Desenvolvimento de redes MLP em um problema de classificação com tratamento de dados espúrios.

Helon de Franca Medeiro [1], Tiago César Cunha da Costa [2]

[1]Universidade Federal Rural do Rio de Janeiro
Seropédica – RJ – Brazil

{helonfranca02@ufrrj.br, tiago.person.student@gmail.com}

RESUMO

Este trabalho foi feito para a disciplina de Inteligência Artificial do curso de Sistemas de Informação da Universidade Federal Rural do Rio de Janeiro. Ele teve como objetivo realizar o tratamento e análise de um conjunto de dados brutos e, posteriormente, treinar e avaliar redes neurais artificiais do tipo MLP (Perceptron Multicamadas) para um problema de classificação binária. Utilizou-se a linguagem Python, juntamente com bibliotecas como Pandas, NumPy, Matplotlib, scikit-learn, seaborn e Jupyter Notebook para manipulação e análise dos dados, normalização dos valores, cálculo da média, desvio padrão e correlação cruzada entre as variáveis. Teve-se como resultado final a seleção do melhor modelo de rede neural MLP para o problema proposto e a análise completa do desempenho de todas as redes treinadas. Os resultados foram apresentados em um relatório escrito e os códigos e modelos treinados foram disponibilizados no GitHub.

PALAVRAS-CHAVE: Inteligência Artificial, MLP, Perceptron multicamadas, Análise de dados, Tratamento de dados, Normalização linear.

1. INTRODUÇÃO E TRATAMENTO DOS DADOS (FASE 1)

Utilizou-se:
- a linguagem Python;
- com a biblioteca Pandas (para manipulação e análise de dados),
- NumPy (computação científica - utilizada nas curvas gaussianas),
- Matplotlib (para os gráficos/visualizações),
- scikit-learn (MinMaxScaler - para realizar a normalização de valores),
- seaborn (para o gráfico da correlação das variáveis - heatmap),
- Jupyter Notebook (Jupyter Lab - ambiente de desenvolvimento interativo com *'notebooks'*).
- sklearn.metrics: para avaliar com métricas, como matriz de confusão (fase 2 antiga).
- sklearn.neural_network.MLPClassifier: para criar e treinar modelos de RNA.
- joblib: para salvar e carregar objetos Python(modelos, variáveis).
- os: para as operações relacionadas ao sistema operacional, como criar diretórios.

a. TRATAMENTO DE DADOS ESPÚRIOS:

Os dados a seguir tiveram que passar pelo seguintes modos de tratamento:

1- “A vírgula”: foi necessário fazer a substituição das "vírgulas" dos valores de toda a tabela, por "pontos": pois o valor decimal no Python, é separado por "pontos" (e não por "vírgulas" como nas planilhas).

2- Valores negativos e vazios: substitui-se os seguintes valores espúrios usando o método de trocá-los pela média de sua coluna (desconsiderando o seu valor para o cálculo da média):

- Negativos
- Vazios/NaN (representados aqui por NaN).

3- Outliers: capturou-se os valores “fora da curva” através do o método de detecção de outliers baseado em desvio padrão (número de desvios padrão a partir da média). Previamente, plotou-se um gráfico que mostrava a distribuição desses dados (a dispersão, através de pontos), e pode-se observar aqueles pontos que estavam mais longe da maioria dos outros. Em seguida, manualmente viu-se que o valor apropriado, para capturar apenas esses valores **exageradamente** grandes, ou exageradamente pequenos de cada coluna, era o valor do limite setado para 5 desvios padrão a partir da média. E devido a natureza dos gráficos, pela análise, removeu-se apenas valores acima da maioria (maiores).Os armazenando em um dicionário.
Substitui-se os valores considerados outliers de cada coluna, pela média daquela coluna. Tanto o gráfico quanto os outliers (no código final), estão sendo impressos na tela, com os resultados dos outliers abaixo.

