Relatório do trabalho parcial

disciplina SIN5016 - Aprendizado de máquina

Bruno Kemmer - NUSP 5910474

Junho 2020

1 Introdução

O objetivo desse trabalho é comparar a performance de alguns métodos de aprendizado de máquina tradicionais: classificadores lineares, regressão logística, máquinas de vetores de suporte e redes neurais em três datasets: *Iris Data Set*, *Pima Indians Diabetes Database* e *Hepatitis Data Set*.

2 Datasets

2.1 *Iris Data Set*

Dataset com 3 diferentes classes: Iris Setosa, Iris Versicolour, Iris Virginica. E com os seguintes atributos:

- sepal length (em cm)
- sepal width (em cm)
- petal length (em cm)
- petal width (em cm)

Como consta na descrição da atividade, o dataset foi dividido em 2/3 para treino e 1/3 para teste. Também foi codificada as três classes em:

- "Iris-setosa": [1,0,0]
- "Iris-versicolor": [0,1,0]
- "Iris-virginica": [0,0,1]

2.2 *Pima Indians Diabetes Database*

O dataset consiste em diferentes variáveis preditoras médicas (independentes) e uma variável classe (dependente). As variávies independentes incluem: número de gravidezes que a paciente teve, seu BMI, nível de insulina, idade, entre outros.

1. Pregnancies - Gravidezes - número de vezes em que a pessoa já engravidou
2. Glucose - Nível de glicose - concentração de de glicose no plasma em 2 horas, em um teste de tolerância oral.
3. BloodPressure - Diastolic blood pressure (mm Hg) - Pressão sanguínea diastólica
4. SkinThickness - Triceps skin fold thickness (mm) - Grossura da pele via o quanto é possível dobrar do tríceps
5. Insulin - Nível de insulina 2-Hour serum insulin (mu U/ml) -
6. BMI - Body mass index - Índice de massa corporal
7. Age - Idade
8. Outcome - Variável classe (0 ou 1) - 268 de 768 exemplos são da classe 1, os outros são.

Como o dataset só contém uma classe (binário), a codificação da variável dependente foi: 0: -1 e 1: +1. Inicialmente foi utilizado o critério de divisão aleatório, porém como foi disponibilizado o dataset dividido, esta, foi utilizada para treino e teste.

2.3 *Hepatitis Data Set*

- Número de instâncias: 155
- Tem valores faltantes: Sim
- Número de atributos: 19

Contém os seguintes atributos:

1. Class: DIE, LIVE
2. AGE: 10, 20, 30, 40, 50, 60, 70, 80
3. SEX: male, female
4. STEROID: no, yes
5. ANTIVIRALS: no, yes
6. FATIGUE: no, yes
7. MALAISE: no, yes
8. ANOREXIA: no, yes
9. LIVER BIG: no, yes
10. LIVER FIRM: no, yes
11. SPLEEN PALPABLE: no, yes
12. SPIDERS: no, yes
13. ASCITES: no, yes
14. VARICES: no, yes
15. BILIRUBIN: .39, .80, 1.20, 2.00, 3.00, 4.00
16. ALK PHOSPHATE: 33, 80, 120, 160, 200, 250
17. SGOT: 13, 100, 200, 300, 400, 500,
18. ALBUMIN: 2.1, 3.0, 3.8, 4.5, 5.0, 6.0
19. PROTIME: 10, 20, 30, 40, 50, 60, 70, 80, 90
20. HISTOLOGY: no, yes

Dataset binário e sua codificação foi feita como: 1: -1 e 2: +1.

Como nesse dataset tem dados faltantes, estes foram completados com a média de suas variáveis (colunas).

3 Normalizações

Para todos os datasets foi primeiramente treinado sem nenhuma normalização, normalizando todas as colunas para *z_score* - uma distribuição normal padrão (média 0 e desvio padrão 1) e também todas as colunas para *minmax* - *mínimo e máximo*. Foi tomado o cuidado de utilizar somente os dados de treinamento, e aplicar essas mesmas agregações (do dataset de treinamento) nos dados de teste no momento de inferência.

Como os melhores resultados foram obtidos usando a normalização *z_score*, esta, será a única presente nos resultados.

4 Algoritmos Utilizados

O desenvolvimento dos algoritmos foi feito usando a biblioteca de cálculos Numpy na linguagem Python, e sempre que possível utilizando a forma matricial para obter uma melhor performance nos cálculos.

