FACULDADE DE TECNOLOGIA DO ESTADO DE SÃO PAULO
CURSO SUPERIOR TECNOLÓGICO DE CIÊNCIA DE DADOS

KAUÃ SANTANA DA SILVA

CLASSIFICAÇÃO DE AUTENTICIDADE DE CÉDULAS BANCÁRIAS COM UM MODELO DE REDE NEURAL ARTIFICIAL

SANTOS 2024

Sumário

1 Os Dados

Para o treinamento do modelo, foi utilizado o conjunto de dados Banknote Authentication, de autoria de Volker Lohweg (2012), um conjunto de dados reais, utilizado no procedimento de avaliação de autenticidade de cédulas bancárias. Os dados do conjunto foram extraídos da digitalização de imagens em 400x400 pixels, utilizando-se da Transformada de Wavelet para a extração de dados estatísticos. O conjunto está disponível no repositório UCI Machine Learning Repository (2013). Não foram encontradas fontes sobre a origem das notas estudadas.

O conjunto apresenta 1372 registros e 4 dimensões, sendo composto por três variáveis contínuas e uma qualitativa — representando a autenticidade da nota. A descrição do conjunto, segundo UCI Machine Learning Repository, pode ser visualizada na tabela 1 abaixo

Tabela 1: Dicionário de dados do conjunto

Variável	**Atributo**	**Tipo**	**Descrição**
variance	Feature	contínua	variância da imagem pós Transformada de Wavelet
skewness	Feature	contínua	assimetria da imagem pós Transformada de Wavelet
kurtosis	Feature	contínua	curtose da imagem pós Transformada de Wavelet
entropy	Feature	contínua	entropia da imagem
class	Target	inteiro	Autenticidade da nota observada: 0 - Falsa; 1 - Verdadeira

Fonte: Adaptado de UCI Machine Learning Repository, 2013.

2 Modelo

Para a realização da classificação, foi utilizada uma rede neural artificial densa *Multi-layer Perceptron* (MLP), por meio da linguagem de programação Python e, por questões de aprendizado, escolhida a construção de uma rede do "zero", sem o uso de bibliotecas e APIs, tais como TensorFlow, Keras e Pytorch, para a modelagem da rede.

2.1 Arquitetura da Rede

A rede contou com a arquitetura padrão MLP, com uma camada de entrada, uma camada de saída e uma série de camadas ocultas. Tanto o número de saídas quanto o tamanho da série oculta foram generalizados de forma dinâmica, para serem estabelecidos no instante do treinamento, porém, padrões foram estabelecidos para tais parâmetros, sendo: 10 camadas ocultas e 2 neurônios de saída.

Para a inicialização dos pesos e dos *bias* da rede, foi utilizado o método *Xavier Normal Initialization*, inicializando os pesos aleatóriamente a partir de uma distribuição normal e os *bias* como zero, o que ajuda a manter a variância dos sinais ao longo das camadas (Stanford, 2024).

2.2 Feedforward: Ativação e Perda

Para o processo de sinapse da rede, a comunicação entre as camadas, informação foi vetorizada por meio da equação da reta, dada por:

$$z = wx + b \tag{1}$$

Onde z é o vetor resultante, x a informação de entrada, w o peso respectivo à conexão entre os neurônios da camada de entrada e da camada escondida e b o *bias* inicializado para a conexão.

Na primeira sinapse da rede, entre a camada de entrada e de saída, foi utilizada a Tangente Hiperbólica como função de ativação entre a camada de entrada e as camadas escondidas. Essa função "ativa a informação' resultante da sinapse, separando os dados entre -1 e 1:

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2}$$

Já na segunda sinapse, da camada escondida com a camada de saída, utilizou-se a função *softmax* na camada de saída, cujo objetivo é transformar os vetores calculados pela rede em uma distribuição de probabilidade. Isso permite que a rede forneça uma "resposta final", em termos de probabilidade, indicando a confiança atribuída a cada classe:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3}$$

Para o cálculo da perda, foi escolhido a função *Categorical Cross-Entropy* (CCE).

