CEFET/RJ
Programa de Pós-graduação
em Ciência da Computação
Aprendizado de Máquina - Trabalho 01

Prof. Eduardo Bezerra (ebezerra@cefet-rj.br)

2018

Conteúdo

1 Regressão Linear com uma Variável

Nessa parte do trabalho, você irá implementar a regressão linear para predizer o lucro para uma cadeia de *food truck*. Essa cadeia já possui diversas filiais em diferentes cidades. Você possui dados do lucro e população para cada uma dessas cidades.

O arquivo `ex1data1.txt` contém os dados a serem usados nessa parte do trabalho. A primeira coluna corresponde à população de cada cidade, enquanto que a segunda coluna corresponde ao lucro da filial daquela cidade. Um valor negativo para o lucro indica que a filial correspondente está dando prejuízo.

1.1 Visualização dos Dados

Para a maioria dos conjuntos de dados do mundo real, não é possível criar um gráfico para visualizar seus pontos. Mas, para o conjunto de dados fornecido nesse trabalho, isso é possível. Use o script `plot_ex1data1.py` para gerar um *gráfico de dispersão* (*scatter plot*) dos dados fornecidos. Esse script deve produzir um resultado similar ao apresentado na Figura 4.

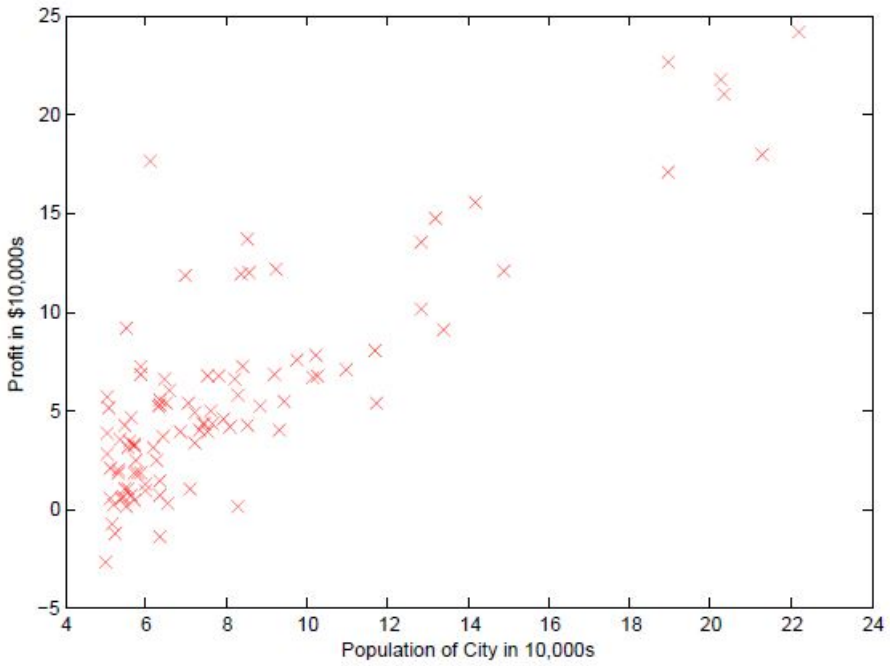

Figura 1: Pontos de dados do conjunto `ex1data1.txt`.

1.2 Gradiente Descendente

Nessa parte, sua tarefa é determinar os parâmetros do modelo de regressão linear por meio do algoritmo Gradiente Descendente. Use a versão *batch gradient descendente* desse algoritmo. Inicie todos os parâmetros com o valor 0 (zero). Além disso, use o valor 0,01 para a taxa de aprendizado. Crie uma função em

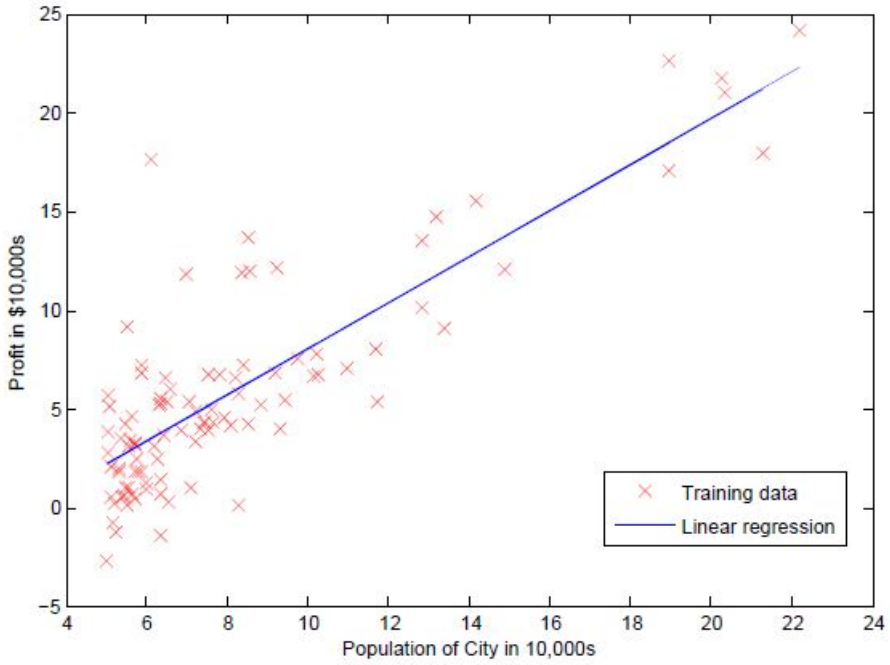

Figura 2: Pontos de dados do conjunto `ex1data1.txt`.

