Trabalho II de MAC5832:

Gradiente Estocástico Descendente, Modelos de Regressão e Regularização L_2

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 relatório 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.

`./src` Contém o código implementado para os modelos de regressão e experimentos.

`./img` Contém as figuras apresentadas neste relatório.

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 (X): Contém os exemplos para teste em suas linhas.

Vetor de Classificações (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 $m = |\mathbf{X}| \cdot \sigma$, e $|\mathbf{X}|$ é o número total de exemplos. 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

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

3 Modelos de Regressão

Esta Seção apresenta as expressões para os modelos de regressão implementados em `Python 3.5.1`, usando o módulo `numpy`.

3.1 Regressão Linear

A ideia deste modelo é separar, se possível, o conjunto de dados com um plano de dimensão $|\mathbf{w}|$, isto é, de dimensão igual ao número de características dos exemplos. Além disso, a soma dos quadrados das distâncias entre os exemplos e o plano deve ser minimizada. A hipótese no caso da Regressão Linear é dada por:

$$h(\mathbf{X}(i), \mathbf{w}) = \mathbf{w}^T \mathbf{X}(i)$$

Expressão para $E_{in}(\cdot)$

$$E_{in}(\mathbf{w}, \mathbf{X}, \mathbf{y}) = \frac{1}{|\mathbf{X}|} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

Onde $|\mathbf{X}|$ é o número total de exemplos.

Expressão para $\nabla E_{in}(\cdot)$

$$\nabla E_{in}(\mathbf{w}, \mathbf{X}, \mathbf{y}) = \frac{2}{|\mathbf{X}|} (\mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{X}^T \mathbf{y})$$

3.2 Regressão Logística

A ideia deste algoritmo é considerar a *probabilidade* de classificação dos exemplos, e não apenas um valor binário. A hipótese no caso da Regressão Logística é dada por:

$$h(\mathbf{X}(i), \mathbf{w}) = \theta(\mathbf{w}^T \mathbf{X}(i))$$

Onde a função $\theta(\cdot)$ produz valores *entre* 0 e 1, e é chamada de *função logística*:

$$\theta(s) = \frac{e^s}{1+e^s}$$

Expressão para $E_{in}(\cdot)$

$$E_{in}(\mathbf{w}, \mathbf{X}, \mathbf{y}) = \frac{1}{|\mathbf{X}|} \sum_{i=1}^{|\mathbf{X}|} ln(1 + e^{-\mathbf{y}(i)\mathbf{w}^T\mathbf{X}(i)})$$

Expressão para $\nabla E_{in}(\cdot)$

$$\nabla E_{in}(\mathbf{w}, \mathbf{X}, \mathbf{y}) = -\frac{1}{|\mathbf{X}|} \sum_{i=1}^{|\mathbf{X}|} \frac{\mathbf{y}(i)\mathbf{X}(i)}{1 + e^{\mathbf{y}(i)\mathbf{w}^T\mathbf{X}(i)}}$$

3.3 Regressão Linear com Regularização L_2

A *Regularização* é uma estratégia para "combater" a tendência que os modelos de regularização apresentam de ajustar-se demais ao conjunto de dados de aprendizado, num processo chamado de *overfitting*.

Utilizamos neste trabalho o método para regularização introduzido por *Andrey Tikhonov*, conhecido como *Ridge Regression*. Na discussão do livro-texto, esse método de regressão é apresentado como a adição de uma limitação, ou *constraint*, à otimização do E_{in} de um dado modelo de regressão.

No caso da *Regularização* L_2 essa limitação é representada pelo termo:

$$E_{in}(\mathbf{w}) = -\lambda \|\mathbf{w}\|^2$$

Onde $\lambda > 0$ é um parâmetro a ser escolhido. O gradiente do termo de regularização é dado por:

$$\nabla E_{in}(\mathbf{w}) = -2\lambda\mathbf{w}$$

Adicionando o termo de regularização ao modelo de Regressão Linear, temos:

Expressão para $E_{in}(\cdot)$

$$E_{in}(\mathbf{w}, \mathbf{X}, \mathbf{y}) = \frac{1}{|\mathbf{X}|}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 - \lambda\|\mathbf{w}\|^2$$

Expressão para $\nabla E_{in}(\cdot)$ Considerando a propriedade de ∇:

$$\nabla(f + g) = \nabla(f) + \nabla(g)$$

E usando a expressão para $\nabla E_{in}(\cdot)$ da Regressão Linear, temos:

$$\nabla E_{in}(\mathbf{w}, \mathbf{X}, \mathbf{y}) = \frac{2}{|\mathbf{X}|}(\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y}) - 2\lambda\mathbf{w}$$

