CEFET/RJ
Programa de Pós-graduação
em Ciência da Computação
Aprendizado de Máquina - Trabalho 02

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

Outubro/2017

Conteúdo

1 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 a QA, cada microchip passa por vários testes para garantir se está funcionando corretamente. Suponha que você seja o gerente de produto da fábrica e 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)

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, 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 1.

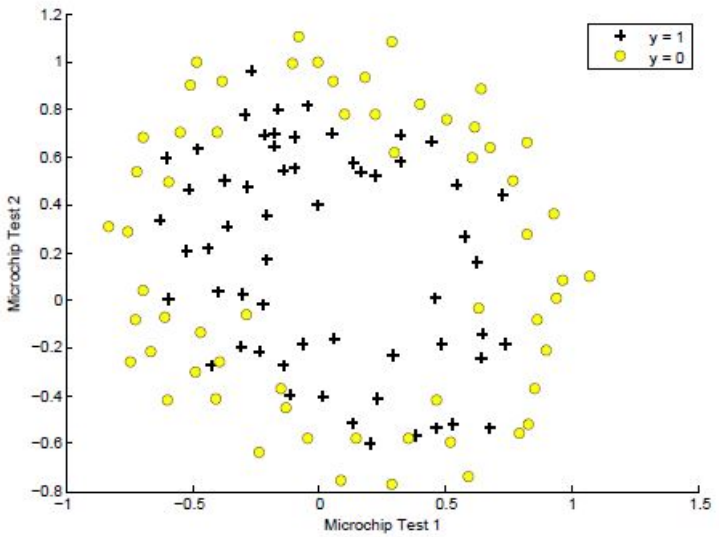

Figura 1: Pontos de dados do conjunto `ex2data2.txt`.

1.2 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`. essas 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, nosso vetor de dois características (os escores em dois testes de QA) será transformado em um vetor de 28 dimensões. Uma logística 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 do exercício, você implementará a regressão logística regularizada sobre os dados fornecidos e também verá como a regularização pode ajudar a combater o problema do sobreajuste.

1.3 Função de custo e gradiente

Agora, você deverá 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:

$$\frac{\partial J(\theta)}{\partial \theta_0} = \sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)},\ \text{ para } j = 0$$

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

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 é de cerca de 0,693.

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

1.4 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 2.

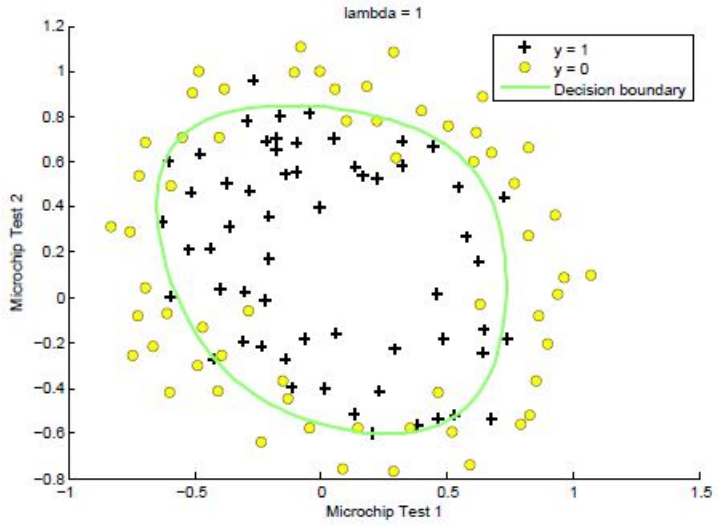

Figura 2: Esboço da fronteira de decisão ($\lambda = 1$).

2 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`[1]. Nesse arquivo,

[1] 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`

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
- 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.

2.1 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 3.

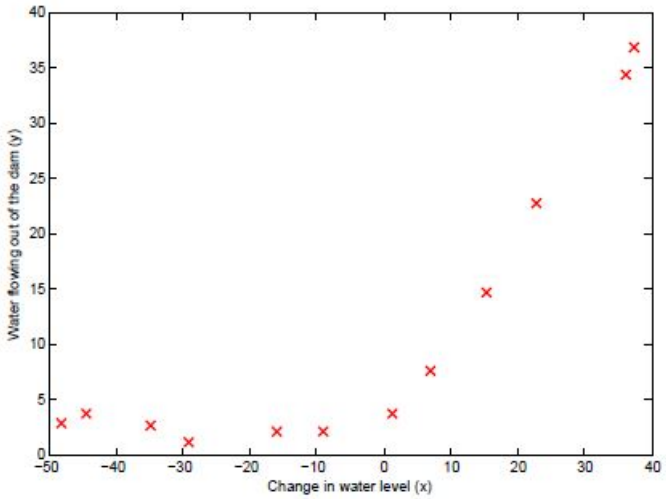

Figura 3: Pontos de dados do conjunto de treinamento.