Python denominada `custo_regrlin`. Essa função deve ser definida no arquivo `custo_reglin_uni.py`. Após implementar essa função, você pode verificar a corretude executando com todos os parâmetros iguais a zero. Nessa situação, sua função deve gerar um valor aproximadamente igual a 32,07.

Após estudar o cálculo da função de custo $J(\theta)$, você deve analisar o código do GD propriamente dito. Essa implementação é fornecida no arquivo `gd_reglin_uni.py`. Repare que essa implementação chama a função `custo_reglin` de forma apropriada.

Presumindo que o GD e o cálculo da função de custo foram implementados corretamente, os valores sucessivos do gradiente nunca devem crescer. Além disso, o valor de $J(\theta)$ deve convergir no fim da execução do algoritmo. Em particular, realize a execução do bloco de código apresentado na Listagem 1 e confirme que ele produz um valor aproximado de 4,47. Em seu relatório, explique cada passo do código apresentado nessa listagem.

Listing 1: Teste da função `custo_reglin`.

```
custo, theta = custo_reglin(x, y, 0.01, 5000)
print(custo)
```

Agora, utilize o script `visualizar_reta.py` para visualizar a reta correspondente aos parâmetros determinados. O resultado da execução desse script deve ser similar ao apresentado na Figura 2.

Como última tarefa nessa parte do trabalho, você deve usar o modelo de regressão linear produzido pelo seu código para predizer o lucro em regiões com populações de 35.000 e 70.000 habitantes. Forneça no seu relatório o código

(em Python) para isso, assim como os valores correspondentes do lucro para cada uma daquelas duas cidades.

1.3 Visualização de $J(\theta)$

Para melhor entender a função de custo, você irá nessa parte do trabalho plotar valores da função de custo J sobre uma grade bidimensional de valores de θ_0 e de θ_1. Para isso, você deve usar os scripts `visualizar_J_countour.py` e `visualizar_J_surface.py`. Esses scripts geram um array bidimensional de valores de $J(\theta)$. Os valores gerados estão contidos na faixa a seguir: $-10 \leq \theta_0 \leq +10$ e $-1 \leq \theta_0 \leq +4$. Um incremento de 0,01 é utilizado para gerar os valores de θ_0 e de θ_1. A seguir, esses scripts produzem um gráfico de curvas de contorno (*contour plot*) e um gráfico da superfície (*surface plot*) correspondentes a $J(\theta)$.

O objetivo dos gráficos gerados é fornecer alguma intuição acerca de como $J(\theta)$ varia com os valores de θ_0 e θ_1. A função $J(\theta)$ tem a forma de uma bacia e possui um único mínimo global. Esse mínimo corresponde à combinação ótima de valores para θ_0 e θ_1. Cada passo do algoritmo gradiente descendente se aproxima cada vez mais desse ponto ótimo.

2 Regressão Linear com Múltiplas Variáveis

Nessa parte do trabalho, você irá estudar uma implementação da regressão linear com múltiplas variáveis para predizer o preço de venda de imóveis. O arquivo `ex1data2.txt` contém informações acerca de preços de imóveis. A primeira coluna corresponde ao tamanho do imóvel (em pés quadrados[1]). A segunda coluna corresponde à quantidade de dormitórios no imóvel em questão. A terceira coluna corresponde ao preço do imóvel.

2.1 Normalização das características

Se você inspecionar os valores do conjunto de dados fornecido, irá notar que os valores na coluna relativa aos tamanhos dos imóveis são aproximadamente 1000 vezes maiores do que as quantidades encontradas na coluna de quantidade de dormitórios. Sua tarefa nessa parte é analisar a implementação da função `normalizar_caracteristica` no arquivo `normalizacao.py`. Essa função recebe a matriz de dados X de dados como parâmetro (na forma de um *numpy array*). Essa função realiza dois passos principais:

- subtrai o valor médio de todas as características do conjunto de dados X.
- após subtrair a média, divide cada característica pelo seu respectivo desvio padrão.

A implementação da função `normalizar_caracteristica` deve funcionar com conjuntos de dados de variados tamanhos (i.e., com qualquer quantidade

[1] 1 pé corresponde a 0,3048 metros.

de características e/ou exemplos). Em seu relatório, forneça uma justificativa para essa afirmação. Por fim, repare que cada coluna da matriz de dados X passada para a função `normalizar_caracteristica` deve corresponder a uma característica.

Nota de Implementação: Ao normalizar as características, é importante armazenar os valores utilizados para a normalização - o valor médio e o desvio padrão utilizados para a normalização. A razão para isso é que, depois de aprender os parâmetros do modelo, muitas vezes queremos prever os preços das casas cujos dados não foram vistos durante o treinamento. Dado um novo exemplo x (área da sala de estar e número de quartos), devemos normalizar x usando a média e o desvio padrão previamente calculados no conjunto de treinamento.

