Segundo Trabalho Prático

Luis Gustavo Lorgus Decker
209819
luisgustavo.decker@gmail.com

Luiz Antonio Falaguasta Barbosa
075882
lafbarbosa@gmail.com

I. Introdução

Neste trabalho, aplicamos duas soluções de classificação de dados por aprendizado de máquina no contexto de classificação de imagens: Uma regressão logística e uma rede neural multicamadas. Aplicamos ambas no banco de imagens CIFAR-10 [2], que se constitui de 50000 imagens de treino e 10000 imagens de teste, cada uma com um rótulo associado ao objeto de interesse na imagem. Expomos a modelagem de cada uma destas técnicas, suas principais características e resultados de testes efetuados acima dos modelos propostos.

II. Soluções Propostas

Visando classificar corretamente as 10000 imagens de teste, implementamos um classificador modelado como uma regressão logística e outro como uma rede neural artificial multicamadas. Devido a alta dimensionalidade oriunda destes dados, totalizando 3072 dimensões no total, sendo 1024 de cada canal de cada imagem analisada, utilizamos a técnica de Análise de Componente Principal para reduzir as dimensões dos dados.

A. Análise de Componentes Principais

A análise de componentes principais (PCA) é uma das técnicas mais utilizadas como algorítimo de redução de dimensionalidade. Seu funcionamento se resume em encontrar o hiperplano que esteja mais próximo dos dados,e então projetar os mesmos neste hiperplano.

O primeiro passo na tarefa de redução de dimensionalidade é encontrar o hiperplano correto para projetar os dados. Este processo se resume em encontrar o hiperplano cuja variância entre os dados se preserve ao máximo, permitindo assim uma menor perda de informação em relação a outras projeções.

O PCA identifica o eixo que preservam a maior variância no conjunto de dados. Após isso, encontra outro eixo, ortogonal ao primeiro, que preserve ao máximo o restante da variância dos dados, e repete este processo até que tenha obtido mesmo número de vetores quanto dimensões houverem nos dados. Cada um destes vetores é, ordenadamente, um componente principal.

Neste trabalho, utilizamos a técnica de *Decomposição em Valores Singulares*(SVD), uma técnica que é capaz de decompor uma matriz X que representa os dados de treinamento em três matrizes, $V^T, \Sigma\, e\, U$, onde V^T contém os componentes principais.

Visando reduzir a dimensionalidade, temos que escolher um número k de componentes para gerar o hiperplano em que projetaremos os dados. Fazemos esta escolha da seguinte maneira: sendo $\alpha \in [0, 1]$ o valor de porcentagem da variância que desejamos manter, escolhemos os k primeiros componentes tal que a soma da variância associada a estes componentes represente $\alpha\%$ da variância total, ou seja, partindo de $k = 1$, aumentamos k até que

$$\frac{\sum_{i=1}^{k} \Sigma_i}{\sum_{i=1}^{m} \Sigma_i} >= \alpha$$

, sendo que Σ_i é a variância associada ao iésimo componente principal, encontrada na matriz Σ retornada pelo SVD.

Encontrado um valor de k que satisfaça a restrição de α, criamos um hiperplano em $\mathbb{R}^k$ com os k primeiros componentes principais, e projetamos os dados de treino neste hiperplano. Estes dados projetados estarão sendo utilizados como os novos dados de treino.

B. Regressão Logística

Uma regressão logística é um modelo linear de classificação. Seu principal uso é estimar a probabilidade de uma amostra pertencer a um grupo. Se a probabilidade de pertinência for maior do que 50%, o modelo prediz que a instância pertence ao grupo, caso contrário não, o que o torna um classificador binário.

De maneira semelhante a uma regressão linear, a regressão logística computa uma soma ponderada das *features* de entrada, mas ao invés de retornar o resultado diretamente como uma regressão linear, a logística retorna o resultado aplicado na função logística.

