должно быть в точность ровно 0,5. Ваш код также должен работать с векторами и матрицами. **Для матрицы ваша функция должна вычислять значение сигмоиды поэлементно, т.е. ее результатом тоже будет матрица.**

2.2.2 Функция стоимости и градиент

Теперь вы реализуете функцию стоимости и градиент для логистической регрессии.

Допишите код в costFunction.m, чтобы он вычислял значение самой функции и ее градиента. Напомним, что функция стоимости для логистической регрессии равна

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

а ее градиент – это вектор той же размерности, что и θ, в котором j-тая компонента определяется по формуле:

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

Обратите внимание, что, хотя этот градиент выглядит идентично градиенту для линейной регрессии, формула на самом деле отличается, поскольку линейная и логистическая регрессия имеют разные определения $h_\theta(x)$.

Когда закончите, запустите скрипт ex2.m – он вызовет вашу функцию стоимости, используя некоторое начальное значение вектора параметров θ. Вы должны получить значение около 0,693.

2.2.3 Обучение классификатора с использование `fminunc`

В предыдущем задании вы находили оптимальные значения параметров модели линейной регрессии с помощью градиентного спуска. Для этого вы определили функцию стоимости, вычислили ее градиент, а затем соответствующим образом выполняли шаг градиентного спуска.

На этот раз для решения аналогичной задачи вы будете использовать встроенную функцию Octave / MATLAB под названием fminunc.

Функция fminunc в Octave / MATLAB – это решатель для задач оптимизации, который находит безусловный минимум некоторой функции. Для логистической регрессии вы хотите оптимизировать функцию стоимости $J(\theta)$, то есть найти θ, на котором она достигает своего минимума. Функция fminunc принимает следующие параметры:

- начальное значение аргумента оптимизируемой функции,
- функцию, вычисляющую значение оптимизируемой функции и ее градиента.

В ex2.m уже содержится код, вызывающий fminunc с правильными аргументами.

```
% задаем опции для функции fminunc
options = optimset('GradObj', 'on', 'MaxIter', 400);

% вызываем fminunc; результат – оптимальное значение theta
% и соответствующая ему стоимость
[theta, cost] = fminunc(@(t)(costFunction(t, X, y)), initial_theta, options);
```

В этом фрагменте кода мы сначала задаем параметры функции fminunc. В частности, для параметра gradobj мы устанавливаем значение on, что сообщает fminunc, что наша функция возвращает как стоимость, так и градиент. Это позволяет fminunc использовать градиент при минимизации функции. Кроме того, мы устанавливаем параметр MaxIter равным 400, так что fminunc будет выполнять не более 400 шагов.

Чтобы указать фактическую функцию, которую мы минимизируем, мы используем лямбда-выражение с параметром t, которое вызывает costFunction(t, X, y).

Если вы правильно реализовали costFunction, то fminunc сойдется и вернет конечные значения стоимости и θ. Обратите внимание, что в этом упражнении вы используете **встроенный решатель** – вам не нужно было самостоятельно писать какие-либо циклы, определять значения мета-параметров, как вы это делали для градиентного спуска. Все это делает fminunc: вам нужно только предоставить ему функцию стоимости и градиент.

Как только fminunc завершится, ex2.m вызовет costFunction с найденным значением θ. Если все реализовано корректно, вы должны увидеть результат, равный около 0,203.

Вычисленное значение θ затем используется для построения графика решающей границы (Рисунок 2). Рекомендуем вам ознакомиться с кодом в plotDecisionBoundary.m, чтобы узнать, как строится этот график.

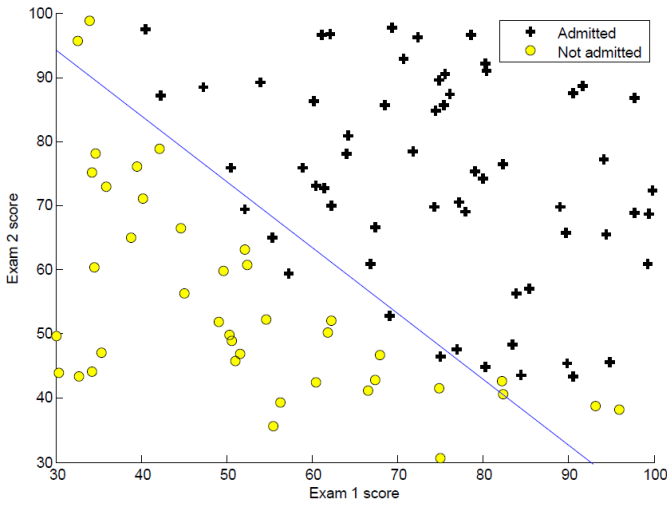

Рисунок 2: Обучающая выборка с решающей границей логистической регрессии

2.2.4 Предсказание

Обучив модель, вы теперь можете ее использовать для прогнозирования вероятности того, поступит ли конкретный студент. Например, для студента, набравшего 45 баллов на экзамене 1 и 85 баллов на экзамене 2, вы должны получить вероятность поступления около 0,776.