2.2 Gradiente descendente

Anteriormente, você analisou a implentação do GD para realizar regressão linear univariada. A única diferença agora é que há mais uma característica na matriz de dados X. A função hipótese $h_\theta(x)$ e a atualização dos gradientes em lote permanecem inalteradas. Você deve implementar código nos arquivos denominados `custo_reglin_multi.py` e `gd_reglin_multi.py` para implementar a função de custo e o algoritmo GD para regressão linear com múltiplas variáveis, respectivamente. Se o seu código na parte anterior (variável única) já provê suporte a múltiplas variáveis, você também pode reusá-lo aqui.

Repare que o código fornecido (1) dá suporte a qualquer número de características e (2) está vetorizado. Explique em seu relatório de que forma esse código implementa essas duas funcionalidades.

3 Regressão Logística

Nessa parte do trabalho, você irá executar um classificador para predizer se um estudante será admitido em uma universidade, com base nos resultados de duas avaliações. O arquivo `ex2data1.txt` contém os dados a serem usados. Nesse arquivo, estão disponíveis dados históricos acerca de realizações passadas dessas avaliações, e que esses dados históricos podem ser usados como conjunto de treinamento. Para cada exemplo desse conjunto de treinamento, temos as notas das duas avaliações e a decisão acerca do candidato (aprovado ou reprovado).

Sua tarefa é construir um modelo de classificação que provê uma estimativa da probabilidade de admissão de um candidato, com base na notas que ele obteve nas duas avaliações.

3.1 Visualização dos dados

Antes de começar a implementar qualquer algoritmo de aprendizado, é adequado visualizar os dados, quando possível. Nessa parte do trabalho, você deve carregar o arquivo com o conjunto de treinamento e plotar (i.e., produzir um

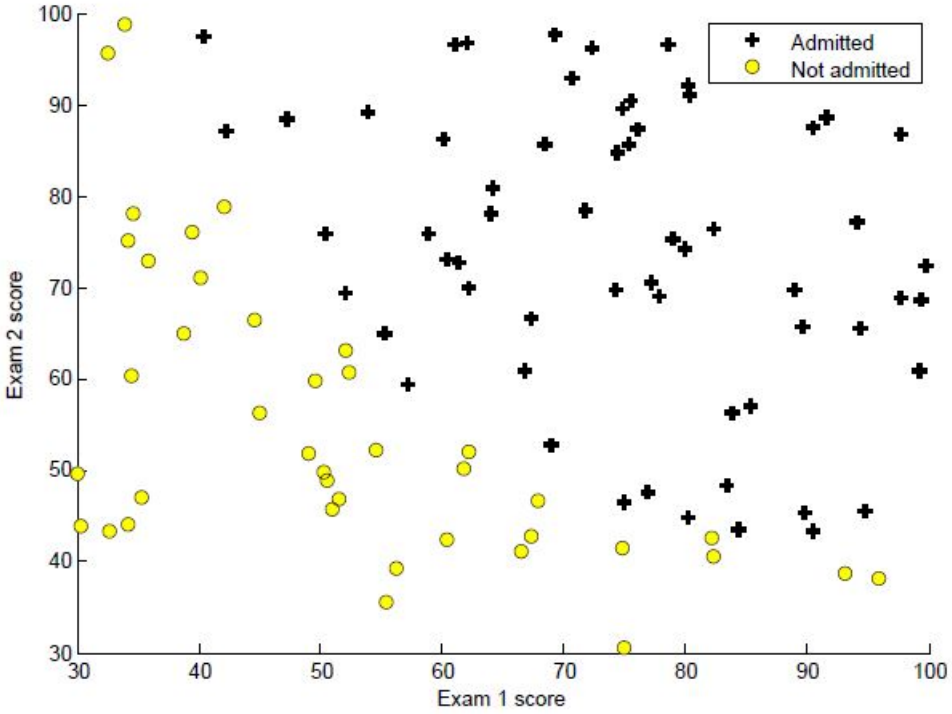

Figura 3: Pontos de dados do conjunto `ex2data1.txt`.

gráfico) os pontos de dados. O resultado dessa tarefa deve ser um gráfico similar ao apresentado na Figura 3. Para isso, use o código fornecido no script `plot_ex2data1.py`.

3.2 Implementação

3.2.1 Função sigmoide

Como primeiro passo nessa parte, realize testes com a função fornecida no arquivo `sigmoide.py`, de forma que ela possa ser chamada de outras parte do seu código. Se você chamar `sigmoide(0)`, o valor retornado deve ser 0,5. Se você chamar essa função com valores muito grandes e positivos (muito grandes negativos), ela deve retornar um valor muito próximo de 1 (0). O código fornecido também deve funcionar com vetores (i.e., o código está vetorizado). Em particular, se um *array* de valores for passado para essa função, a função sigmoide é aplicada a cada componente do array.

3.2.2 Função de custo e gradiente

Agora, você deve analisar a implementação fornecida para a função de custo para a regressão logística. Essa função retorna o valor da função de custo e o gradiente. Esse código está em um arquivo denominado `custo_reglog.py`. Repare que o gradiente é um vetor com o mesmo número de elementos que θ.

Uma vez você que tenha analisado essa função, realize uma chamada usando

o valor inicial de θ, conforme a Listagem 2. Você deve confirmar que o valor produzido é aproximadamente 0,693.