B. NORMALIZAÇÃO:

Utilizou-se a normalização Min-Max, em que cada valor é substituído pelo seu equivalente normalizado usando a fórmula:

valor_normalizado = (valor - valor_min) / (valor_max - valor_min)

onde:

- valor é o valor original,
- valor_min é o valor mínimo (da respectiva coluna).
- e valor_max é o valor máximo (da respectiva coluna).

Assim, o DataFrame (df_normalizado) contém os dados normalizados, onde os valores de cada coluna estão no intervalo [0, 1].

C. CÁLCULO DA MÉDIA, DESVIO PADRÃO, CORRELAÇÃO CRUZADA ENTRE CADA PAR DE VARIÁVEIS (E TABELA AUXILIAR):

Seguiu-se o enunciado do problema. Sendo que para a correlação, utilizou-se a correlação de Pearson.

D. ANÁLISE DA SIGNIFICÂNCIA DE CADA VARIÁVEL EM RELAÇÃO À CLASSIFICAÇÃO (... - segue-se o enunciado):

Seguiu-se o enunciado do problema. E plotou-se as curvas gaussianas de cada classe, de cada variável.

E. DECISÃO DE QUAIS VARIÁVEIS SERÃO SELECIONADAS E QUAIS SERÃO DESCARTADAS:

1- Analisando as curvas gaussianas já impressas, avaliou-se as variáveis, e se retirou aquelas em que não se tinha muito poder de discriminar entre as duas classes. Isso foi feito manualmente, pela análise gráfica.
Os resultados da primeira análise estão descritos abaixo na tabela.
Para ajudar na identificação, foram usados símbolos coloridos de "correto" e "errado": quando as classes de uma variável possuíam diferentes valores (com uma delas obtendo mais densidade dentro de um determinado intervalo), a variável era selecionada, e passava pela análise (✔️).
Do contrário, não passava (❌).

2- Usando a correlação de Pearson já feita, imprimiu-se um mapa de calor para deixar as correlações mais visíveis (e "mais analisáveis"), e assim decidiu-se por considerar as relações fortes, quando seu valor de correlação fosse a partir de 0.6.
Assim se obteve relação forte entre:

- v1, v5 e v10 (escolheu-se v1).
- v7 e v11 (escolheu-se v7).
- v13 e v14: (escolheu-se v14)

Para critério de decisão, escolheu-se manualmente, entre as curvas que se mostravam com um pico levemente maior, e que ao mesmo tempo, abrangiam um intervalo maior de valores com densidade considerável.

Por fim, a escolha ficou:

Variáveis	Análise 1	Análise 2	Resultado Final
V1	✔️	✔️	✔️
V2	✔️	-	✔️
V3	❌	-	
V4	❌	-	
V5	Iria	❌	
V6	❌	-	
V7	✔️	✔️	✔️

V8	✔️	-	✔️
V9	❌	-	
V10	Iria	❌	
V11	Iria	❌	
V12	✔️	-	✔️
V13	❌	❌	
V14	✔️	✔️	✔️

Legenda:
“✔️” - passou.
“❌” - não passou.
“Iria” - passou nessa etapa, mas não na outra.
“-” - nada mudou, continua com o estado anterior.
Vazio: não passou no final de tudo.

Tudo isso pode ser conferido na pasta “*Fase 1*”, presente em nosso diretório do GitHub, que possui o link no final deste documento.

2. TREINAMENTO DA MLP (FASE 2):

Nosso código da fase 2 realiza um processo de treinamento e avaliação de modelos de Rede Neural Artificial (RNA) utilizando a biblioteca Scikit-learn no contexto de classificação binária. Nessa fase, se criou 2 pacotes de arquivos/programas. O primeiro, foi enviado a professora, e possui uma arquitetura que treina cada modelo de rede 20 vezes, porém, só salva dados em um arquivo sobre cada um dos 27 grupos (pois se tem uma combinação de 3 * 9, como será explicado abaixo), contando com seus hiperparâmetros, média e desvio padrão. E aqui não há treinamento com validação cruzada (cross-validation).