4.1 Classificador Linear

Implementação do algoritmo: classificador linear, seu objetivo é encontrar o hiperplano de separação que obedeça a equação:

$$w_1x_1 + w_2x_2 + ... + w_dx_d + w_0 = 0 \tag{1}$$

Sua inclinação é determinada por $(w_1, w_2, ..., w_d)$ e sua localização por w_0. Sua predição é:

$$\hat{y} = sinal(\sum_{i=0}^{d} w_i x_i) \tag{2}$$

Com $x_0 = 1$.

O erro quadrático é utilizado para medir sua performance:

$$E(\vec{w}) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{3}$$

Também adicionamos ao erro total um termo de regularização:

$$E_{\vec{w}}(\vec{w}) = \vec{w}^T\vec{w} \tag{4}$$

Portanto o erro total é:

$$E_{Total} = E(\vec{w}) + \gamma E_{\vec{w}}(\vec{w}) \tag{5}$$

Em que γ é o coeficiente de regularização. Podemos obter a seguinte fórmula para o cálculo dos coeficientes:

$$\vec{w} = (X^TX + N\gamma I)^{-1}X^T\vec{y} \tag{6}$$

Em que N é a quantidade de exemplos de treinamento.

Esta foi a implementação utilizada e nos testes foi variado o valor de γ.

4.2 Regressão Logística

Algoritmo de classificação utilizado baseado na probabilidade condicional da classe, dados aos atributos previamente observados. A sua predição é feita da forma:

$$P(y|\vec{x}) = \theta(y\vec{w}\vec{x}) \tag{7}$$

$$\theta(\vec{s}) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}} \tag{8}$$

Função de erro:

$$E(\vec{w}) = \frac{1}{N}\sum_{i=1}^{N}\ln(1 + e^{-y_i\vec{w}\vec{x_i}}) \tag{9}$$

E para determinar os valores de $\vec{w}$ foi utilizado o algoritmo de gradiente descendente.

$$\nabla E(\vec{w}) = -\frac{1}{N}\sum_{i=1}^{N}\frac{y_i\vec{x_i}}{1+e^{y_i\vec{w_t}\vec{x_i}}} \tag{10}$$

$$\vec{w_{t+1}} = \vec{w_t} - \eta\nabla E(\vec{w}) \tag{11}$$

Para os datasets multiclasse foi utilizado a variação com a função softmax.

4.3 Máquina de vetores de suporte

Implementação do algoritmo: máquina de vetores de suporte *(Support Vector Machine)* proposto por [Vapnik, 1995][1].

A predição é feita utilizando os vetores de suporte, pela equação:

$$\hat{y} = sinal(\sum_{i=1}^{m}(\alpha_i y_i K(x, x_n) + b)) \tag{12}$$

Em que a função K é a função de *Kernel*, ela permite transformar os dados para um espaço de dimensão mais alta que o atual e neste novo espaço é que o algoritmo fará a separação linear dos dados.

Para encontrar quais são os vetores de suporte, e seus multiplicadores de Lagrange $(\alpha_i, ..., \alpha_m)$ é necessário solucionar o seguinte problema quadrático:

$$\begin{aligned} \min_{\alpha} \quad & W(\alpha) = \frac{1}{2}\sum_{i,j}^{N}\alpha_i\alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{N}\alpha_i \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq c \quad i = 1, ..., N \\ & \sum_{i=1}^{N}\alpha_i y_i = 0 \end{aligned} \tag{13}$$

O parâmetro c é um regularizador utilizado para controlar o quanto é permitido que os exemplos sejam classificados de forma incorreta.

O termo de viés *(bias)* pode ser calculado como a média dos vetores de suporte:

$$b = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}) \tag{14}$$

Em que m é a quantidade de vetores de suporte obtidos na otimização do problema quadrático.

Para a resolução do problema quadrático foi utilizada a biblioteca *CVXOPT (Python Software for Convex Optimization)*.

Adicionalmente foram utilizados somente os exemplos em que os multiplicadores de Lagrange tinham valores acima de um limiar definido (10^{-5}) como vetores de suporte e para o cálculo do viés (bias).

