Gradiente Descendente Estocástico – MAC5832

Pedro Henrique Rocha Bruel
phrb@ime.usp.br
nUSP: 7143336

DCC - IME
Universidade de São Paulo
São Paulo, 16 de Junho de 2016

1 Introdução

Este documento descreve as implementações realizadas para o Trabalho 2 da disciplina *MAC5832*, apresenta análises de desempenho desses algoritmos segundo seus parâmetros, e compara o desempenho dessas implementações com algoritmos do pacote `Scikit Learn`.

Foram implementados um algoritmo para Gradiente Descendente Estocástico, dois modelos de regressão e um método de regularização, totalizando quatro combinações de modelos: Regressão Linear, Regressão Linear com Regularização L_2, Regressão Logística e Regressão Logística com Regularização L_2.

1.1 Estrutura de Diretórios

Os diretórios deste trabalho estão organizados da seguinte forma:

`./doc` Contém os arquivos necessários para gerar este documento.

`./data` Contém os conjuntos de dados disponibilizados para treinamento e testes.

`./src` Contém o código implementado para o *Perceptron* e *Gradient Descent.*

`./src/plot` Contém o código para gerar as figuras com as visualizações, e também as figuras.

2 Gradiente Estocástico Descendente

O método do *Gradiente Estocástico Descendente* é uma técnica iterativa para minimização de funções duplamente diferenciáveis. No nosso caso, o método é aplicado para minimizar o erro de classificação $E_{in}(\mathbf{w}, \mathbf{X}, \mathbf{y})$, onde $\mathbf{w}$ é um vetor de pesos, $\mathbf{X}$ é uma matriz de exemplos, ou amostras, e $\mathbf{y}$ é um vetor de classificações, onde cada classificação $\mathbf{y}(i)$ corresponde a um exemplo $\mathbf{X}(i)$.

Para encontrar o vetor de pesos $\mathbf{w}^*$ que minimiza $E_{in}(\cdot)$, cada iteração do método do Gradiente Estocástico Descendente calcula o *gradiente* $\nabla E_{in}(\cdot)$ e atualiza o vetor atual de pesos.

A técnica do Gradiente Estocástico Descendente pode ser compreendida intuitivamente como a "exploração" de um terreno com diferentes inclinações e profundidades, utilizando uma bola sujeita à força da gravidade. A bola tenderá a encontrar o ponto do terreno com menor profundidade, mas o local inicial terá bastante impacto no local final. Formas de atenuar esse problema incluem utilizar várias bolas ao mesmo tempo, ou bolas com "coeficientes de atrito" diferentes.

2.1 Algoritmo

A implementação da técnica do Gradiente Estocástico Descendente deste trabalho foi feita na linguagem `Python 3.5.1` e está descrita em pseudocódigo no Algoritmo 1. O resto dessa Seção descreve a implementação em detalhes.

2.1.1 Parâmetros

A implementação foi feita de forma *modular*, permitindo que funções para o cálculo do gradiente e da taxa de aprendizado fossem passadas como parâmetro. Esta Seção descreve os parâmetros para o algoritmo implementado.

Taxa de Aprendizado inicial (α'): Em todos os experimentos deste trabalho utilizei a seguinte função para o cálculo da taxa de aprendizado $\alpha(i)$, onde i é o número da iteração atual do algoritmo, e α' é a taxa de aprendizado inicial:

$$\alpha(i) = \frac{\alpha'}{1 + log_2 i}$$

Seguindo a metáfora para a compreensão intuitiva da técnica, a ideia é modificar o "coeficiente de atrito", isto é, a taxa de aprendizado, para que em iterações inicias sejam permitidas variações maiores no vetor de pesos, e posteriomente na execução do algoritmo, apenas "saltos" menores.

Função do Gradiente ($\nabla E_{in}(\cdot)$): A expressão para o cálculo do gradiente será diferente, de acordo com o modelo de regressão que utilizarmos em conjunto com a técnica de Gradiente Estocástico Descendente. As expressões para cálculo do gradiente dos quatros modelos de regressão implementados neste trabalho estão descritas na Seção 3.

A implementação de todas as funções de gradiente recebe como entrada um vetor de pesos $\mathbf{w}$, e um subconjunto dos exemplos em $\mathbf{X}$, junto com suas classificações em $\mathbf{y}$ correspondentes. A saída dessas funções é o gradiente calculado de acordo com o modelo de regressão correspondente.

Número de Iterações (I): É o número de iterações realizadas antes de se devolver o vetor final de pesos $\mathbf{w}(I)$.

Vetor Inicial de Pesos ($\mathbf{w}(0)$): Deve ter o mesmo número de elementos em cada exemplo de $\mathbf{X}$. É inicializado com zeros.

Matriz de Exemplos ($\mathbf{X}$): Contém os exemplos para teste em suas linhas.

Vetor de Classificações ($\mathbf{y}$): A posição $\mathbf{y}(i)$ contém a classificação do exemplo $\mathbf{X}(i)$.

Tamanho do *Batch* (σ): Determina o tamanho do subconjunto de exemplos que será utilizado em cada iteração. O tamanho do conjunto de dados não permitiu que os gradientes fossem calculados usando todo o conjunto de uma só vez. Para contornar isso utilizei um parâmetro que determina a *porcentagem* do conjunto que será utilizada nas iterações.

A cada iteração são calculados os conjuntos $\mathbf{X}[\sigma]$ e $\mathbf{y}[\sigma]$, que contém m elementos escolhidos *ao acaso*, onde $m = |X| \cdot \sigma$. Dessa forma é possível usar todo o conjunto de dados sem pagar o custo dos produtos escalares de matrizes enormes.

2.1.2 Saída

Vetor Final de Pesos ($\mathbf{w}(I)$): Contém um peso para cada característica dos exemplos de teste em $\mathbf{X}$. É usado posteriormente para fazer as predições de classificação.

Algorithm 1 Gradiente Estocástico Descendente
com Taxa de Aprendizado Variante

Input:

$\nabla E_{in}(\cdot)$ ▷ Função para cálculo do Gradiente
I ▷ Número de Iterações
σ ▷ Tamanho do *batch*
α' ▷ Taxa de Aprendizado inicial
$\mathbf{w}(0)$ ▷ Vetor inicial de pesos
$\mathbf{X}$ ▷ Matriz de exemplos
$\mathbf{y}$ ▷ Vetor de classificações

Output:

$\mathbf{w}(I)$ ▷ Vetor final de pesos

```
for i = 1, 2, ..., I do
    Calcule g_i = ∇E_in(w(i), X[σ], y[σ])
    Calcule α(i) = α' / (1 + log_2 i)
    Calcule w(i) = w(i − 1) − α g_i
end for
return w(I)
```

for $i = 1, 2, \ldots, I$ **do**
  Calcule $\mathbf{g}_i = \nabla E_{in}(\mathbf{w}(i),\ \mathbf{X}[\sigma],\ \mathbf{y}[\sigma])$
  Calcule $\alpha(i) = \frac{\alpha'}{1+log_2 i}$
  Calcule $\mathbf{w}(i) = \mathbf{w}(i-1) - \alpha\mathbf{g}_i$
end for
return $\mathbf{w}(I)$

---

3 Modelos de Regressão

Esta Seção discute os modelos de regressão e suas implementações, feitas em `Python 3.5.1` e usando o módulo `numpy`.