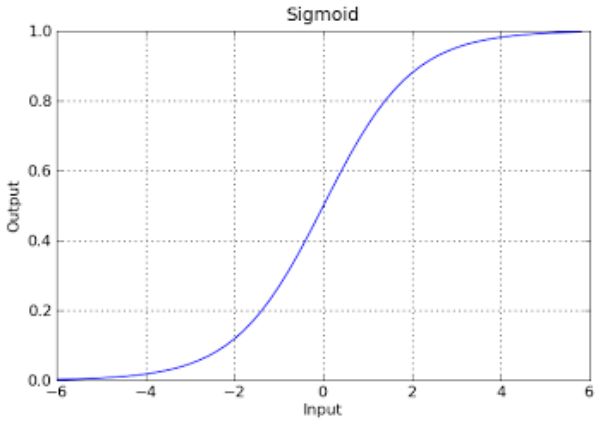
Formalmente, sendo $x = \{x_1, x_2, \ldots, x_n\}$ um vetor representando as *features* de um determinado dado x, e sendo $\theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$ um vetor de pesos associados a cada uma das *features* de x, podemos definir a probabilidade h_θ estimada da amostra representada por x pertencer ao grupo ao qual o classificador foi treinado como sendo

$$h_\theta(x) = g(\theta^T \cdot x)$$

onde $g(x)$ é a função logística, dada como

$$g(x) = \frac{1}{1 + e^{-x}}.$$

Figure 1. Plot da função sigmoide



O treinamento de uma regressão logística tem como objetivo atualizar os valores do vetor de parâmetros θ de tal maneira que maximize a probabilidade para dados pertinentes ao conjunto, ou seja, saída mais próxima a 1, e minimize para dados não pertinentes ao conjunto, ou seja, saída mais próxima de zero. Isto pode ser expresso matematicamente pela função de custo para uma entrada. Sendo p a probabilidade estimada de uma certa entrada x pertencer a classe analisada, temos que a função de custo é

$$f_{custo}(\theta) = \begin{cases} -log(p) & \text{se o elemento pertence ao grupo} \\ -log(1-p) & \text{se o elemento não pertence ao grupo} \end{cases}$$

Esta função penaliza fortemente resultados que se encontram com probabilidades opostas as esperadas.

Quando queremos encontrar uma função que representa o custo em todo o conjunto de treinamento, e não só em um item específico, podemos usar o custo médio por todo o conjunto de treinamento, $J(\theta)$:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[^{(i)}log(p^{(i)}) + (1-y^{(i)})log(1-p^{(i)})\right]$$

onde $y^{(i)}$ é a probabilidade real de uma amostra pertencer a classe e m é o número de amostras no conjunto de treinamento.

Visando minimizar o erro, ou seja, encontrar os coeficientes que maximizem as probabilidades corretas para todos os elementos do conjunto de treinamento utilizado na regressão logística, pode ser utilizada uma função de minimização, como o gradiente descendente [1], [3]. Neste trabalho utilizamos a função de gradiente médio estocástico, assim como sugerida em [4].

Assim que o processo de treinamento foi concluído, ou seja, assim que o método de minimização de custos convergir para o mínimo (esta convergência é garantida, já que a função de custo é convexa), a regressão linear está apta a retornar probabilidades de pertinência ao conjunto ao qual foi treinada.

Neste trabalho, porém, precisamos classificar imagens em 10 categorias diferentes. Já que uma regressão linear somente é capaz de retornara a probabilidade de pertinência ou não de uma amostra a uma classe somente, utilizamos o método de *Um Contra Todos* para encontrar a classe da imagem analisada. Neste método, treina-se a regressão logística com uma classe por vez, analisando a pertinência da amostra a cada uma das classes. O algoritmo classifica a amostra como pertinente a classe cuja regressão logística retornou maior probabilidade.

Outra abordagem para classificação multiclasses utilizando regressão logística é o uso de regressões logísticas multinomiais, também conhecida como regressão *softmax* [5].

Na regressão logística multinomial, dada uma entrada x a ser classificada, é computado um *score* $s_k(x)$ para cada uma das k classes, e então a probabilidade de x pertencer a cada classe é obtida aplicando uma função exponencial normalizada, ou função softmax, aos *scores*.

A equação utilizada para o cômputo da probabilidade s_k é a mesma utilizada em regressores lineares, ou seja, é a soma do vetor de parâmetros θ multiplicado pelo vetor de características x. Porém, como cada classe possui seu próprio vetor de parâmetros θ_k, todos estes vetores são armazenados em uma matriz de parâmetros Θ.