3.4 Regressão Logística com Regularização L_2

Adicionando o termo de regularização ao modelo de Regressão Logística, temos:

Expressão para $E_{in}(\cdot)$

$$E_{in}(\mathbf{w}, \mathbf{X}, \mathbf{y}) = \frac{1}{|\mathbf{X}|}\sum_{i=1}^{|\mathbf{X}|} ln(1 + e^{-\mathbf{y}(i)\mathbf{w}^T\mathbf{X}(i)}) - \lambda\|\mathbf{w}\|^2$$

Expressão para $\nabla E_{in}(\cdot)$

$$\nabla E_{in}(\mathbf{w}, \mathbf{X}, \mathbf{y}) = -\frac{1}{|\mathbf{X}|}\sum_{i=1}^{|\mathbf{X}|} \frac{\mathbf{y}(i)\mathbf{X}(i)}{1 + e^{\mathbf{y}(i)\mathbf{w}^T\mathbf{X}(i)}} - 2\lambda\mathbf{w}$$

3.5 Modelos do `Scikit Learn`

Utilizei as implementações para Regressão Linear, Regressão Linear com Regularização L_2 e Regressão Logística com Regularização L_2 disponíveis no módulo `sklearn.linear_model` do pacote `Scikit Learn`. Não utilizei o método de Regressão Logística sem Regularização.

4 Resultados

Esta Seção apresenta os resultados obtidos medindo a acurácia dos modelos de regressão implementados para o trabalho. O desempenho dos modelos foi avaliado usando o conjunto de dados disponibilizado pelo monitor da disciplina. Neste trabalho, a acurácia é a porcentagem de exemplos no conjunto de testes que foram classificados corretamente.

O objetivo dessas avaliações de desempenho é estudar como cada modelo de regressão e regularização se comporta quanto à acurácia em relação a seus parâmetros. Busquei encontrar os valores ideais dos parâmetros I (Número de Iterações), σ (Tamanho do *Batch*), λ (Parâmetro de Regularização) e α (Taxa de Aprendizagem) para cada modelo de regressão e regularização.

Os pontos de todos os gráficos apresentados nessa Seção foram obtidos calculando a média de 10 execuções do respectivo modelo com os parâmetros em questão. Cada ponto também apresenta o erro padrão calculado à partir das 10 medições.

4.1 Acurácia vs. Iterações

Como era de se esperar, a acurácia do algoritmo para o Gradiente Estocástico Descendente aumenta com o Número de Iterações. Note que o aumento da acurácia diminui depois de cerca de 15000 iterações. Também é interessante notar que o erro padrão das amostras para o modelo de Regressão Linear com Regularização L_2 é maior que o das amostras de outros modelos. Não sei explicar o porquê.