Listing 2: Teste da função `custo_reglog`.

```
# Normalização das características
examData_norm, labels_norm, mean_examData, std_examData, mean_labels, std_labels =
    ↪ normalizar_caracteristica(examData, labels)

# Insere coluna de 1's no conjunto de dados examData
examData_norm = np.c_[np.ones((examData.shape[0],1)), examData_norm]

# Inicia o vetor de parâmetros
theta = np.array([0, 0, 0],ndmin=2)

# Computa o valor da função de custo J
J = funcaoCustoRegressaoLogistica(theta, examData_norm.T, labels_norm)

print(J)
```

3.2.3 Aprendizado dos parâmetros

Na regressão logística, o objetivo é minimizar $J(\theta)$ com relação ao vetor de parâmetros θ. Sendo assim, nessa parte você deve utilizar a implementação do gradiente descendente fornecida no arquivo `gd_reglog.py` para encontrar o vetor θ que minimiza a função de custo. Repare que essa implementação do gradiente descendente utiliza a função `custo_reglog`. Veja a Listagem 3.

Listing 3: Teste da função `gd_reglog`.

```
import scipy.optimize as opt
result = opt.fmin_tnc(func=custo_reglog, x0=theta, fprime=gd_reglog, args=(X, y))
custo_reglog(result[0], X, y)
```

3.2.4 Avaliação do modelo

Após o aprendizado dos parâmetros, você pode usar o modelo correspondente para predizer se um candidato qualquer será aprovado. Para um candidato com notas 45 e 85 na primeira e segunda avaliações, respectivamente, você deve esperar que ele seja aprovado com probabilidade aproximada de 0,80.

Outro modo de avaliar a qualidade dos parâmetros é verificar o quão bem o modelo aprendido prediz os pontos de dados do conjunto de treinamento. Nessa parte, você deve implementar uma função denominada `predizer` no arquivo `predizer_aprovacao.py`. Essa função deve produzir os valores 0 ou 1, dados um exemplo do conjunto de treinamento o vetor de parâmetros θ. Após implementar essa função, utilize a função `acuracia` (também definida no arquivo `predizer_aprovacao.py`) para produzir a porcentagem de acertos do seu classificador sobre o conjunto de treinamento. (Mais adiante no curso, veremos que avaliar a qualidade de um classificador sobre o conjunto de dados de treinamento não é uma boa idea. Mas, por agora, é o que você deve fazer.)

4 Regressão Logística com Regularização

Nesta parte do trabalho, você implementará a regressão logística regularizada para prever se os microchips de uma usina de fabricação passam na garantia de qualidade (QA). Durante realização da atividade de QA, cada microchip passa por vários testes para garantir se está funcionando corretamente. Suponha que você seja o gerente de produto da fábrica e que você tem o resultados de teste para alguns microchips em dois testes diferentes. A partir desses dois testes, você gostaria de determinar se os microchips deveriam ser aceitos ou rejeitados. Para ajudá-lo a tomar a decisão, você tem um conjunto de dados de resultados de testes anteriores sobre microchips, a partir do qual você pode construir um modelo de regressão logística.

O arquivo `ex2data2.txt` contém os dados a serem usados nessa parte do trabalho. A primeira coluna corresponde aos resultados do primeiro teste, enquanto que a segunda coluna corresponde aos resultados do segundo teste. A terceira coluna contém os valores da classe: $y = 0$ significa rejeitado no teste, e $y = 1$ significa aceito no teste.

4.1 Visualização dos Dados

Para a maioria dos conjuntos de dados do mundo real, não é possível criar um gráfico para visualizar seus pontos. Mas, para o conjunto de dados fornecido, isso é possível. Implemente um script em Python que produza um *gráfico de dispersão* (*scatter plot*) dos dados fornecidos. Após finalizado, seu script deve produzir um resultado similar ao apresentado na Figura 4.

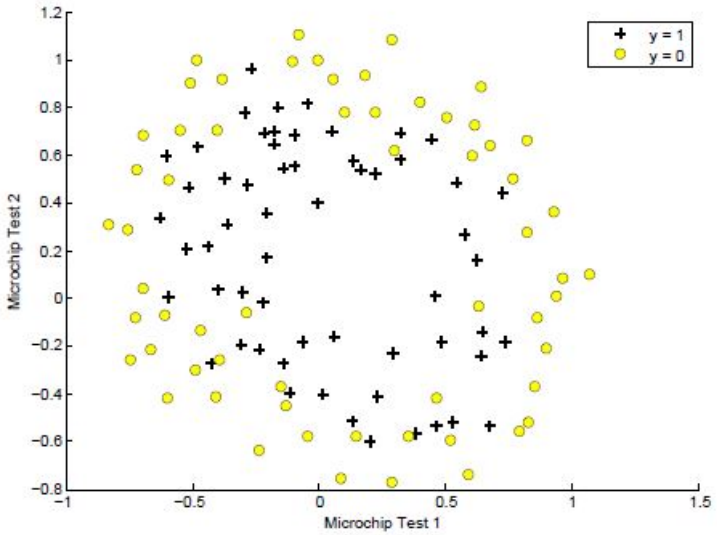

Figura 4: Pontos de dados do conjunto `ex2data2.txt`.