2.2 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 do modelo θ_j 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.

2.3 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:

$$\frac{\partial J(\theta)}{\partial \theta_0} = \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}, \text{ para } j = 0$$

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

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

2.4 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 visualiar o modelo construído. Seu gráfico deve ser similar ao apresentado na Figura 4.

3 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.

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.

3.1 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

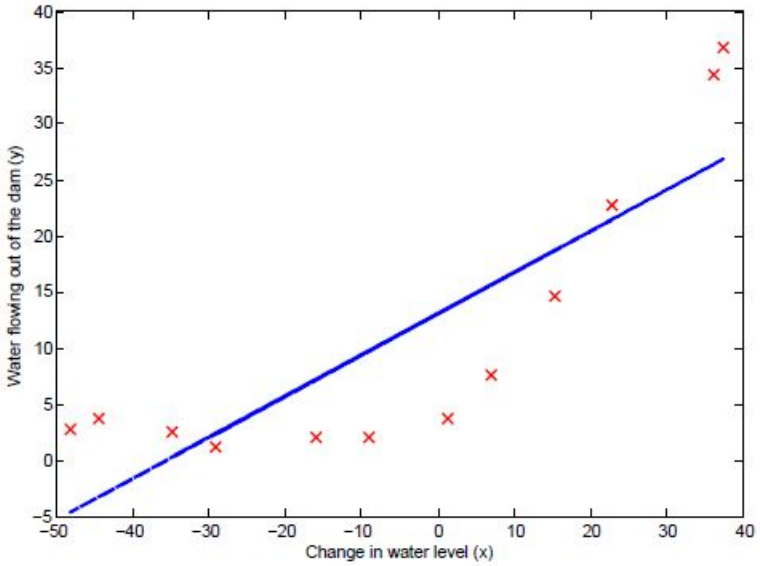

Figura 4: Ajuste linear.

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 5.

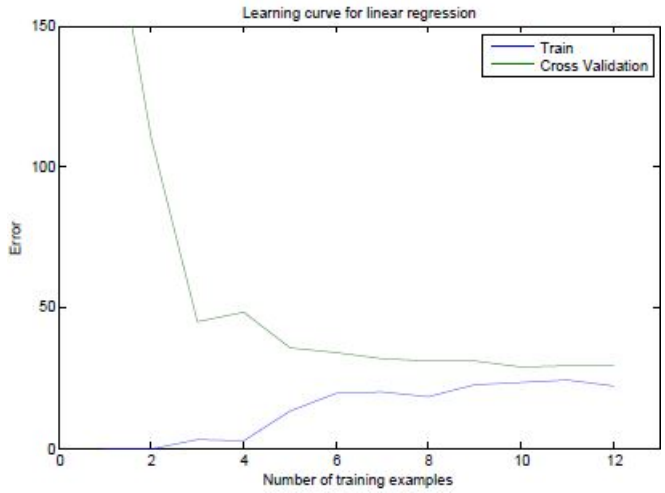

Figura 5: Curva de aprendizado para a regressão linear.

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 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.

4 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 `polyFeatures.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.

5 Regressão Polinomial - aprendizado

Depois de ter completado `polyFeatures.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 6 e 7) gerados com a regressão polinomial com $\lambda = 0$.

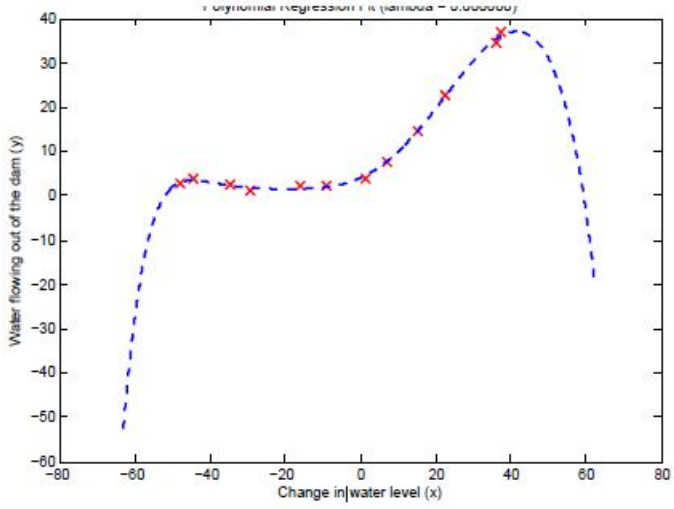

Figura 6: Ajuste polinomial, $\lambda = 0$.