Calculados os scores para cada uma das classes para o dado x, podemos estimar a probabilidade p_k que a instância pertença a uma certa classe k passando os *scores* obtidos através de uma função softmax, definida como

$$f_{softmax}(s(x)) = \frac{e^{s_k(x)}}{\sum_{j=1}^{K} e^{s_j(x)}}$$

onde K é o número de classes e $s(x)$ é um vetor contendo os scores de cada classe para o dado x. Ou seja, calculando a exponencial de cada *score* e a normalizando.Da mesma maneira que uma regressão logística simples, a regressão logística multinomial prevê que o dado pertence a classe com maior probabilidade estimada.

Como função de custo da regressão logística multinomial, utilizamos a função *cross-entropy*:

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{K} y_k^{(i)\log(p_k^(i))}$$

onde

$$y_k^{(i)} = \begin{cases} 1 & \text{se i pertence a classe k} \\ 0 & \text{caso contrário} \end{cases}.$$

Com esta função de custo, podemos utilizar a descida do gradiente ou outra técnica de minimização para encontrar a matriz Θ que minimize a função de custo $J(\Theta)$

C. Redes Neurais Artificiais

Redes neurais artificiais são assim chamadas por serem inspiradas na rede de neurônios animais. A rede é formada pelas camadas de entrada, N camadas escondidas e a camada de saída.

Os neurônios são interconectados para que, em cada um deles, seja realizada uma operação mediante um valor de entrada fornecido ao neurônio. A malha de interconexão dos neurônios se dá havendo uma aresta de conexão de um neurônio de uma dada camada com todos os demais da camada seguinte. Cada uma destas arestas tem um peso w_i associado,

sendo que este peso é atualizado durante o treino da rede visando minimizar a sua função de custo.

Cada neurônio tem a ele associado uma função de ativação, que é executada mediante um dado de entrada. A função de ativação pode ser uma sigmoide, uma tangente hiperbólica ou uma ReLU.

No caso de ser uma sigmoide, utilizamos a mesma função $g(x)$ mostrada acima para regressão logística. A tangente hiperbólica é definida por

$$h(x) = \tanh(x)$$

, já a função ReLU é definida como

$$i(x) = \max(0, x)$$

O dado de entrada, em cada neurônio nas camadas escondidas e na camada de saída, é composto pela somatória da multiplicação dos pesos pelos respectivos valores dos N neurônios da camada anterior. Além da saída dos neurônios da camada anterior, um fator de bias, contante, geralmente definido como 1, é adicionado como se fosse uma entrada do neurônio, tendo um peso w_0 associado.

Uma rede neural é utilizada para se fazer classificação de dados fornecidos na camada de entrada, apresentando os elementos de cada classe em um neurônio da camada de saída. Essa classificação pode ser para identificar se os dados pertencem a uma (sim ou não) ou mais classes (multi-classes). Cada neurônio na camada de saída de uma rede representa a probabilidade do dado propagado pela rede pertencer a uma classe associada. De maneira semelhante a regressão logística, a saída que retornar uma maior probabilidade é estimada como categoria da amostra.

Um tipo simplificado de arquitetura de rede neural artificial é o Perceptron. Ele é baseado em um neurônio artificial denominado unidade de limite linear (linear threshold unit - LTU), onde as entradas e a saída são números e cada conexão de entrada é associada a um peso. Dessa forma, o perceptron calcula a soma dos pesos de suas entradas e aplica a função step à soma e propaga o resultado na saída. z = w1 x1 + w2 x2 + + wn xn = wT ů x hw(x) = step (z) = step (wT ů x)

Pode-se utilizar uma única LTU para fazer uma simples classificação binária linear. Isso é realizado por meio de uma combinação linear das entradas e, se o resultado ultrapassar um limite, ele a classifica de forma positiva senão, como negativa; bem como ocorre em um classificador de Regressão Logística. O treinamento de uma LTU implica em encontrar os valores corretor dos pesos. Dessa forma, o Perceptron é composto de uma única camada de LTUs, com cada neurônio conectado a todas as entradas. Os neurônios da camada de entrada servem como neurônios de passagem pois apenas reproduzem o valor de entrada e a eles é adicionado um bias (x0 = 1). O bias é representado usando um tipo especial de neurônio chamado neurônio de polarização, que apenas exibe o valor 1 o tempo todo.