Essa função basicamente nos diz o quanto o modelo errou na predição da classe, com base em sua probabilidade. Ela avalia o quanto a probabilidade prevista pelo modelo se afasta do valor verdadeiro para cada amostra, e retorna uma média logarítmica ponderada desse erro sobre todo o conjunto. Seja $y \in \{0, 1\}$ o rótulo real da i-ésima amostra, e $\hat{y}$ o rótulo previsto pelo modelo para essa amostra, a função de perda BCE é dada por:

$$CCE = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log(\hat{y}_{ik}) \tag{4}$$

Aqui, N é o total de dados, K representa o total de classes e a somatória dupla é para avaliar o desempenho do modelo para todas as classes, percorrendo todos os dados. Assim, a função compara a distribuição de probabilidade prevista pela rede com a distribuição real dos dados, retornando o erro entre elas.

2.3 Backpropagation: Aprendizado e Otimização

Após o processo de sinapse, e rede inicia o processo de aprendizado, no qual ela refaz todo processo de feedforward, porém inversamente, meio que "andando para trás". A rede retorna em seus cálculos, entendendo onde errou e ajustando os pesos e os *bias* na direção da menor perda. Em outras palavras, ela otimiza o aprendizado, minimizando a função de erro.

Este processo ocorre por meio da descida do gradiente da função de erro. De forma simples, os gradientes de uma função nos dá um vetor que aponta para a direção de sua maior variação. Aplicando isso à CCE, função de erro da rede, teremos a direção que causa maior erro. Logo, basta calcular os gradientes da CCE em relação às variáveis de aprendizado da rede, os pesos e os *bias*, atualizando-os no sentido de diminuir o erro. A ideia é simples, porém, a CCE não está diretamente ligada a estas variáveis, já que ela é aplicada apenas ao final da rede. A solução, então, é entender por onde a CCE se liga às variáveis, para entender como elas influenciam sua variação. Seja a CCE e a saída da softmax, na Figura 1 abaixo, visualiza-se a conexão da CCE com as demais funções e variáveis da rede.

Figura 1: Conexão funcional da rede da saída para a entrada

$$\mathscr{L} \rightarrow \hat{y} \rightarrow z_2 \rightarrow w_2 \rightarrow b_2 \rightarrow \alpha \rightarrow tanh \rightarrow z_1 \rightarrow w_1 \rightarrow b_1 \rightarrow x$$

Fonte: Elaborado pelo autor, 2025.

Com a Figura 1, podemos entender o caminho a ser percorrido pela rede no *back-propagation*, a fim de ajustar as variáveis de aprendizado.

Para isso, o Cálculo nos fornece uma ferramenta poderosa para lidar com este problema: a regra da cadeia. No fundo, pode-se dizer que uma rede neural é uma "enorme regra da cadeia", e seu aprendizado se dá quase que puramente por sua aplicação. Este método nos permite entender a variação de uma função aplicada à outra, ou outras.

Neste caso, a regra da cadeia se dará pelas derivadas parciais da função à esquerda em relação à da direita, seguindo assim, respectivamente, até a entrada, onde as variáveis de aprendizado são inicializadas.

Após os cálculos, basta atualizar as variáveis pelos resultados obtidos, para que os erros de aprendizados sejam corrigidos. Este é um passo simples, porém de extrema importância para o aprendizado da rede. Comparando com aprendizado humano, se as variáveis não forem atualizadas, seria como se nós aprendêssemos, porém nunca nos lembrássemos de nada do que aprendemos.

- Erro em relação ao peso de saída

$$\frac{\partial \mathscr{L}}{\partial w_2} = \frac{\partial \mathscr{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}$$

- Erro em relação ao *bias* de entrada

$$\frac{\partial \mathscr{L}}{\partial b_2} = \delta_2$$

- Erro em relação ao peso de entrada

$$\frac{\partial \mathscr{L}}{\partial w_1} = \frac{\partial \mathscr{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial \alpha_1} \cdot \frac{\partial \alpha_1}{\partial z_1}$$

- Erro em relação ao *bias* de entrada

$$\frac{\partial \mathscr{L}}{\partial b_1} = \delta_1$$

Uma breve, porém importante, observação sobre as derivadas parciais acima: durante os cálculos, o termo $\frac{\partial \mathscr{L}}{\partial z_i}$, a derivada parcial de $\mathscr{L}$ em relação a uma sinapse, seja da camada de entrada ou da camada de saída, torna a aparece repetidas vezes e, por isso, é comum substituí-lo por δ, como feito acima. Este termo representa o erro local de um neurônio durante a sinapse. Nos casos em que se utilizam as funções *softmax* e *tangente hiperbólica* juntas, ele se dá pela diferença entre a previsão e o rótulo real: $\hat{y} - y$.