Da Figura 6, você deve perceber que o polinômio pode se ajustar aos pontos de dados muito bem - assim, obtendo um baixo erro de treinamento. No

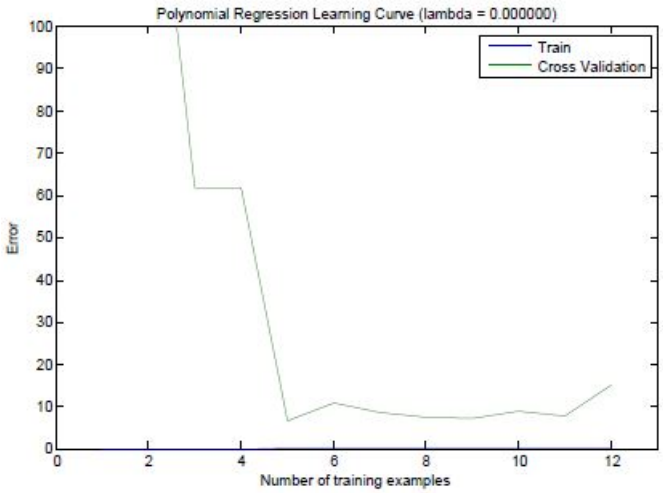

Figura 7: Curva de aprendizado do ajuste polinomial, $\lambda = 0$.

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 7) 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.

6 Tarefas adicionais (OPCIONAIS)

- Ajuste do parâmetro de regularização. Teste o aprendizado para diferentes valores de lambda, $1 \leq \lambda \leq 100$. Para $\lambda = 1$, você deve ver o polinômio segue a tendência de dados bem (Figura 6) e uma curva de aprendizado (Figura 7) mostrando que tanto o erro de validação quanto o erro de treinamento convergem para um valor relativamente baixo. Isso mostra que o modelo de regressão polinomial regularizado com $\lambda = 1$ não sofre dos problema de viés alto ou de variância alta. De fato, esse modelo alcança um bom compromisso entre viés e variância. Para $\lambda = 100$, você deve ver um polinômio (Figura 8) que não segue os dados bem. Neste caso, há muita regularização, e o modelo é incapaz de se ajustar aos dados de treinamento.

- Defina o valor de λ usando o conjunto de validação. Implemente um método automatizado para selecionar o parâmetro λ. Concretamente, use um conjunto de validação cruzada para avaliar quão bom é cada valor de λ. Tente valores de λ no intervalo $\{0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10\}$. Ao realizar essa tarefa, você deve encontrar um valor adequado de λ en torno de 3 (veja a Figura 8).

- Depois de selecionar o melhor valor de λ usando o conjunto de validação,

você pode então avaliar o modelo no conjunto de teste para estimar o quão bem ele será executado em dados reais não vistos.

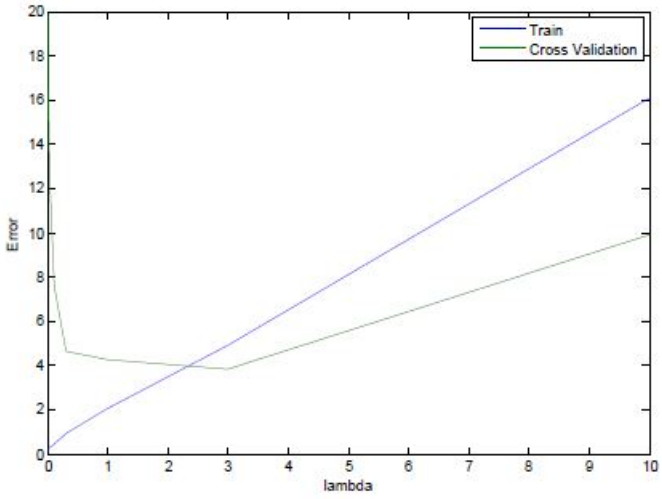

Figura 8: Curva de aprendizado do ajuste polinomial, $\lambda = 0$.

7 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[2].

Independente de escolher entregar um relatório em PDF ou na forma de um notebook Jupyter, entregue também todos os arquivos em Python que você criou para cada parte deste trabalho. Todos os arquivos em Python deve estar em uma única 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.

8 Créditos

Esse trabalho é uma tradução/adaptação dos *programming assignment* do curso *Machine Learning*[3] encontrado no Coursera. O material original é de autoria do prof. Andrew Ng.

[2] `http://jupyter.org/`
[3] `https://www.coursera.org/learn/machine-learning`