Para o cálculo do viés, ao remover os exemplos que estavam com seu Lagrangeano dentro de faixa de valores: $(C - limiar)$, houve uma perda de performance nos 3 datasets analisados, por esse motivo, esse último filtro não foi adotado.

4.4 Máquina de vetores de suporte gêmeas - TW-SVM

Também foi implementado o algoritmo: Máquina de vetores de suporte gêmeas *(Twin Support Vector Machines)* como descrito no artigo [Huang, 2018][2] e proposto por [Jayadeva, 2007][3]. Nele são determinados dois planos não paralelos, cada um o mais próximo de uma das classes e distante da outra. Sua formulação com Kernel é a seguinte:

A são os atributos dos exemplos da classe positiva, e B os mesmos, mas da classe negativa, e_1 e e_2 são vetores de valores 1 de dimensão apropriada.

$$C^T = \begin{bmatrix} A, & B \end{bmatrix}^T \tag{15}$$

$$S = \begin{bmatrix} K(A, C^T) & e_1 \end{bmatrix} \tag{16}$$

$$R = \begin{bmatrix} K(B, C^T) & e_2 \end{bmatrix} \tag{17}$$

$$Z_1 = -(S^T S)^{-1} R^T \alpha \tag{18}$$

$$Z_1 = \begin{bmatrix} u_1 & b_1 \end{bmatrix}^T \tag{19}$$

$$L = \begin{bmatrix} K(A, C^T) & e_1 \end{bmatrix} \tag{20}$$

$$N = \begin{bmatrix} K(B, C^T) & e_2 \end{bmatrix} \tag{21}$$

$$Z_2 = (N^T N)^{-1} L^T \gamma \tag{22}$$

$$Z_2 = \begin{bmatrix} u_2 & b_2 \end{bmatrix}^T \tag{23}$$

Calculada as matrizes, é necessário resolver dois problemas de minimização:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^T R (S^T S)^{-1} R^T \alpha - e_2^T \alpha \\ \text{s.t.} \quad & 0 \leq \alpha \leq c_1 \end{aligned} \tag{24}$$

Sua solução determina o plano da classe positiva:

$$K(x^T, C^T) u_1 + b_1 = 0 \tag{25}$$

E para o determinar o segundo plano:

$$\begin{aligned} \min_{\gamma} \quad & \frac{1}{2} \gamma^T L (N^T N)^{-1} L^T \gamma - e_1^T \gamma \\ \text{s.t.} \quad & 0 \leq \gamma \leq c_2 \end{aligned} \tag{26}$$

Plano da classe negativa:

$$K(x^T, C^T) u_2 + b_2 = 0 \tag{27}$$

Para solucionar os dois problemas de minimização foi utilizado o método: Sequencial Least Squares Programming, presente na biblioteca Scipy[4]. Nos termos regularizadores, foi utilizado o mesmo valor c, tanto para c_1 quanto c_2.

Para evitar problemas na inversão das matrizes em (18) e (22) foi utilizado o método de cálculo da pseudo-inversa de *Moore-Penrose*.

Na predição foi utilizada as seguintes regras:

1. Caso o cálculo do plano positivo(25) dê positiva, a classe será predita como positiva.
2. Caso o cálculo do plano negativo(27) dê negativo, a classe será predita como negativa.
3. Neste caso o ponto está entre os dois planos e será necessário ver para qual plano o ponto está mais próximo, e este será a classe predita.

Foram testadas duas funções de kernel: linear e polinomial.

4.5 Rede Neural

Implementação de uma rede neural de duas camadas: uma camada com uma função de ativação não-linear e uma camada de saída com a função *Softmax*. A função de custo utilizada foi a entropia cruzada:

$$J(\vec{w}) = -\frac{1}{N} \sum_{i=1}^{N} y \ln \hat{y} \tag{28}$$

Na primeira camada foram feitos testes com duas funções de ativação não-linear: *Sigmoid* (8) e *Rectified Linear Unit (ReLU)*, esta última, é definida como:

$$f(x) = \begin{cases} x, & \text{se } x > 0 \\ 0, & \text{caso contrário} \end{cases} \tag{29}$$

Após passar pela primeira camada temos:

$$\vec{Z_1} = \vec{X} \cdot \vec{W_1} + \vec{b_1} \tag{30a}$$

$$\vec{A} = f(\vec{Z_1}) \tag{30b}$$

E na camada de saída temos a função *Softmax* para determinar para qual classe será feita a predição:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{31}$$