Um tipo mais elaborado de rede neural é chamado de rede neural multi-camadas, onde há N camadas escondidas. Neste tipo, cada camada, com exceção da camada de saída, inclui um neurônio de bias e tem cada um de seus neurônios conectados a todos os outros neurônios da camada seguinte. Redes neurais artificiais contendo uma ou mais camadas escondidas são também chamadas de redes neurais profundas.

Durante anos pesquisadores tentaram encontrar uma forma de treinar as redes neurais e em 1986 foi publicado um artigo [6] onde descreviam o algoritmo de treino *backpropagation*; também descrito como gradiente descendente em modo reverso autodiff. Nele, a cada instância de treinamento, o algoritmo a fornece para a rede e calcula a saída de cada neurônio em cada camada seguinte (esta é a passagem para a frente, bem como ocorre em predições). Feito isso, mede-se o erro de saída da rede, ou seja, a diferença entre a saída desejada e a saída real da rede, depois calcula o quanto cada neurônio na última camada oculta contribuiu para o erro de cada neurônio de saída. Depois, mede-se a quantidade dessas contribuições de erro que vieram de cada neurônio na camada escondida anterior; e assim sucessivamente até o algoritmo alcançar a camada de entrada.

Tal processamento reverso mede de forma eficiente o gradiente de erro em todos os pesos de conexão da rede, propagando o gradiente de erro para trás na rede; de onde vem o nome do algoritmo. O final do algoritmo de backpropagation é um passo de gradiente descendente em todos os pesos de conexão na rede, usando os gradientes de erro medidos anteriormente.

Em suma, o backpropagation

- Faz uma previsão (passagem para a frente)
- Mede o erro
- Passa em cada camada em sentido inverso para medir a contribuição de erro de cada conexão (passagem reversa)
- Ajusta os pesos de conexão para reduzir o erro (Descida do Gradiente)

Para que o algoritmo funcionasse de forma correta, os autores fizeram uma mudança importante: trocaram a função step pela função logística. O que foi essencial porque a função step contém uma mudança abrupta de valor, assim, não contém um gradiente para se trabalhar, enquanto a função logística possui uma derivada diferente de zero bem definida em todos os pontos, permitindo que Gradiente Descendente progrida em cada passo.

A função de soma de entrada de neurônio é definida como

$$f_{input} = \{w_0 + x_1 w_1 + \cdots + x_n w_n\}$$

A função de custo de cada neurônio é dependente da função de ativação do mesmo. O custo dos neurônios de saída é igual a

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{K} y{+}k^{(i)}\log(h_\Theta(x^{(i)}))_k{+}(1{-}y_k^{(i)})\log(1{-}h_\Theta(x^{(i)}))_k$$

III. Experimentos e Discussão

Nosso primeiro passo foi carregar o conjunto de treino e separá-lo em 80% como conjunto de treino efetivamente e 20% como conjunto de validação.Visando primeiro estabelecer qual modelo é o melhor adaptado para o caso estudado, primeiro treinamos e avaliamos os modelos acima de um subgrupo

de 20000 imagens do conjunto original de treino. Após isto, aplicamos o algorítimo de Analise de Componente Principal no conjunto de treino, onde testamos mantendo 99%, 95% e 90% da variância.

Table I
DIMENSÕES POR VARIÂNCIA PERMANECIDA

Variância	Dimensões
99%	656
95%	216
90%	99

Percebemos com isso que a redução de 1% da variância reduziria em 2416 dimensões, o que já é satisfatório, porém 5% da variância nos daria em volta de 200 dimensões a menos enquanto 10% da variância implicaria uma redução de apenas 100 dimensões de 5%, o que pode ser mais custoso em relação a variância do que em relação ao número de dimensões. Logo, escolhemos manter a variância dos dados a 95% da variância total, reduzindo de 3072 dimensões para 216.