Nosso código realiza uma busca por diferentes combinações de hiperparâmetros, sendo:

3 valores capazes para o número de camadas ocultas (10, 5), (20, 10), (30, 15).

3 valores capazes para a taxa de aprendizado 0.01, 0.1, 0.2.

3 valores capazes para o momentum 0.5, 0.9, 0.99.

O número total de combinações possíveis é dado pelo produto cartesiano entre esses conjuntos de valores:

Número de combinações = 3 * 3 * 3 = 27

Dessarte, o código irá treinar e avaliar 27 modelos de redes neurais, cada um com uma combinação específica de hiperparâmetros.

No intuito de melhorar nossa rede neural e cumprir ainda mais com o enunciado que foi pedido, se implementou o treinamento com validação cruzada (cross-validation) na nova

fase 2, e também, salvou-se 540 arquivos, que correspondem aos 20 treinos de cada uma das 27 arquiteturas de hiperparâmetro das redes (20 * 27).

Sendo assim, neste artigo, e para o trabalho final, considerou-se como fase 2, a fase 2 final (a última desenvolvida, presente no diretório da fase 3 - nomeada para fins de simplicidade e fácil entendimento como "*MLP - Com treinos CV (agora exporta certo - eh da fase 2)*").

Seguindo o enunciado:

a. Foi selecionada uma biblioteca para a linguagem de programação Python, especificamente a biblioteca "scikit-learn", que fornece diversas ferramentas para aprendizado de máquina, incluindo a implementação de redes neurais artificiais do tipo Perceptron Multicamadas (MLP).

b. e c. : "O programa testa e salva modelos de redes neurais com várias configurações, realizando múltiplos treinamentos para estatísticas mais precisas."

Foi desenvolvido um programa que realiza o teste de vários modelos de redes neurais, com diferentes configurações. O programa utiliza um loop aninhado para percorrer todas as combinações possíveis de hiperparâmetros, como o número de neurônios em cada camada escondida (hidden_layer_sizes), taxas de aprendizado (learning_rate_init) e valores de momentum (momentum). Para cada combinação de hiperparâmetros, são realizados múltiplos treinamentos (num_trainings) para obter uma estatística mais realista de cada modelo. As acurácias dos modelos são armazenadas em uma lista para posterior análise.

O arquivo fonte jupyeter (python) em que a rede foi treinada, na fase 2, foi nomeado de forma bem simples e comum para fim de entendimento rápido, optando-se por remover o acento do "É" e fazer uma troca pelo "H".
"MLP - Com treinos CV (agora exporta certo - eh da fase 2)"

O código da fase 2 salva os modelos treinados em arquivos separados na pasta "modelos", que totalizam 540 modelos, usando a biblioteca joblib. Os nomes dos arquivos incluem informações sobre os hiperparâmetros e a acurácia média do modelo para fácil identificação.

Algumas partes interessantes e um pouco de explicação:

1- Importação de bibliotecas:

O código começa importando diversas bibliotecas como pandas, numpy, matplotlib, scikit-learn (sklearn), joblib, os e json - para manipulação e análise de dados, geração de gráficos, métricas de avaliação de modelos e operações relacionadas ao sistema operacional.

2- Carregamento do Dataset:

Um conjunto de dados é lido a partir de um arquivo ".CSV" usando a biblioteca pandas e é armazenado em um dataframe chamado "dataset". A primeira coluna é removida do dataframe.

3- Divisão do conjunto de dados:

O conjunto de dados é dividido em conjuntos de treinamento e teste usando a função “train_test_split” do scikit-learn. Essa divisão é feita em uma proporção de 70% para treinamento e 30% para teste.

4- Treinamento e avaliação dos modelos:

Define-se uma função chamada "train_and_evaluate_model" que treina e avalia um modelo de Rede Neural Artificial com hiperparâmetros específicos. Os hiperparâmetros testados são: tamanho das camadas ocultas, taxa de aprendizado inicial e momento.