4.2 Mapeamento de características (*feature mapping*)

Uma maneira de tornar os dados mais apropriados para a classificação é criar mais características a partir das já existentes. Para isso, você deve criar uma função `mapFeature`. Essa função deve ser implementada em um arquivo de nome `mapFeature.py`, que irá mapear as características para todos os termos polinomiais de x_1 e x_2, até a sexta potência.

$$\text{mapFeature}(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_1^3 \\ \vdots \\ x_1 x_2^5 \\ x_2^6 \end{bmatrix}$$

Como resultado desse mapeamento, cada exemplo de treinamento correspondente a um vetor de duas características (os escores em dois testes de QA) será transformado em um vetor de 28 dimensões. Um classificador que usa regressão logística treinado nesse vetor de características de maior dimensão terá uma *fronteira de decisão* mais complexa e parecerá não linear quando desenhado em um gráfico bidimensional.

Embora o mapeamento de características nos permita construir um classificador mais expressivo, também é mais suscetível a sobreajuste (*overfitting*). Nas próximas partes, você implementará a regressão logística regularizada sobre os dados fornecidos e também verá como a regularização pode ajudar a combater o problema do sobreajuste.

4.3 Função de custo e gradiente

Nessa parte, você deve implementar o código para calcular a função de custo e o gradiente para a regressão logística regularizada. Crie um arquivo de nome `costFunctionReg.py` que contém uma função de nome `costFunctionReg.py` e que computa o custo e o gradiente. Lembre-se de que a função de custo regularizada na regressão logística é dada por:

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

O gradiente da função de custo é um vetor no qual o j-ésimo elemento é definido conforme a seguir:

$$\frac{\partial}{\partial\theta_j}J(\theta) = \begin{cases} \sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}, & \text{para } j = 0 \\ \sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m}\theta_j, & \text{para } j \in \{1, 2, \ldots, n\} \end{cases}$$

Depois de concluir a implementação da função `costFunctionReg`, você deve testar a corretude dela usando o valor inicial de θ (inicializado todo com zeros). Você deve ver que o custo para $\lambda = 1$ é de cerca de 0,693. Realize também testes com valores $\lambda = 0$ e $\lambda = 100$.

Por fim, usando a função `costFunctionReg`, você agora deve computar os valores ótimos para θ.

4.4 Esboço da fronteira de decisão

Nessa parte, você deve esboçar (plotar) a fronteira de decisão que foi aprendida para separar os exemplos positivos dos negativos. Crie uma arquivo de nome `plotDecisionBoundary.py`, para criar esse gráfico que traça o limite da decisão não-linear. Seu gráfico deve ser semelhante ao apresentado na Figura 5.

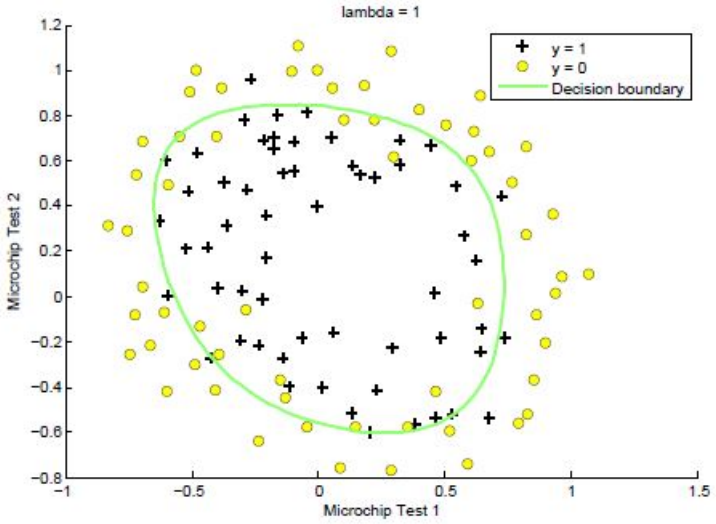

Figura 5: Esboço da fronteira de decisão ($\lambda = 1$).

5 Regressão Linear com Regularização

Na primeira metade desta parte, você implementará a regressão linear com regularização para prever a quantidade de água fluindo de uma barragem usando a mudança do nível da água em um reservatório. Na próxima metade, você realizará diagnósticos dos algoritmos de aprendizado por meio de depuração e irá examinar os efeitos de viés (*bias*) e davariância (*variance*).

Você irá usar o dataset fornecido no arquivo `ex5data1.mat`[2]. Nesse arquivo, há registros históricos na mudança no nível da água, x, e da quantidade de água que sai da barragem, y. Este conjunto de dados é dividido em três partes:

- Um conjunto de treinamento que seu modelo aprenderá em: X, y

[2]Arquivos com a extensão mat são normalmente criados no Octave ou no Matlab. Para carregar esse arquivo no Python, você pode usar o procedimento descrito em `http://www.blogforbrains.com/blog/2014/9/6/loading-matlab-mat-data-in-python`

- Um conjunto de validação cruzada para determinar o parâmetro de regularização: X_{val}, y_{val}
- Um conjunto de testes para avaliar o desempenho. Estes são exemplos que seu modelo não irá usar durante o treino: X_{test}, y_{test}

Os nomes das variáveis contidas no arquivo `ex5data1.mat` são os seguintes: `X`, `Xtest`, `Xval`, `y`, `ytest`, `yval`. Você irá precisar desses nomes para carregar os dados do arquivo para usar em seus scripts em Python.