Para evitar possíveis problemas de *overflow* foi somado a mesma constante tanto no numerador quanto no denominador, sendo essa constante o valor máximo da matriz, deixando assim a função mais estável.

$$softmax(A) \leftarrow softmax(A - max(A)) \tag{32}$$

Logo na camada de saída temos:

$$\vec{Z_2} = \vec{A} \cdot \vec{W_2} + \vec{b_2} \tag{33a}$$

$$\hat{y} = softmax(\vec{Z_2}) \tag{33b}$$

Com isso temos as predições na época atual, nesse ponto é utilizado o algoritmos de retropropagação do erro para atualização dos parâmetros (W_1, b_1, W_2, b_2).

$$\frac{\partial J}{\partial W_1} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial A} \cdot \frac{\partial A}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial W_1} \tag{34}$$

$$\frac{\partial J}{\partial b_1} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial A} \cdot \frac{\partial A}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial b_1} \tag{35}$$

$$\frac{\partial J}{\partial W_2} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial W_2} \tag{36}$$

$$\frac{\partial J}{\partial b_2} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial b_2} \tag{37}$$

As derivadas parciais são as seguintes:

$$\frac{\partial J}{\partial \hat{Z_2}} = \frac{1}{N}(\hat{y} - y) \tag{38}$$

Devido ao fato de $\sum y = 1$ para cada exemplo.

$$\frac{\partial Z_2}{\partial \hat{W_2}} = \vec{A} \tag{39}$$

$$\frac{\partial Z_2}{\partial \hat{b_2}} = 1 \tag{40}$$

Se a função de ativação utilizada tenha sido a *Sigmoid*:

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x) \cdot (1 - \sigma(x)) \tag{41}$$

Caso a função de ativação tenha sido *Rectified Linear Unit (ReLU)*:

$$\frac{\partial\, relu(x)}{\partial x} = \begin{cases} 1, & \text{se } x > 0 \\ 0, & \text{caso contrário} \end{cases} \tag{42}$$

Logo, para calcular $\frac{\partial A}{\partial Z_1}$ será preciso aplicar em cada elemento da matriz.

$$\frac{\partial J}{\partial W_1} = \frac{\partial J}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial A} \cdot \frac{\partial A}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial W_1} \tag{43}$$

$$\frac{\partial Z_1}{\partial W_1} = \vec{X} \tag{44}$$

$$\frac{\partial Z_1}{\partial b_1} = 1 \tag{45}$$

Para os testes, foram variados os seguintes hyperparâmetros: a quantidade de neurônios na camada oculta, taxa de aprendizado e a quantidade de épocas de treinamento.

A inicialização das camadas foi feita de modo aleatório e multiplicado por um fator .01 para manter os valores em regiões estáveis das funções.

5 Resultados

5.1 Validação cruzada

Nos testes foi utilizada a técnica de validação cruzada, com exceção do dataset *Pima Indians Diabetes Database*, no qual, foi disponibilizado os dados já separados entre treino e teste. Nos outros dois, foi separado de forma aleatória com 70% dos dados para treino, e 30%, para teste.

5.2 *Iris Data Set*

Pela tabela 1, podemos ver que o dataset pode ser totalmente classificado corretamente. Porém ao utilizar a técnica um-contra-todos *(one-vs-all)* não foi obtida uma boa performance (menos de 60% de acurácia) utilizando máquinas de vetores de suporte. Um dos possíveis motivos, é o desbalanceamento que esse método gera nos dados. Mesmo usando um classificador linear sem regularização, usando sua formulação multi-classe, é possível obter 84% de acurácia, o que mostra que esse problema não é tão complexo. Tanto que, pelo método de regressão logística foi possível fazer a classificação de 100% do dataset de teste.

Algoritmo	**Hyperparâmetros**	**Acurácia**
Regressão Logística	'Taxa de aprendizado': .5, 'Max Iter': 1000	1
Rede Neural	'Função de ativação': 'sigmoid', 'Número de neurônios': 1, 'Taxa Aprendizado': '.5', 'Épocas': 1000	1
Rede Neural	'Função de ativação': 'relu', 'Número de neurônios': 1, 'Taxa Aprendizado': '.5', 'Épocas': 1000	1
Classificador Linear	'Regularizador': 0	.84
Classificador Linear	'Regularizador': .5	.8
Classificador Linear	'Regularizador': 1	.8

Tabela 1: Resultados selecionados obtidos no dataset Iris, usando a técnica de validação cruzada.

5.3 *Pima Indians Diabetes Database*

Algoritmo	**Hyperparâmetros**	**Acurácia**
Rede Neural	'Função de ativação': 'Sigmoid', 'Número de neurônios': 10, 'Taxa Aprendizado': '5', 'Épocas': 100	.7739
Rede Neural	'Função de ativação': 'Relu', 'Número de neurônios': 10, 'Taxa Aprendizado': '1', 'Épocas': 10000	.7739
Regressão Logística	'Taxa de aprendizado': 10, 'Max Iter': 100	.7739
Classificador Linear	'Regularizador': .5	.7696
Rede Neural	'Função de ativação': 'Sigmoid', 'Número de neurônios': 10, 'Taxa Aprendizado': '.5', 'Épocas': 100	.7652
SVM	'c': 1, 'Kernel': 'Linear'	.7609
TWSVM	'c': 2, 'Kernel': 'Polinomial', 'Grau': 2	.7565
Classificador Linear	'Regularizador': 0	.7565

Tabela 2: Resultados selecionados obtidos no dataset Diabetes, usando a técnica de validação cruzada.

Os resultados obtidos, Tabela 2, mostram que esse problema é mais complexo de ser classificado que os outros. As melhores performances foram obtidas com redes neurais, porém não

houve muita diferença alterar a quantidade de neurônios ou a quantidade de épocas de treinamento. Uma possibilidade é que eles foram inicializados randomicamente usando a mesma semente, logo, seus pesos se inicializaram iguais, o que pode ter levado para o mesmo mínimo local.

5.4 *Hepatitis Data Set*

Algoritmo	**Hyperparâmetros**	**Acurácia**
TWSVM	{'c': .5, 'Kernel': 'Polinomial', 'Grau': 4}	.8936
TWSVM	{'c': 2, 'Kernel': 'Polinomial', 'Grau': 2}	.8723
Rede Neural	{'Função de ativação': 'Sigmoid', 'Número de neurônios': 10, 'Taxa Aprendizado': '.5', 'Épocas': 10000}	.8723
Rede Neural	{'Função de ativação': 'Sigmoid', 'Número de neurônios': 100, 'Taxa Aprendizado': '5', 'Épocas': 1000}	.8723
Rede Neural	{'Função de ativação': 'Relu', 'Número de neurônios': 10, 'Taxa Aprendizado': '5', 'Épocas': 1000}	.8723
Classificador Linear	{'Regularizador': 0}	.8298
SVM	{'c': 1, 'Kernel': 'Linear'}	.8298

Tabela 3: Resultados selecionados obtidos no dataset Hepatitis, usando a técnica de validação cruzada.

Como podemos ver pela Tabela 3 o método TWSVM utilizando o kernel polinomial com grau 4 foi o que obteve a melhor performance no dataset de teste (utiizando a técnica de validação cruzada), porém, como a diferença para o grau 2 foi baixa, é preferível utilizar este segundo modelo. A rede neural com poucos neurônios na camada oculta e a partir de 1.000 épocas também apresentou resultados promissores, no caso esta, foi com a função de ativação "Relu".

Referências

[1] Corinna Cortes e Vladimir Vapnik. "Support-vector networks". Em: *Machine Learning* 20.3 (set. de 1995), pp. 273–297. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/BF00994018. URL: http://link.springer.com/10.1007/BF00994018 (acesso em 21/06/2020).

[2] Huajuan Huang, Xiuxi Wei e Yongquan Zhou. "Twin support vector machines: A survey". Em: *Neurocomputing* 300 (jul. de 2018), pp. 34–43. ISSN: 09252312. DOI: 10.1016/j.neucom.2018.01.093. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231218302923 (acesso em 21/05/2020).

[3] Jayadeva, R. Khemchandani e S. Chandra. "Twin Support Vector Machines for Pattern Classification". Em: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.5 (2007), pp. 905–910.

[4] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". Em: *Nature Methods* 17 (2020), pp. 261–272. DOI: https://doi.org/10.1038/s41592-019-0686-2.