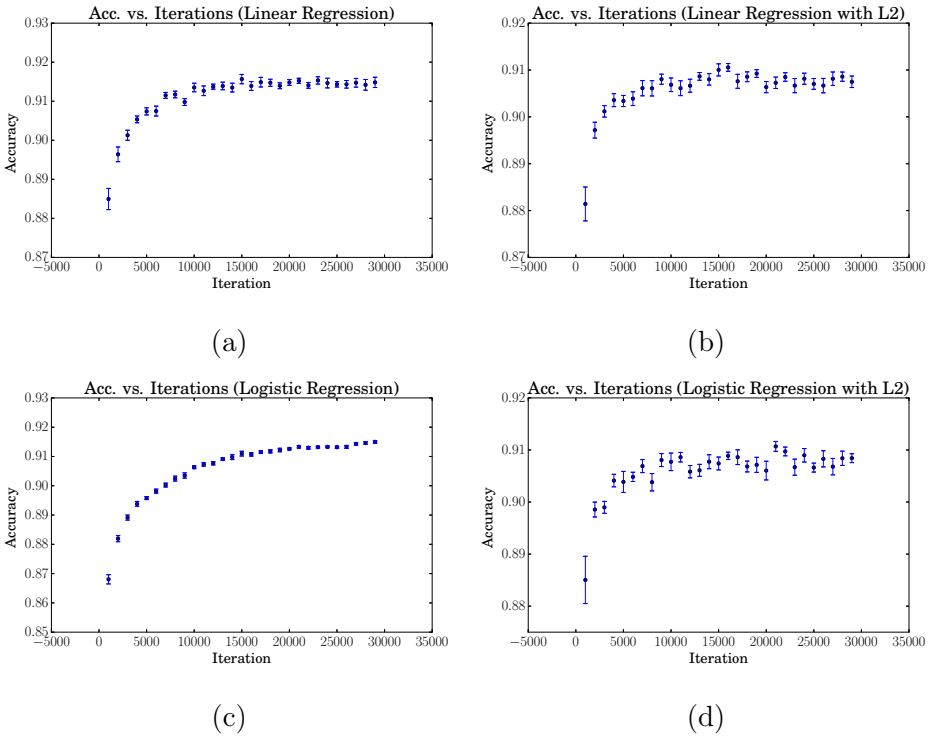


Figura 1: Acurácia vs. Número de Iterações (I) do Gradiente Estocástico Descendente, com parâmetros fixos: $\lambda = 0.0051$, $\alpha = 0.5$, $\sigma = 10^{-3}$

4.2 Acurácia vs. Taxa de Aprendizado

Os resultados para a Regressão Linear e a Regressão Linear com Regularização L_2 apresentaram acurácia de 0% para $\alpha > 2^0$ por conta de erros de *overflow*. Uma possível maneira de resolver esse problema seria normalizar o vetor de pesos. Não experimentei essa solução pois os pontos válidos para esses modelos mostram que a acurácia começa a diminuir para $\alpha > 2^{-1}$. Note que cada algoritmo tem um valor preferível de α.

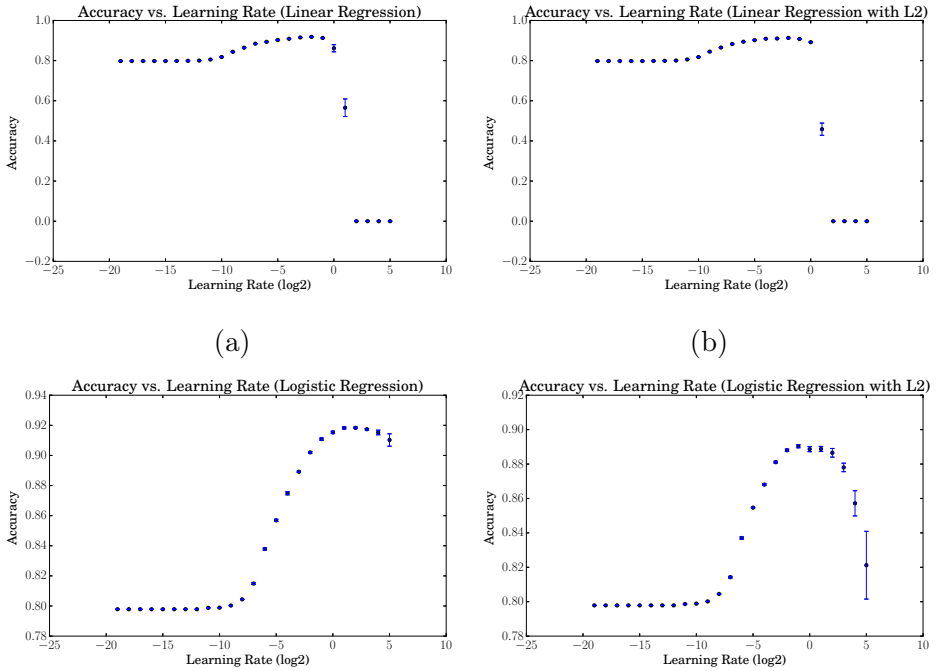


Figura 2: Acurácia vs. Taxa de Aprendizado (α), com parâmetros fixos: $\lambda = 0.0051$, $I = 15000$, $\sigma = 10^{-3}$

4.3 Acurácia vs. Tamanho do *Batch*

Novamente, quanto maior a porcentagem σ de exemplos do conjunto usados em cada iteração, maior a acurácia do modelo. É necessário balancear a acurácia com a capacidade computacional, pois a duração de uma iteração é proporcional a σ. Note que o erro padrão é inversamente proporcional a σ, pois com mais exemplos usados a variabilidade fica menor.