Utiliza-se de loops aninhados para testar diferentes combinações de hiperparâmetros - que totalizam 9*3 grupos de hiperparâmetros iguais -. Ele treina cada combinação várias vezes (definido por "num_de_treinos" que é igual a 20) e calcula a acurácia usando validação cruzada (com 5 folds) para cada modelo treinado.
Os modelos são salvos em arquivos com seus hiperparâmetros e acurácia em seus nomes.

5- Pós-processamento (a partir dessa parte, os tópicos a seguir não foram literalmente pedidos na fase 2, mas foi feita para fins de conhecimento):

Os modelos são ordenados pela maior acurácia e as estatísticas de cada modelo (hiperparâmetros e acurácia média) são impressas no console.
As estatísticas também são exportadas para um arquivo de texto.

6- Seleção do melhor modelo:

O modelo com a maior acurácia é selecionado. Ele é novamente treinado usando todos os dados de treinamento. O modelo final é avaliado no conjunto de teste e a matriz de confusão é impressa no console.

7- Exportação dos resultados:

É feita as exportações necessárias para continuar o código em um outro arquivo python. A lista de todos os modelos treinados (com seus hiperparâmetros e acurácia) é exportada para um arquivo Python usando a biblioteca joblib. As variáveis necessárias foram capturadas no momento em que se executa o código e foram exportadas também.

Quanto aos arquivos gerados e seus locais de armazenamento (explicitando):

A fase 2 gera e guarda os 540 modelos treinados na pasta “modelos” no formato “pickle”. E em outro momento, ordena esses modelos por grupos (de hiperparâmetros iguais), armazenando seus valores de acurácia (ou seja, temos 27 grupos, e 20 valores de acurácia em cada) na pasta “estatistica_por_grupos_no_formato_json”, em formato “.json”. É exportado tambem um arquivo de texto legível com os 540 modelos, acurácia e seus valores quando agrupador pelo hiperparametro ("estatisticas_modelos.txt").

Quanto aos seus hiperparâmetros:

São utilizados três hiperparâmetros principais:

- hidden_layer_sizes: define a quantidade e o tamanho das camadas ocultas da rede. Cada valor nessa lista representa uma tupla com o número de neurônios em cada camada oculta. Por exemplo, (10, 5) significa que há uma camada oculta com 10 neurônios e outra camada oculta com 5 neurônios. O código testa três combinações diferentes: (10, 5), (20, 10) e (30, 15).

- learning_rate_init: controla a taxa de aprendizado inicial da rede neural (utilizando-se o método DA “Descida do Gradiente Estocástico” - Stochastic Gradient Descent - SGD). O código testa três valores diferentes: 0.01, 0.1 e 0.2.

- momentum: O momentum é um hiperparâmetro que ajuda a acelerar a convergência do algoritmo de otimização. O código testa três valores diferentes: 0.5, 0.9 e 0.99.

Com o fim de não estender esse documento, não será feita uma tabela com os 540 modelos, pois eles também podem ser conferidos no arquivo "estatisticas_modelos.txt"

Abaixo, apresenta-se os dados de acurácia média para cada “grupo de hiperparâmetro” (provindos de “estatisticas_modelos.txt”).

Hiperparâmetros:	Média de Acurácia:	Desvio Padrão de Acurácia:
(10, 5), 0.01, 0.5	0.6657	0.0269
(10, 5), 0.01, 0.9	0.7096	0.0309
(10, 5), 0.01, 0.99	0.6426	0.0331
(10, 5), 0.1, 0.5	0.7035	0.0314
(10, 5), 0.1, 0.9	0.7052	0.0296
(10, 5), 0.1, 0.99	0.6644	0.0374
(10, 5), 0.2, 0.5	0.6650	0.0347
(10, 5), 0.2, 0.9	0.6805	0.0244
(10, 5), 0.2, 0.99	0.6574	0.0363
(20, 10), 0.01, 0.5	0.6936	0.0187
(20, 10), 0.01, 0.9	0.7493	0.0188