Другой способ оценить качество найденных нами параметров – это посмотреть, насколько хорошо обученная модель предсказывает результаты на обучающей выборке. В этой части упражнения ваша задача – завершить код в predict.m. Функция predict должна выдавать 1 или 0 в соответствии со значением найденной гипотезы.

После того, как вы завершите код в predict.m, скрипт ex2.m сообщит о точности обучения вашего классификатора, вычислив процент примеров, которые он классифицировал правильно.

3. Регуляризованная логистическая регрессия

В этой части упражнения вы будете применять регуляризованную логистическую регрессию, чтобы предсказывать, проходят ли микрочипы с завода проверку качества на основе результатов различных тестов.

Предположим, вы являетесь менеджером по качеству на заводе, и у вас есть результаты тестирования некоторых микрочипов на двух разных тестах. На их основе вы хотели бы определить, следует ли принимать микрочипы или отбраковать их. В качестве обучающей выборки для логистической регрессии вы будете использовать данные о результатах тестирования прошлых микрочипов.

В этой части упражнения вам необходимо использовать второй скрипт – ex2_reg.m.

3.1 Визуализация данных

Как и в предыдущих частях этого упражнения, plotData используется для создания графика данных (Рисунок 3), где осями являются результаты двух тестов, а положительные (y = 1) и отрицательные (y = 0) примеры показаны разными маркерами.

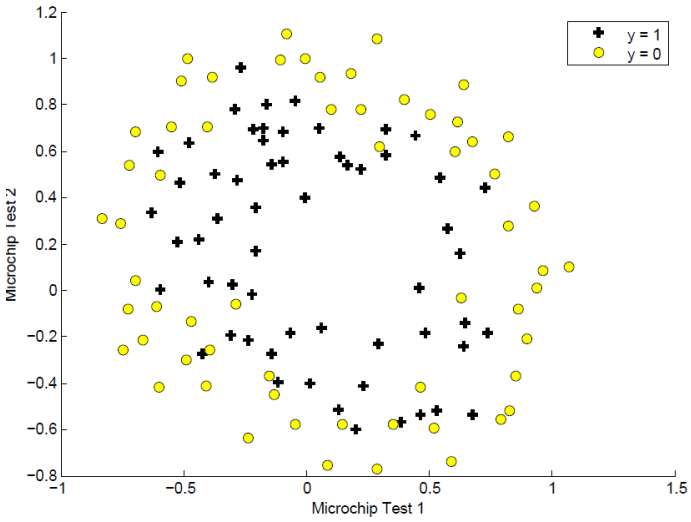

Рисунок 3: Визуализация обучающей выборки

Рисунок 3 показывает, что наш набор данных не может быть разделен на положительные и отрицательные примеры прямой линией (то есть наши данные в этом пространстве признаков не являются **линейно сепарабельными**). Следовательно, непосредственное применение логистической регрессии не даст хорошего результата, поскольку логистическая регрессия умеет находить только гиперплоскость (в нашем двумерном случае – прямую).

3.2 Маппинг признаков (Feature Mapping)

Один из способов повысить точность предсказания модели – создать больше признаков для каждой точки обучающей выборки. Функция mapFeature.m для каждого объекта создает все члены полинома от x_1 и x_2 вплоть до шестой степени.

$$mapFeature(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_1^3 \\ \dots \\ x_2^6 \end{bmatrix}$$

В результате «маппинга» наш вектор из двух признаков (результаты двух тестов) был преобразован в 28-мерный вектор. Решающая граница классификатора, обученного на этом пространстве признаков, если ее спроецировать обратно в двумерное пространство x_1 и x_2, будет выглядеть нелинейной.

Однако мы должны помнить, что чем больше признаков мы вводим в нашу модель, тем больше она становится подвержена проблеме переобучения. В следующих частях упражнения вы реализуете регуляризированную логистическую регрессию, а также сами увидите, как регуляризация помогает в борьбе с переобучением.

3.3 Функция стоимости и градиент

Напишите код в файле costFunctionReg.m для вычисления функции стоимости и градиента для регуляризированной логистической регрессии.

Как мы знаем, регуляризированная функция затрат в логистической регрессии имеет вид

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

Обратите внимание, что вы не должны регуляризировать параметр θ_0.

Градиент функции затрат:

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

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

Когда вы закончите, ex2_reg.m вызовет вашу функцию costFunctionReg, используя начальное значение θ (инициализированное всеми нулями). Вы должны получить значение функции стоимости около 0,693.

3.3.1 Обучение модели с помощью fminunc

Вызовите скрипт ex2_reg.m – он использует fminunc для нахождения оптимального значения вектора параметров θ.

3.4 Построение графика решающей границы

Чтобы помочь вам визуализировать модель, в файле plotDecisionBoundary.m представлена функция, которая рисует (нелинейную) решающую границу, разделяющую положительные и отрицательные примеры. Для этого мы вычисляем предсказания классификатора на равномерно распределенной сетке, а затем рисуем контурную линию, на которой предсказания меняются с y = 0 на y = 1 (Рисунок 4).