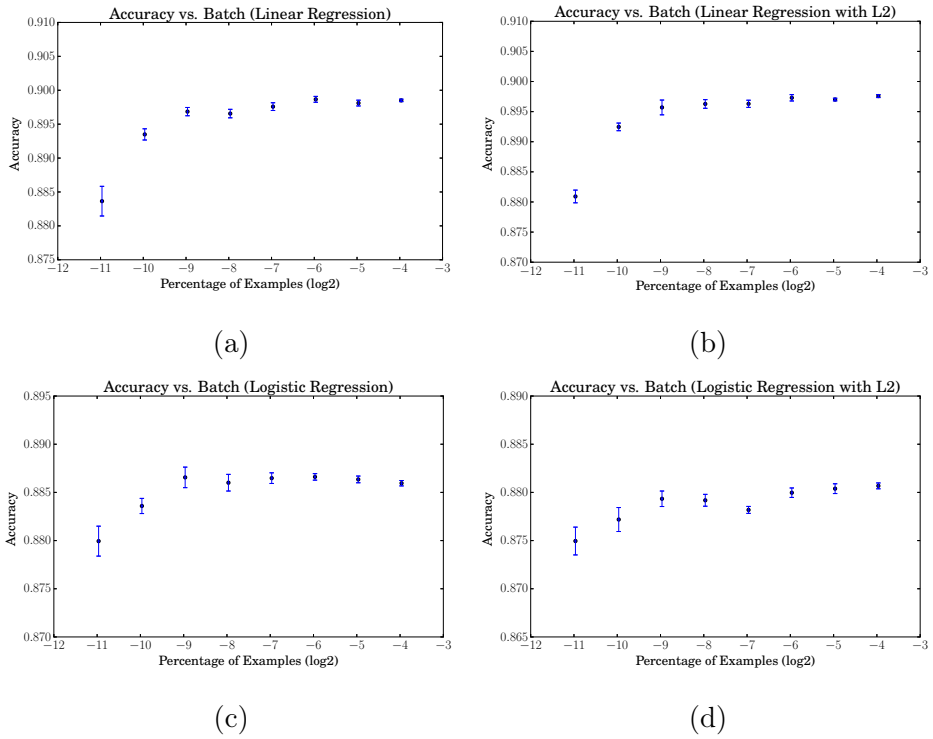
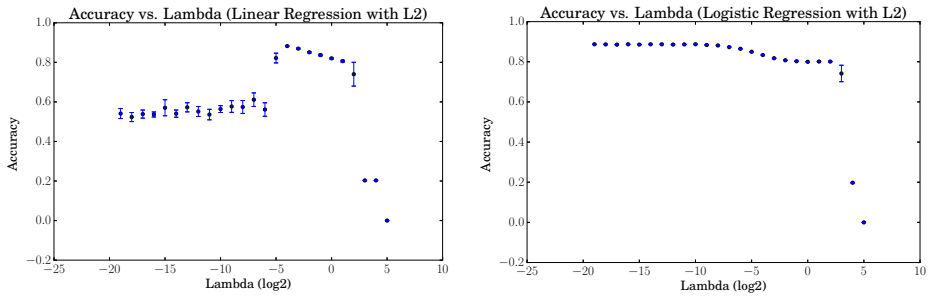


Figura 3: Acurácia vs. Tamanho do *Batch* (σ), com parâmetros fixos: $\lambda = 0.0051$, $I = 500$, $\alpha = 0.5$

4.4 Acurácia vs. λ

A acurácia dos modelos em relação a λ se comporta de forma distinta para os dois modelos com regularização implementados. O modelo de Regressão Linear com Regularização L_2 apresenta maior acurácia para $2^{-5} < \lambda < 2^0$, e o modelo de Regressão Logística com Regularização L_2 apresenta melhor acurácia para $\lambda < 2^{-5}$. Alguns valores de λ também resultaram em *overflow*, mas também decidi não implementar normalizações por que a acurácia pareceu aumentar na outra direção de variação do parâmetro.



(a) (b)

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

4.5 Acurácia vs. Tamanho do *Batch* para o `Scikit Learn`

A acurácia dos algoritmos do `Scikit Learn` se beneficiou mais do aumento de tamanho do *batch*. Acredito que isso se deva ao fato de que as minhas implementações ainda tinham acesso ao conjunto total de dados, e escolhiam ao acaso quais exemplos comporiam o *batch* a ser utilizado numa dada iteração. Assim, as implementações desse trabalho se beneficiaram do conjunto total de dados e também do benéficio de realizar operações em porções menores de exemplos. O erro padrão das medições também foi maior para esses algoritmos.