(20, 10), 0.01, 0.99	0.6773	0.0361
(20, 10), 0.1, 0.5	0.7209	0.0170
(20, 10), 0.1, 0.9	0.7041	0.0312
(20, 10), 0.1, 0.99	0.6834	0.0397
(20, 10), 0.2, 0.5	0.6699	0.0313
(20, 10), 0.2, 0.9	0.7132	0.0298
(20, 10), 0.2, 0.99	0.6690	0.0383
(30, 15), 0.01, 0.5	0.6896	0.0154
(30, 15), 0.01, 0.9	0.7515	0.0290
(30, 15), 0.01, 0.99	0.6933	0.0413
(30, 15), 0.1, 0.5	0.7086	0.0225
(30, 15), 0.1, 0.9	0.7101	0.0275
(30, 15), 0.1, 0.99	0.7111	0.0367
(30, 15), 0.2, 0.5	0.6672	0.0322
(30, 15), 0.2, 0.9	0.7152	0.0199
(30, 15), 0.2, 0.99	0.6798	0.0289

FASE 3

Nosso código na fase 3 tem como objetivo realizar a análise de desempenho das redes neurais treinadas na Fase 2 do projeto. Ele carrega os modelos, testa cada um deles com os dados de teste e salva os resultados em um arquivo de texto. Além disso, realiza a seleção do melhor modelo com base em sua acurácia e imprime a matriz de confusão correspondente. Essas etapas ajudam a determinar quais modelos têm melhor desempenho e qual é o melhor modelo geral para o problema proposto.

1- Importação de bibliotecas: As bibliotecas necessárias para o treinamento e avaliação das redes neurais são importadas, incluindo NumPy para computação científica, Matplotlib para geração de gráficos, e diversas funções e classes do scikit-learn, como MLPClassifier para criar e treinar redes neurais.

2- Importação das variáveis de treinamento e teste: O código importa as variáveis de treinamento e teste que foram exportadas em uma etapa anterior do projeto. Essas variáveis contêm os dados e rótulos necessários para treinar e avaliar as redes neurais.

3- Teste e salvamento dos resultados em um arquivo de texto: Cada modelo de rede neural é testado individualmente usando o conjunto de teste, e os resultados de desempenho são salvos em um arquivo de texto. Além disso, a média do desempenho do conjunto de redes com os mesmos hiperparâmetros é calculada e também é salva no arquivo.

4- Carregamento dos modelos treinados: Os modelos previamente treinados foram salvos em arquivos. O código carrega esses modelos de volta e os armazena em uma lista para posterior avaliação.

5- Cálculo da média do desempenho para cada grupo de modelos: Os modelos são agrupados por hiperparâmetros e suas acurácias são calculadas. Os resultados são salvos no mesmo arquivo de texto mencionado anteriormente.

6- Seleção das 10 melhores redes: Os modelos são ordenados de acordo com suas acurácias em ordem decrescente. As 10 melhores redes são selecionadas com base na acurácia média.

7- Teste individual das 10 melhores redes: Cada uma das 10 melhores redes é testada individualmente e seus resultados são impressos, incluindo a acurácia e a matriz de confusão.

8- Escolha do melhor modelo: A melhor rede entre as 10 é selecionada com base na acurácia. A matriz de confusão da melhor rede é impressa.

LINK PARA O GITHUB:
https://github.com/cesartiago/Trabalho-5---Redes-Neurais-Art-e-Tratamento-de-Dados-GIT-

CURIOSIDADES:

Em alguns testes, o índice de acerto foi bem maior do que os dos modelos que foram enviados. No entanto, eram testes em que se executou a “Fase 2” e depois a “Fase 3, e que por assim ser (devido ao fato da fase 2 demorar muito), eu colocou-se valores pequenos de loops, como 3 loops para cada hiperparâmetros por exemplo. Em uma dessas vezes, se registrou o valor final obtido, como se pode ver abaixo - mas não modelo, pois não se sabia que não se atingiria essa valor mais tão fácil: