Classificando Tweets – MAC5832

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

DCC - IME
Universidade de São Paulo
São Paulo, 15 de Junho de 2016

1 Introdução

Este relatório apresenta visualizações e algoritmos de aprendizado de máquina utilizados no primeiro *dataset* da disciplina **MAC5832**.

O *dataset* consiste de um conjunto de *tweets* sobre companhias aéreas, associado a avaliações de *sentimento*. Os algoritmos de aprendizado de máquina implementados foram o *Perceptron* e o *Gradient Descent*. Os dois algoritmos estão descritos nas Seções **??** e **??**. Visualizações do *dataset* são apresentadas na Seção **??**. A Seção **??** discute os resultados e conclui este relatório.

1.1 Estrutura de Diretórios

Os diretórios deste trabalhos 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 Algoritmos de Aprendizado de Máquina

Esta Seção discute os algoritmos de aprendizado de máquina e suas implementações, feitas em `Python 3` e usando os módulos `nltk`, `sklearn`, `numpy`, `csv` e `re`.

Utilizei o tutorial sobre *Bag of Words*[1], disponibilizado na página da Tarefa I, para gerar um *vocabulário* com as palavras mais frequentes nos *tweets* do *dataset*.

O vocabulário foi então usado para gerar os *vetores de características* utilizados para treinar os algoritmos e realizar as predições. Os vetores de características contêm um `1` na posição correspondente a uma determinada palavra se o *tweet* contém aquela palavra, e um `0` caso contrário.

Além disso, cada vetor de características é inicializado com um `1` na primeira posição, correspondente ao *threshold* do classificador.

2.1 Perceptron

Considere um vetor $\boldsymbol{w}$ de pesos inicializado com $N + 1$ `1`s, onde cada peso $w_i, 0 \leq i \leq N$ corresponde a uma característica dos vetores de características dos *tweets* do *dataset*, e N é o número de características extraídas do conjunto.

A cada passo do algoritmo *Perceptron*, vamos calcular a classificação c_j de cada um dos M vetores $\boldsymbol{x}_j, 0 \leq j \leq M$ de características usando a seguinte equação:

$$c_j = step\left(\sum_{i=0}^{N} w_i \boldsymbol{x}_j^i\right) = step\left(\boldsymbol{w}^T \boldsymbol{x}_j\right)$$

Onde a função *step* é dada por:

$$step(y) = \begin{cases} 1 & \text{se } y \geq 0 \\ -1 & \text{caso contrário.} \end{cases}$$

Depois, para cada vetor $\boldsymbol{x}_j, 0 \leq j \leq M$ vamos atualizar o vetor $\boldsymbol{w}$ de pesos da seguinte maneira:

[1] `https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-1-for-beginners-bag-of-words`

$$\boldsymbol{w} = \boldsymbol{w} + \alpha \boldsymbol{x}_j (z_j - c_j)$$

Aqui, $\boldsymbol{x}_j$ é um vetor de características, α é a *taxa de aprendizado*, z_j é a classificação correta de $\boldsymbol{x}_j$ e c_j é a predição feita pelo algoritmo nessa iteração.

A minha implementação do *Perceptron* executa esse procedimento para todos os exemplos no conjunto de testes até que o número de exemplos classificados erroneamente seja menor ou igual a um limite μ. Nesse *dataset*, utilizei $\mu = 6$. Depois do treinamento do modelo, obtemos um vetor $\boldsymbol{w}^*$, que é usado para gerar as predições para cada *tweet* contido nos dados para teste.

2.2 Gradient Descent

Considere um vetor $\boldsymbol{w}$ de pesos inicializado com $N + 1$ `1`s, onde cada peso w_i, $0 \leq i \leq N$ corresponde a uma característica dos vetores de características dos *tweets* do *dataset*, e N é o número de características extraídas do conjunto.

A cada passo do algoritmo *Gradient Descent*, vamos calcular a classificação c_j de todos os vetores $\boldsymbol{x}_j$, $0 \leq j \leq M$, $\boldsymbol{x}_j \in \boldsymbol{X}$ de características, onde $\boldsymbol{X}$ é a matriz de exemplos. Faremos a classificação usando a seguinte equação:

$$\boldsymbol{C} = \boldsymbol{w}^T \boldsymbol{X}$$

Aqui, $\boldsymbol{C}$ é um vetor contendo um *valor real* para cada exemplo $\boldsymbol{x}_j \in \boldsymbol{X}$. Para obter uma predição da classificação de algum $\boldsymbol{x}_j$, usamos função $step(\boldsymbol{C}_j)$, onde $\boldsymbol{C}_j$ é j-ésimo valor em $\boldsymbol{C}$. A função *step* é, novamente, dada por:

$$step(y) = \begin{cases} 1 & \text{se } y \geq 0 \\ -1 & \text{caso contrário.} \end{cases}$$

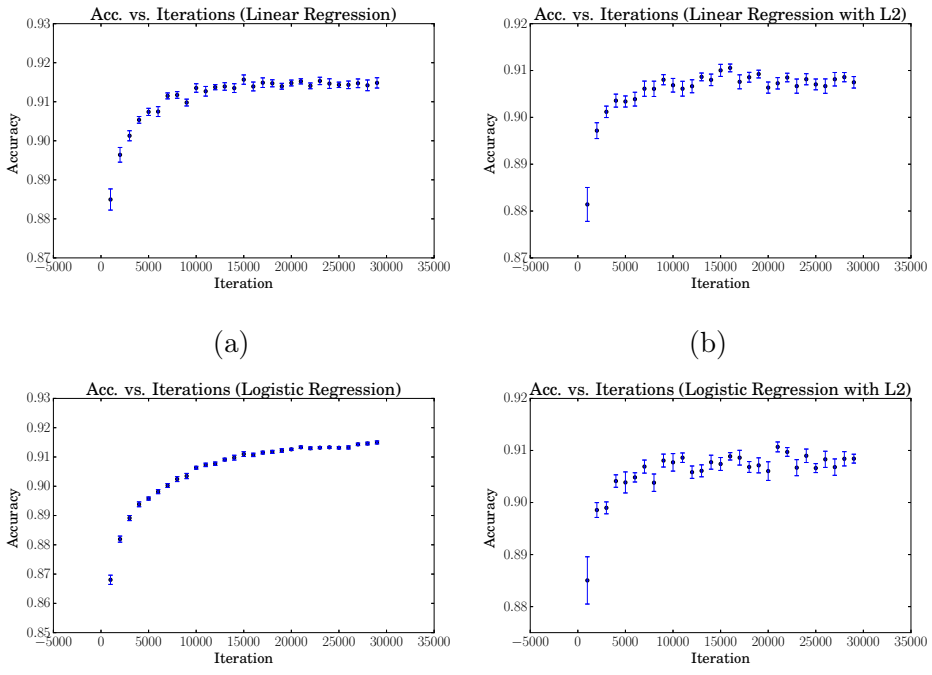
Depois de calcular $\boldsymbol{C}$, atualizamos o vetor $\boldsymbol{w}$ de pesos da seguinte maneira:

$$\boldsymbol{w} = \boldsymbol{w} - \alpha \frac{1}{M} \boldsymbol{X}(\boldsymbol{C} - \boldsymbol{Z})$$

Aqui, $\boldsymbol{X}$ é a matriz de exemplos, α é a *taxa de aprendizado*, $\boldsymbol{Z}$ é a matriz com as classificações corretas de cada exemplo em $\boldsymbol{X}$ e $\boldsymbol{C}$ é a matriz de predições feita pelo algoritmo nessa iteração.

A minha implementação do *Gradient Descent* executa esse procedimento até que o número de iterações seja maior do que um limite ρ. Nesse *dataset*, utilizei $\rho = 500$. Depois do treinamento do modelo, obtemos um vetor $\boldsymbol{w}^*$, que é usado para gerar as predições para cada *tweet* contido nos dados para teste.