3.5 Изучение регуляризации

В заключительной части упражнения вы изучите эффект влияния регуляризации на обобщающую способность классификатора. Для этого вам необходимо построить три графика для трех различных значений параметра λ, отвечающего за степень регуляризации.

Обратите внимание на поведение решающей границы в зависимости от λ. Если значение параметра маленькое, то вы должны обнаружить, что эффект регуляризации проявляется слабо и классификатор корректно обрабатывает почти каждый пример из обучающей выборки, однако при этом «рисует» очень сложную решающую границу (Рисунок 5). Эта модель является переобученной, в результате чего она выдает некорректные результаты на новых точках: например, она предсказывает, что точка x = (-0.25, 1.5) принадлежит положительному классу (y = 1), что, скорее всего, не соответствует действительности.

При слишком большом значении параметра λ вы должны увидеть более простую границу (Рисунок 6), но смещенную относительно «корректной» позиции. Говорят, что модель является недообученной, или с высоким смещением (high bias).

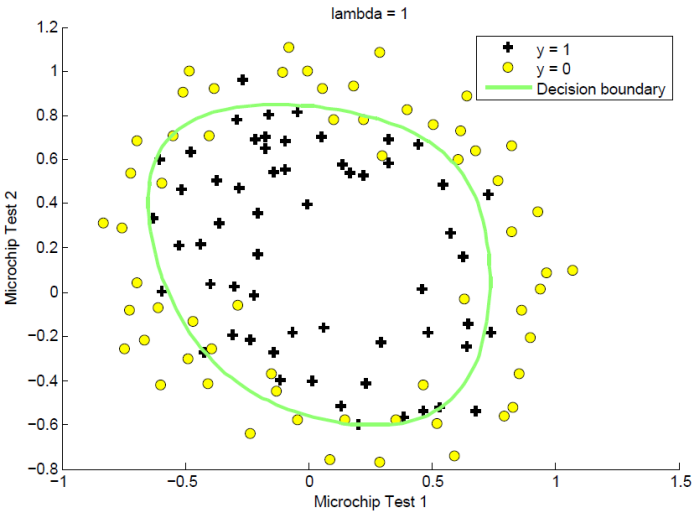

Рисунок 4: Обучающая выборка с решающей границей ($\lambda = 1$)

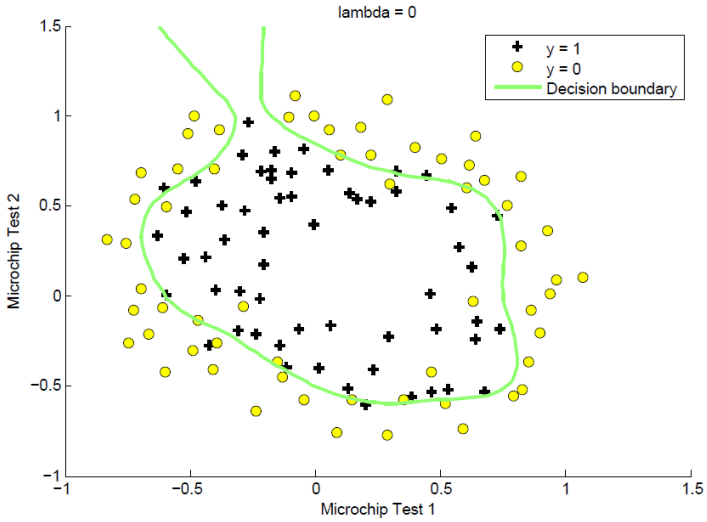

Рисунок 5: Нет регуляризации, переобучение ($\lambda = 0$)

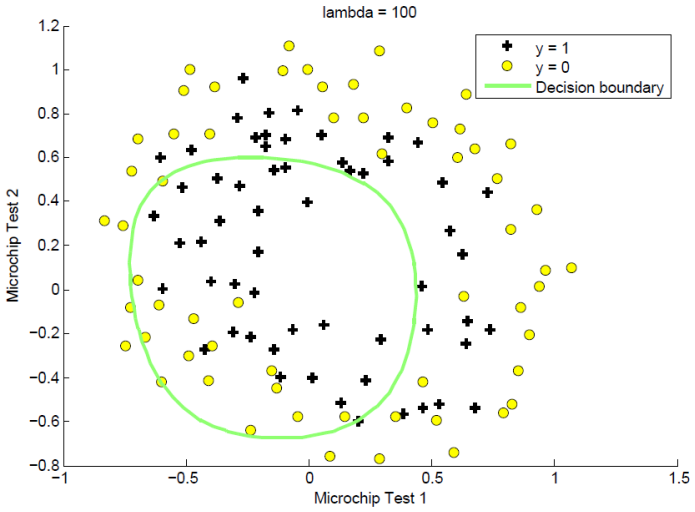

Рисунок 6: Слишком много регуляризации, недообучение ($\lambda = 100$)