5.1 Visualização dos Dados

Você deve começar por produzir uma visualização do conjunto de dados de treinamento. O gráfico que você deve produzir deve ser similar ao apresentado na Figura 6.

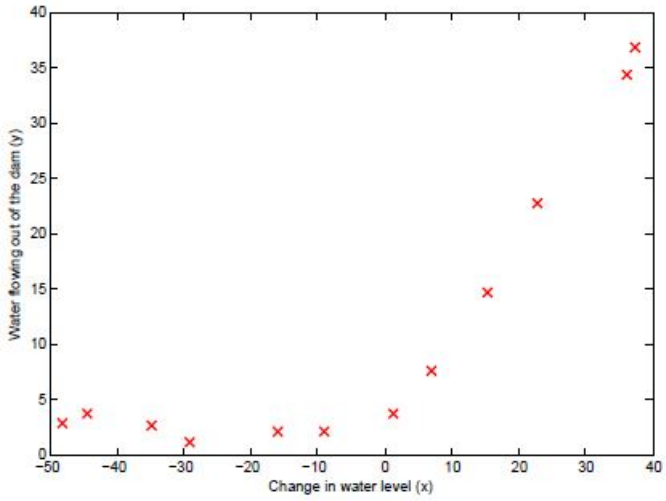

Figura 6: Pontos de dados do conjunto de treinamento.

5.2 Função de custo da regressão linear regularizada

Lembre-se de que a regressão linear regularizada tem a seguinte função de custo:

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

Lembre-se de que λ é um hyperparâmetro que controla o grau de regularização (e assim, ajuda a prevenir o excesso de sobreajuste). O termo de regularização impõe uma penalidade sobre o custo total J. Conforme as magnitudes dos parâmetros θ_j do modelo aumentam, a penalização aumenta também. Note que você não deve regularizar o termo θ_0.

Sua tarefa é escrever uma função para calcular a função de custo da regressão linear regularizada. Você deve implementar esse código em um arquivo de nome

`linearRegCostFunction.py`. Se possível, tente vetorizar seu código e evitar o uso de *loops*. Quando você tiver completado a implementação, verifique a corretude da sua função de custo usando `theta` inicializado com $(1, 1)$. Você deve esperar ver uma saída de 303,993.

5.3 Gradiente na regressão linear regularizada

A derivada parcial do gradiente da função de custo da regressão linear regularizada é um vetor no qual o j-ésimo elemento é definido conforme a seguir:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \begin{cases} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}, & \text{para } j = 0 \\ \left(\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j, & \text{para } j \in \{1, 2, \ldots, n\} \end{cases}$$

No arquivo `linearRegCostFunction.py`, adicione código para calcular o gradiente, Quando você tiver completado essa implementação, teste a corretude usando `theta` inicializado em $(1, 1)$. Você deve esperar ver um gradiente de $(-15.30, 598.250)$.

5.4 Ajustando os parâmetros da regressão linear

Nesta parte, use a função `linearRegCostFunction` para computar os valores ótimos para θ, mas sem usar regularização, i.e., defina $\lambda = 0$. Após isso, construa um gráfico para visualizar o modelo construído. Seu gráfico deve ser similar ao apresentado na Figura 7.

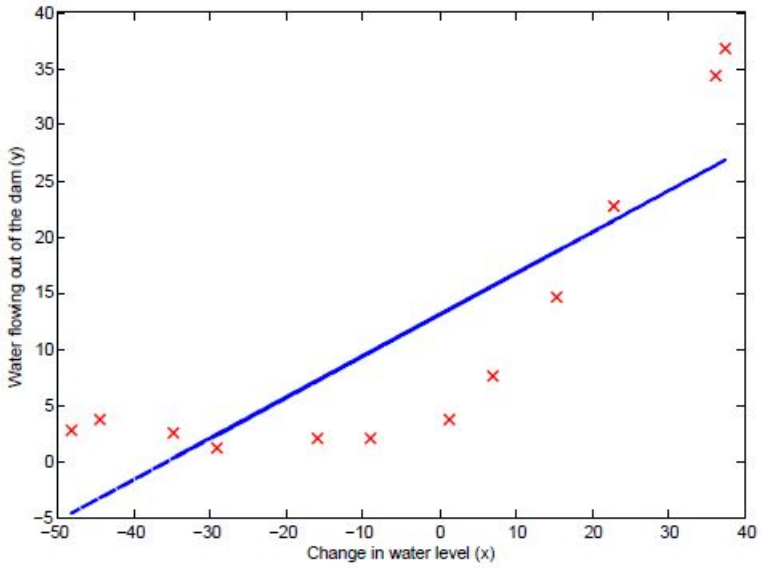

Figura 7: Ajuste linear.

6 Viés-Variância

Um conceito importante no Aprendizado de Máquina é o relacionamento entre o viés (bias) e a variância (variance) de um modelo de aprendizado. Modelos com viés elevado não são suficientemente complexos para os dados e tendem a sofrer de *subajuste* (*underfitting*), enquanto que modelos com alta variância tendem a sofrer de *sobreajuste* (*overfitting*).

Nesta parte do trabalho, você irá produzir gráficos dos erros de treinamento e teste na forma de curvas de aprendizado para diagnosticar problemas de viés-variância.