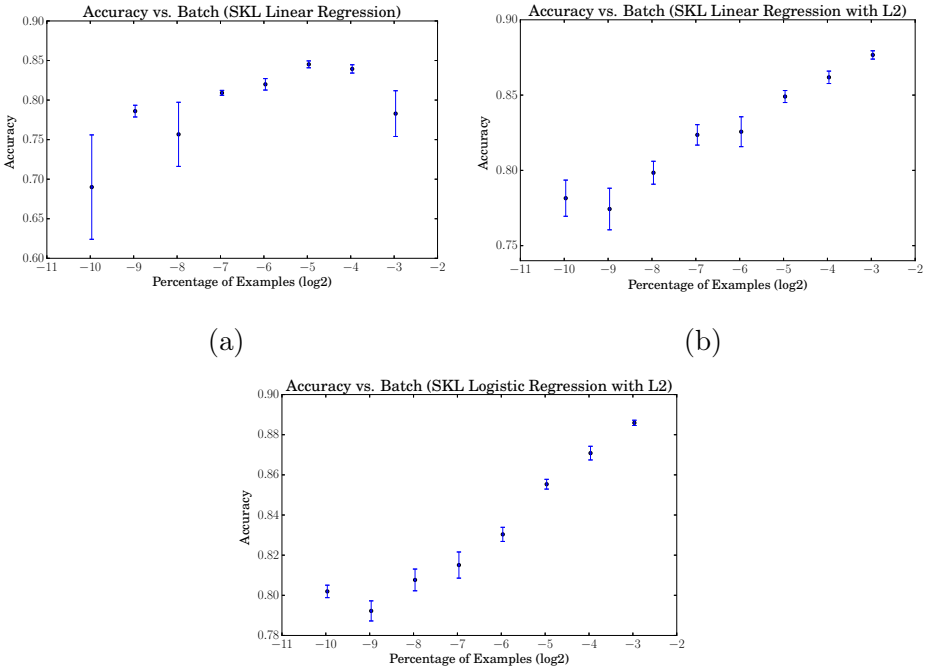


Figura 5: Acurácia vs. Tamanho do *Batch* (σ) para algoritmos do `Scikit Learn`

4.6 Otimização dos Parâmetros

A Tabela 1 apresenta a escolha de parâmetros para cada modelo feita após os experimentos.

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)

4.7 Comparação de Acurácia

A Figura 6 apresenta comparações da acurácia dos modelos implementados neste trabalho e dos modelos disponíveis no `Scikit Learn`, denotados por "*SKL*" na figura.

Acredito que as minhas implementações tiveram acurácia maior que as implementações do `Scikit Learn` por conta do motivo descrito na Seção 4.5: nas minhas implementações, o parâmetro σ determina qual a *porcentagem* do conjunto de exemplos que será utilizada em cada iteração, mas não limita *quais* exemplos podem ser selecionados. A cada iteração, um novo conjunto de exemplos é selecionado. Nas implementações do `Scikit Learn` o parâmetro σ determina a qual porcentagem do conjunto de exemplos o algoritmo terá *acesso*.

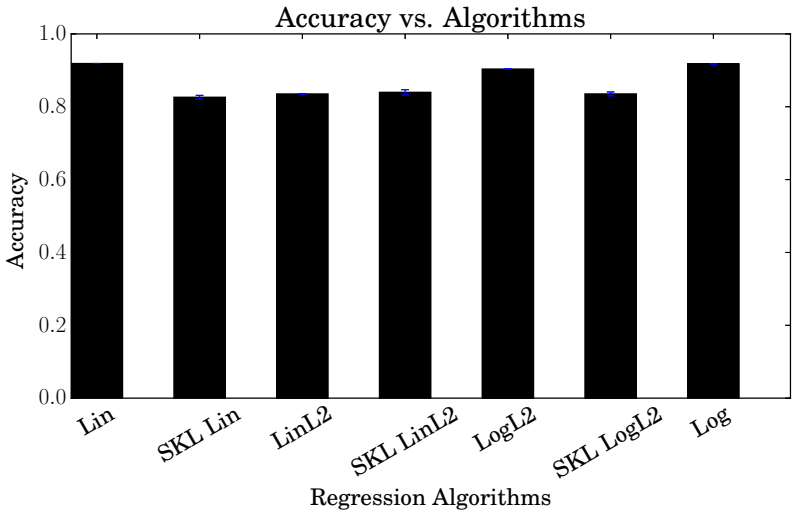


Figura 6: Acurácia dos algoritmos implementados e do `Scikit Learn`. Cada algoritmo foi executado com diferentes parâmetros otimizados

5 Conclusão

Este relatório apresentou e discutiu implementações dos modelos de Regressão Linear e Regressão Logística, usando o método de Regularização L_2. Foram feitas análises de acurácia em relação à variação dos parâmetros mais importantes desses modelos, e comparações de acurácia com implementações dos modelos no pacote `Scikit Learn`.