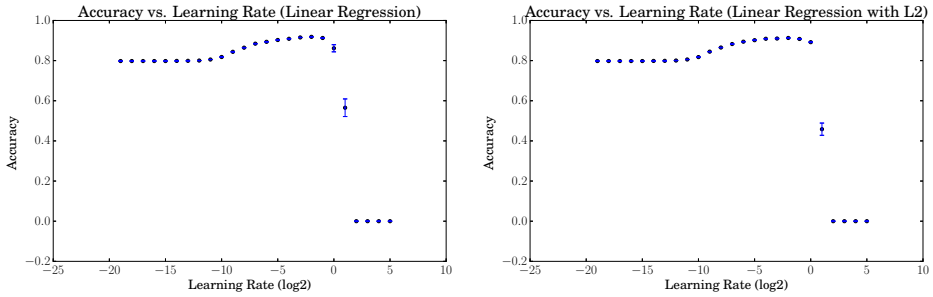
3 Resultados



(a) (b)

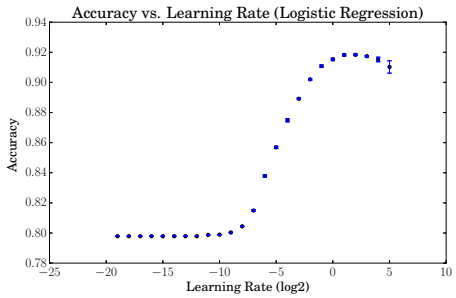
(c) (d)

Figura 1

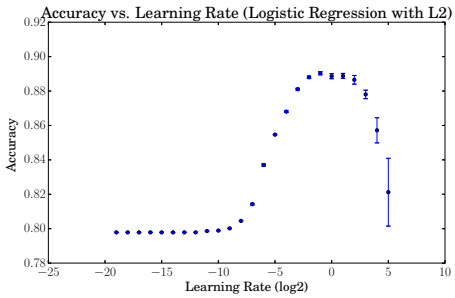


(a)

(b)

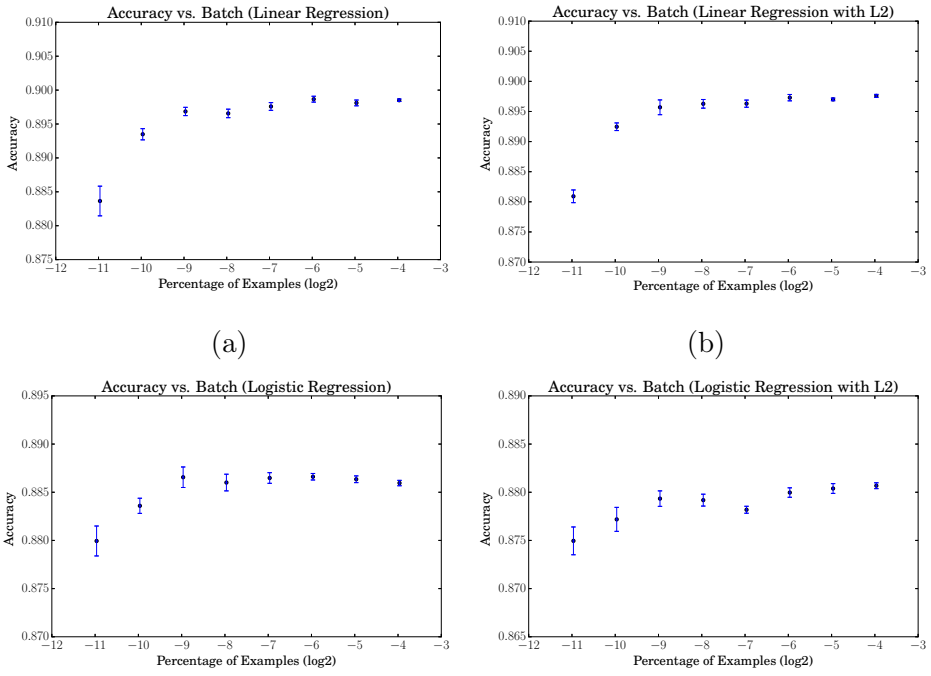


(c)



(d)

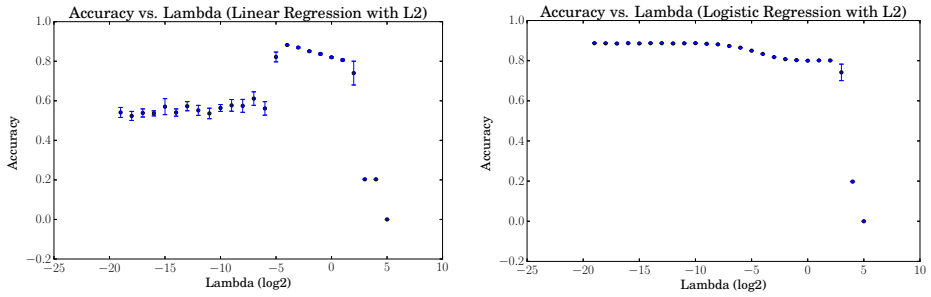
Figura 2



(a) (b)

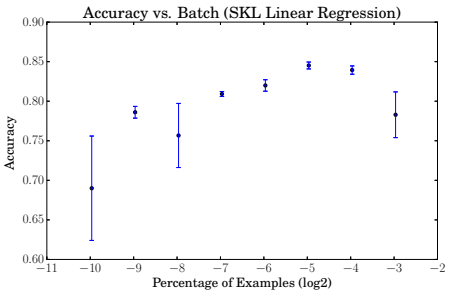
(c) (d)

Figura 3

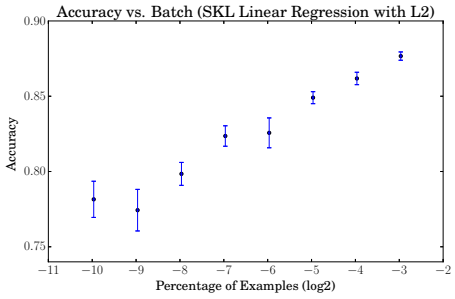


(a) (b)

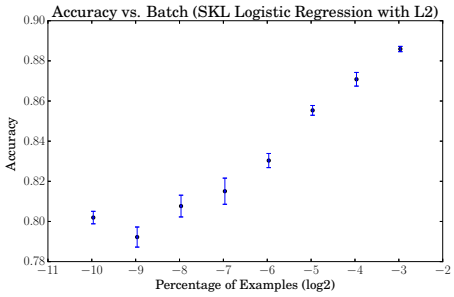
Figura 4



(a)



(b)



(c)

Figura 5

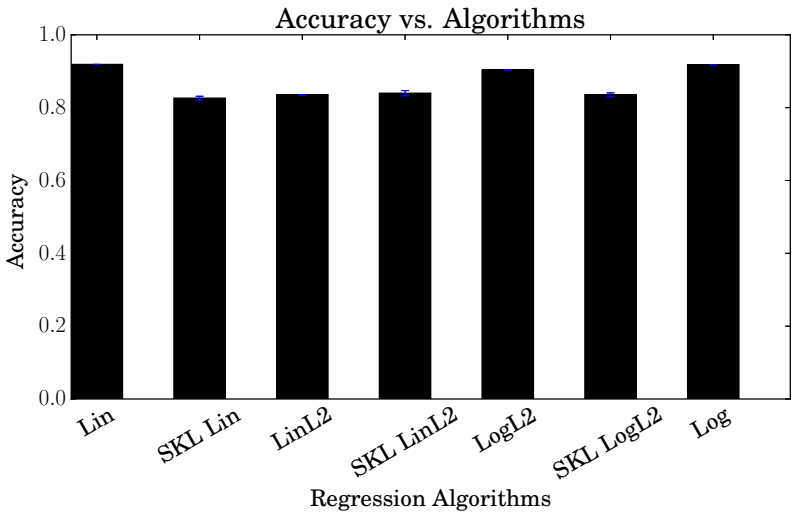


Figura 6

4 Conclusão

A minha implementação do *Perceptron* teve um desempenho melhor do que a implementação do *Gradient Descent* mas, usando metade dos exemplos como treinamento e outra metade como teste, ambos os algoritmos atingiram resultados de por volta de 10% de exemplos classificados erroneamente.