6.1 Curvas de Aprendizado

Agora você implementará código para gerar as curvas de aprendizado que serão úteis na depuração de algoritmos de aprendizagem. Lembre-se de que uma curva de aprendizagem traça erros de treinamento e de validação cruzada como funções do tamanho do conjunto de treinamento. Crie um arquivo de nome `learningCurve.py` que deve conter uma função (também chamada `learningCurve`) que retorna um vetor de erros para o conjunto de treinamento e conjunto de validação cruzada.

Para traçar a curva de aprendizado, precisamos de um conjunto de treinamento e validação cruzada erro para diferentes tamanhos de conjuntos de treinamento. Para obter diferentes tamanhos de conjuntos de treinamento, você deve usar subconjuntos diferentes do conjunto de treinamento original X. Especificamente, para um tamanho de conjunto de treinamento de i, você deve usar os primeiros exemplos de i (ou seja, $X(1:i,:)$ e $y(1:i)$).

Para cada tamanho de conjunto de treinamento, você encontrar os parâmetros θ. Note que o `lambda` deve ser passado como um parâmetro para a função `learningCurve`. Depois de aprender os parâmetros θ, você deve calcular o erro nos conjuntos de treinamento e de validação. Lembre-se de que o erro de treinamento para um conjunto de dados é definido como:

$$J_{train}(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2\right]$$

Em particular, note que o erro de treinamento não inclui o termo de regularização. Uma maneira de calcular o erro de treinamento é usar a sua função de custo já implementada e definir $\lambda = 0$ apenas para calcular o erro de treinamento e o erro de validação cruzada. Quando você estiver computando o erro no conjunto de treinamento, certifique-se de computá-lo no subconjunto de treinamento (ou seja, $X(1:n,:)$ e $y(1:n)$), em vez de usar todo o conjunto de treinamento). No entanto, para o erro de validação cruzada, você deve calculá-lo usando todo o conjunto de validação cruzada. Você deve armazenar os erros calculados em dois vetores.

Quando você estiver terminado o que foi descrito acima, imprima as curvas de aprendizado e produza um gráfico similar ao apresentado na Figura 8.

Na curva que você irá produzir, você poderá observar que os erros de treinamento e de validação cruzada são ambos altos quando o número de exemplos

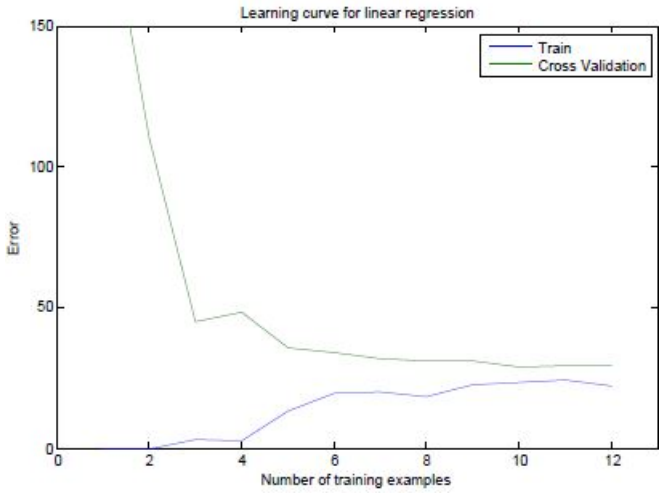

Figura 8: Curva de aprendizado para a regressão linear.

de treinamento é aumentado. Isso reflete o **viés alto** do modelo (o modelo de regressão linear é muito simples e não consegue se ajustear bem ao conjunto de dados). Na próxima seção, você irá implementar regressão polinomial para ajustar um modelo melhor a este conjunto de dados.

7 Regressão Polinomial

O problema com nosso modelo linear é que ele é muito simples para os dados e resultou em *subajuste* (*viés alto*). Nesta parte, você irá resolver esse problema adicionando mais características. Para usar a regressão polinomial, defina uma hipótese da seguinte forma:

$$h_\theta(x) = \theta_0 + \theta_1 \times \text{ waterLevel} + \theta_2 \times \text{ waterLevel}^2 + \ldots + \theta_p \times \text{ waterLevel}^p$$

Observe que, ao definir $x_1 = (WaterLevel)$, $x_2 = (WaterLevel)^2$, ..., $x_p = (WaterLevel)^p$, obtemos um modelo de regressão linear onde os características são as diferentes pontências do valor original (WaterLevel).

Agora, você irá adiciona mais características usando as pontências da característica existente x no conjunto de dados. Sua tarefa nesta parte é implementar código em um arquivo de nome `poly_features.py`. Nesse arquivo, crie um função de mesmo nome que mapeie o conjunto de treinamento original X de tamanho $m \times 1$ em suas potências mais altas. Especificamente, quando um conjunto de treinamento X de tamanho $m \times 1$ for passado para essa função, ela deve retornar uma matriz $m \times p$ de nome `X_poli`, onde a coluna 1 contém os valores originais de x, a coluna 2 contém os valores de x^2, a coluna 3 contém os valores de x^3, e assim por diante. Note que você não tem que considerar a potência de expoente zero nessa função.

Após implementar o descrito acima, você terá uma função que mapeia características para uma maior dimensão.

7.1 Regressão Polinomial - aprendizado

Depois de ter completado `poly_features.py`, você deve treinar um modelo de regressão polinomial usando sua função de custo da regressão linear.