Após isto, treinamos uma regressão logística, utilizando o esquema de um versus outros como solução multiclasses. Utilizamos 100 iterações máximas no algorítimo de minimização, utilizando normalização L2. Para este caso, a regressão foi capaz de predizer a classe de 39.2% dos exemplos.

Em sequência, treinamos uma regressão logística multinomial, com os mesmos parâmetros da regressão logística, e obtivemos um acerto de classe em 39.17% dos exemplos avaliados. Com isto, verificamos que o método de comparação entre classes não tem um efeito significativo no resultado. Porém, o tempo de execução da regressão logística multinomial é muito menor do que a simples, devido ao fato de que a simples executa uma regressão para cada classe.

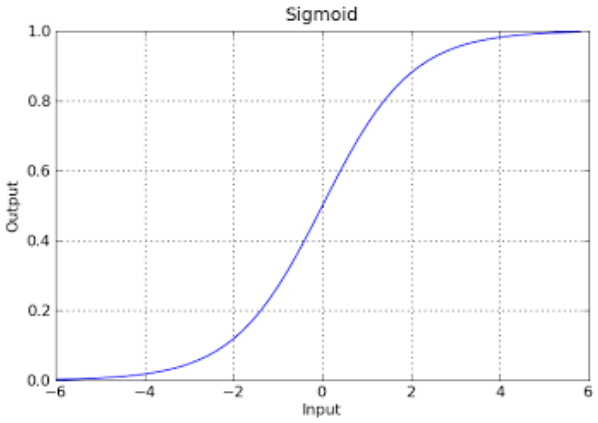
Passamos então para as soluções de redes neurais. Inicializamos três redes, cada uma com uma função de ativação (logística, tangente hiperbólica e ReLU, respectivamente), utilizando a Descida Estocástica do Gradiente como função de minimização, uma camada escondida com 216 neurônios, 10 neurônios de saída (um para cada classe), utilizamos treino em *batches* de 200 dados, com learning rate adaptativo (que se divide por 5 a cada vez que duas épocas consecutivas falharem em reduzir o custo) e 100 iterações.

Obtivemos 39.55% de sucesso para a função de ativação logística, 37.37% para tangente hiperbólica e 41.47% para ReLU. Aumentamos então o número máximo de iterações para 200, e obtivemos 39.32% para logística, 36.77% para tangente hiperbólica e 39.72% para ReLU. Com isto vimos que aumentando o número de interações, melhoramos a convergência da rede.

Logo em seguida, adicionamos mais uma camada escondida em nossas redes e executamos novamente. Obtivemos 38.75% de sucesso para a função de ativação logística, 37.50% para tangente hiperbólica e 9.82% para ReLU. Nisto percebemos que, para as funções de ativação logística e tangente hiperbólico tivemos um pequeno decréscimo na eficiência, enquanto para a ReLU tivemos uma queda grande no desempenho.

Nossa melhor rede, a com 100 iterações e utilizando o classificador ReLU em uma única camada escondida foi utilizada no conjunto de testes. Obtivemos % de precisão.

Figure 2. Plot da matriz de confusão para a melhor rede



IV. CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho, fomos capazes de perceber a eficiência das redes neurais em comparação com regressões logísticas. Apesar das redes demorarem mais para serem treinadas e serem computacionalmente mais exigentes, tanto em memória quanto em tempo de processamento, obtiveram resultados melhores que as regressões logísticas e classificaram corretamente 41% do conjunto de testes.

REFERENCES

[1] Christopher M. Bishop. “Pattern Recognition and Machine Learning”. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] Alex Krizhevsky et al. “The CIFAR-10 dataset”. https://www.cs.toronto.edu/ kriz/cifar.html.

[3] Bottou L. (2010) Large-Scale Machine Learning with Stochastic Gradient Descent. In: Lechevallier Y., Saporta G. (eds) Proceedings of COMPSTAT’2010. Physica-Verlag HD

[4] Mark Schmidt, Nicolas Le Roux, Francis Bach. "Minimizing Finite Sums with the Stochastic Average Gradient". Revision from January 2015 submission.

[5] Böhning, D. "Multinomial logistic regression algorithm". Ann Inst Stat Math (1992)

[6] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." Cognitive modeling 5.3 (1988): 1.