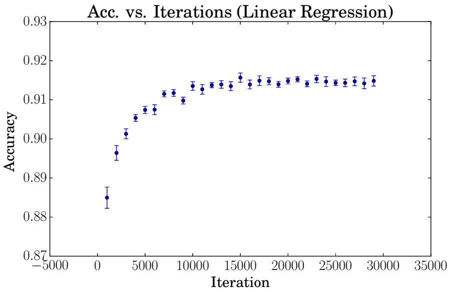
3.1 Regressão Linear

3.2 Regressão Linear com Regularização L_2

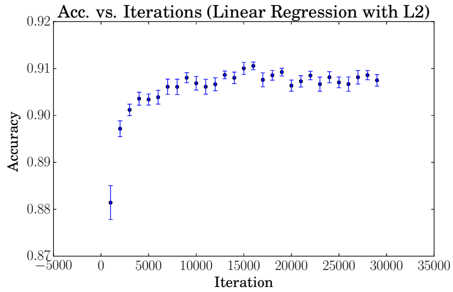
3.3 Regressão Logística

3.4 Regressão Logística com Regularização L_2

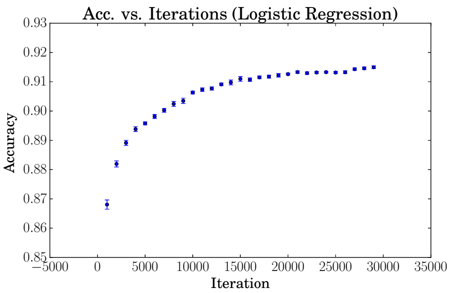
4 Resultados



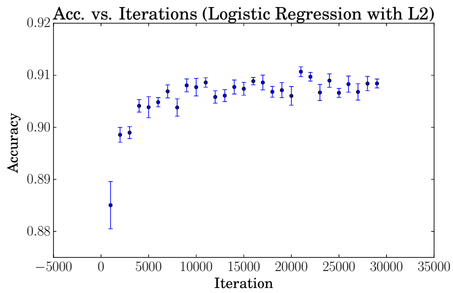
(a)



(b)



(c)



(d)

Figura 1: Acurácia vs. Número de Iterações (I) do Gradiente Estocástico Descendente, com parâmetros fixos: $\lambda = 0.0051$, $\alpha = 0.5$, $\sigma = 10^{-3}$

5 Conclusão

A minha implementação do *Perceptron* teve um desempenho melhor do que a implementação do *Gradient Descent* mas, usando metade dos exemplos

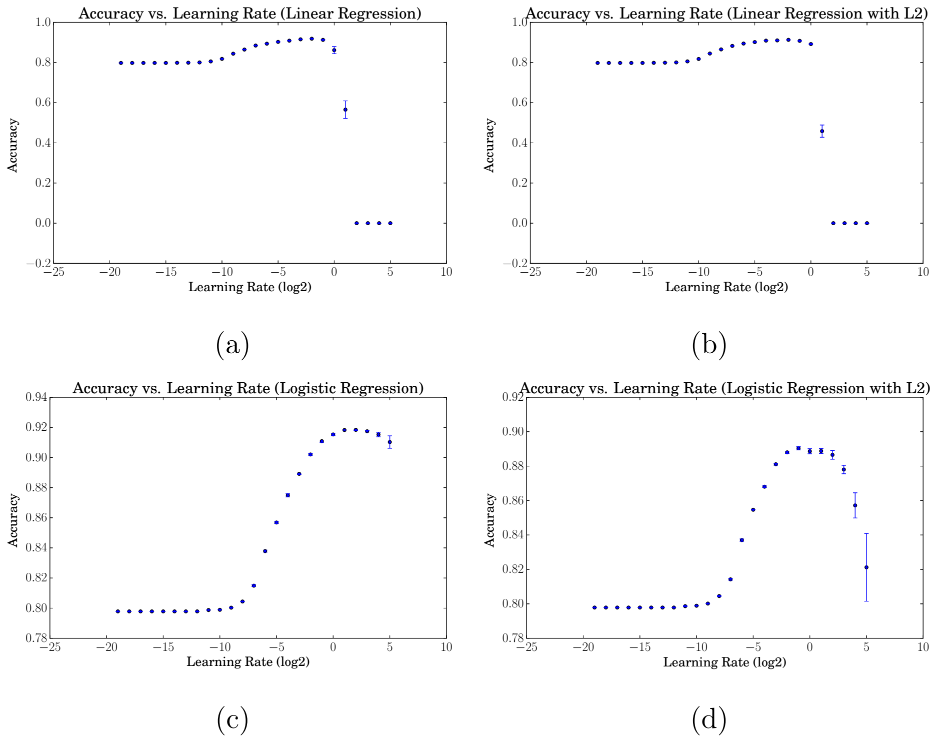
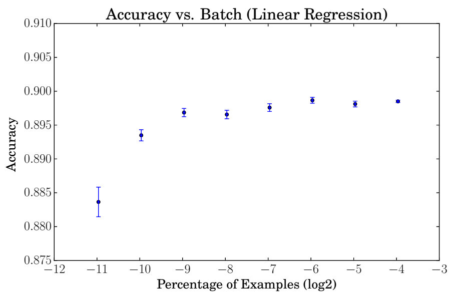
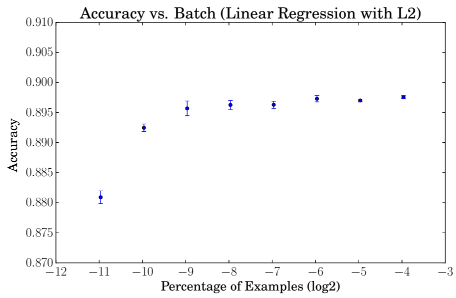


Figura 2: Acurácia vs. Taxa de Aprendizado (α), com parâmetros fixos: $\lambda = 0.0051$, $I = 15000$, $\sigma = 10^{-3}$

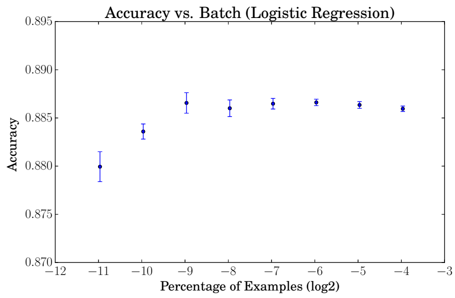
Modelos	**Parâmetros**	
	λ	α
Regr. Lin.	-	0.25
Regr. Lin. (L_2)	2^{-4}	0.25
Regr. Log.	-	2
Regr. Log. (L_2)	2^{-10}	0.5

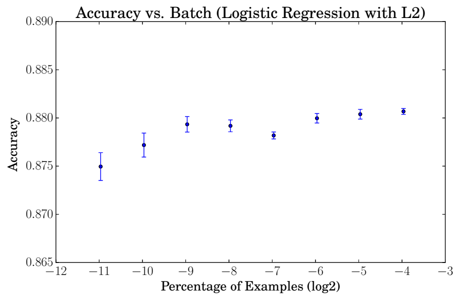
Tabela 1: Escolha de parâmetros para as implementações, após os experimentos. Todos os modelos usaram o mesmo número de iterações, $I = 10^4$, e tamanho de *batch*, $\sigma = 2^{-6}$ (aprox. 1.5% do conjunto de dados por iteração)





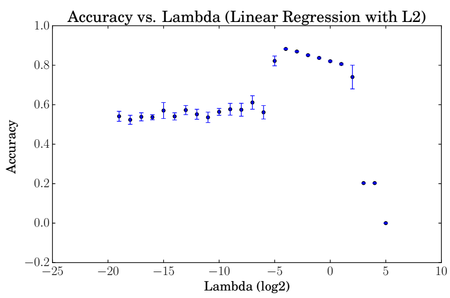
(a) (b)

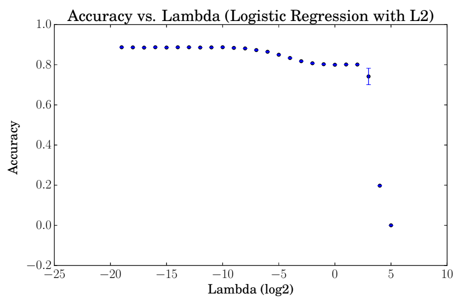




(c) (d)

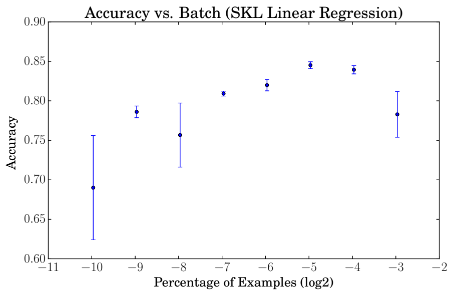
Figura 3: Acurácia vs. Tamanho do *Batch* (σ), com parâmetros fixos: $\lambda = 0.0051$, $I = 500$, $\alpha = 0.5$



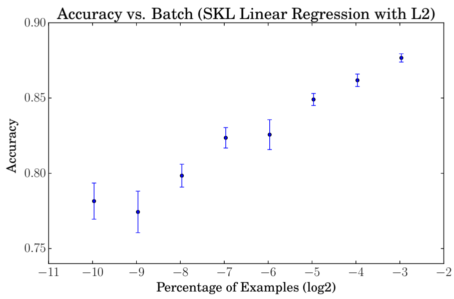


(a) (b)

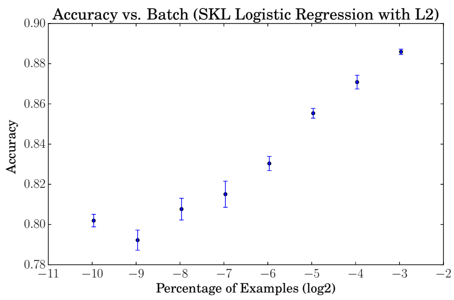
Figura 4: Acurácia vs. Parâmetro λ, com parâmetros fixos: $\sigma = 0.003$, $\alpha = 2.0$, $I = 500$



(a)



(b)



(c)

Figura 5: Acurácia vs. Tamanho do *Batch* (σ) para algoritmos do `Scikit Learn`

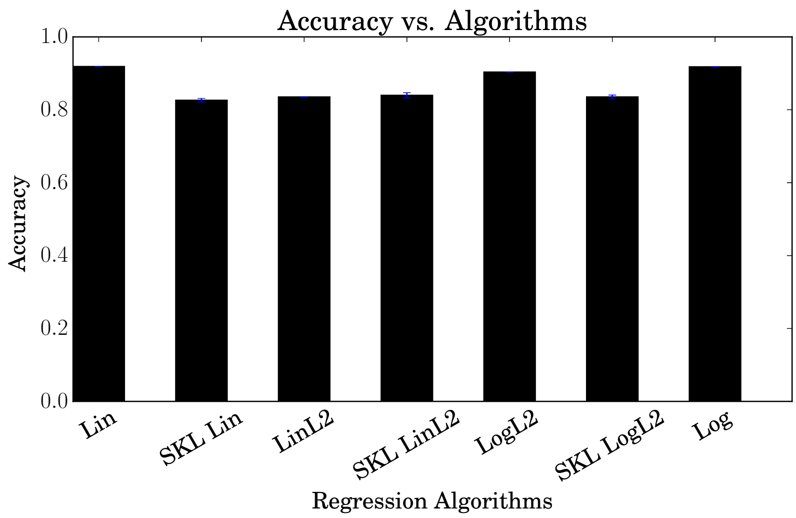


Figura 6: Acurácia dos algoritmos implementados e do `Scikit Learn`. Cada algoritmo foi executado com diferentes parâmetros otimizados

como treinamento e outra metade como teste, ambos os algoritmos atingiram resultados de por volta de 10% de exemplos classificados erroneamente.