Tenha em mente que, apesar de termos termos polinomiais no vector de características, ainda estamos resolvendo um problema de otimização de regressão linear. Os termos polinomiais simplesmente se transformaram em características que podemos usar para aplicar regressão linear. Estamos usando a mesma função de custo e gradiente que você implementou para a parte anterior deste trabalho.

Para esta parte do trabalho, você usará um polinômio de grau 8. Se executarmos o treinamento diretamente sobre os dados projetados, não iremos obter um bom resultado, porque as características não irão estar na mesma escala (por exemplo, um exemplo com $x = 40$ agora terá uma característica $x_8 = 40^8 = 6{,}5 \times 10^12$). Portanto, você vai precisar aplicar a normalização de características. Portanto, antes de aprender os parâmetros θ para a regressão polinomial, você deve normalizar as características do conjunto de treinamento, e armazenar os parâmetros μ e σ.

Depois de aprender os parâmetros θ, você deve ver gerar dois gráficos (que devem ser similares aos das Figuras 9 e 10) gerados com a regressão polinomial com $\lambda = 0$.

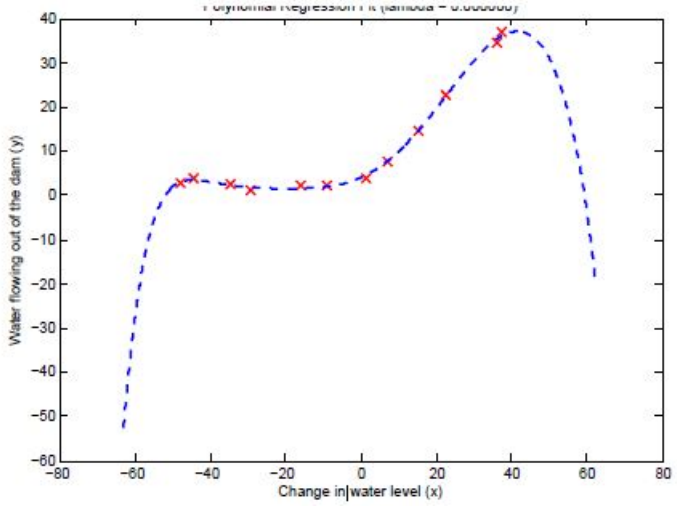

Figura 9: Ajuste polinomial, $\lambda = 0$.

Da Figura 9, você deve perceber que o polinômio pode se ajustar aos pontos de dados muito bem - assim, obtendo um baixo erro de treinamento. No entanto, o polinômio é muito complexo e até mesmo despenca nos extremos. Isso é um indicador de que o modelo de regressão polinomial está se ajustando demasiadamente aos dados de treinamento e que não irá generalizar bem.

Para entender melhor os problemas com o modelo não regularizado ($\lambda = 0$), você pode ver que a curva de aprendizado (Figura 10) apresenta erro de treinamento baixo, mas erro de validação alto. Há uma lacuna entre os erros de treinamento e validação cruzada, indicando um problema de variância alta.

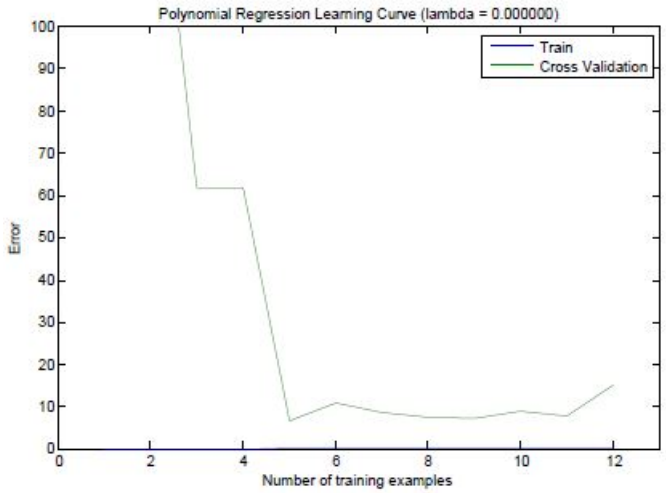

Figura 10: Curva de aprendizado do ajuste polinomial, $\lambda = 0$.

8 O que deve ser entregue

Você deve preparar um único relatório para a apresentar sua análise e conclusões sobre as diversas partes desse trabalho. O formato desse relatório deve ser em PDF. Alternativamente à entrega do relatório em PDF, você pode entregar um notebook Jupyter[3].

Independente de escolher entregar um relatório em PDF ou na forma de um notebook Jupyer, entregue também todos os arquivos em Python que você criou para cada parte deste trabalho. Todos os arquivos em Python devem estar na mesma pasta.

Crie um arquivo compactado que contém o relatório (ou notebook Jupyter) e os arquivos (*scripts*) em Python. Esse arquivo compactado deve se chamar `SEU_NOME_COMPLETO_T1.zip`. Esse arquivo compactado deve ser entregue pelo Moodle, até a data acordada.

9 Créditos

Esse trabalho é uma tradução/adaptação de partes dos *programming assignments* do curso *Machine Learning*[4] encontrado no Coursera. O material original é de autoria do prof. Andrew Ng.

[3] `http://jupyter.org/`

[4] `https://www.coursera.org/learn/machine-learning`