Por fim, com as variáveis atualizadas, basta repetir o processo até que o erro seja mínimizado. Para isso, utiliza-se as *epochs*, as vezes em que a rede revisitará os dados, ajustando as variáveis e reparando seus erros. O número de *epochs* é escolhido durante

o processo de treinamento e deve ser escolhido com cautela. Caso o número de epochs seja pequeno, a rede não se ajustará o suficiente aos dados e não aprenderá seus padrões, causando o chamado *underfitting*. Caso o número de epochs seja grande, a rede se ajustará demais aos dados, deixando de aprender e passando a memorizar os padrões dos dados, causando o chamado *overfitting*. Ambos os casos ocasionará em uma má performance da rede.

3 Aplicação

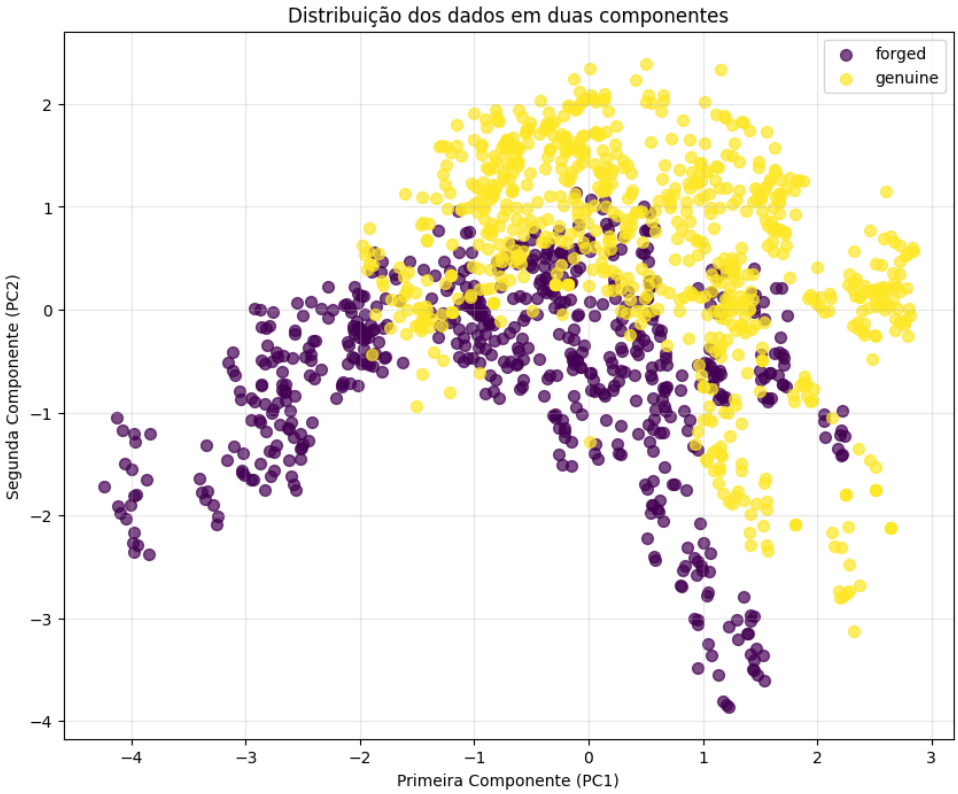
Com a obtenção do conjunto em formato CSV, foi necessário fazer alguns ajustes de pré-processamento, antes da aplicação do modelo.

A fim de uma melhor interpretação dos resultados, foi realizada a alteração dos valores das classes, sendo substituídos de 0 e 1 para genuine e forged, representando observações verdadeiras e falsas, respectivamente. Além disso, para viabilizar a aplicação do modelo aos dados, foi feita a separação dos dados em Feature e Target, para entrada e saída do modelo. A variável class foi separada do restante dos dados, num conjunto Y. Todas as variáveis restantes foram separadas no conjunto X.

3.1 PCA

Considerando a exigência de visualização dos resultados, ainda no pré-processamento, foi feita a aplicação do *Principal Component Analysis* (PCA), sendo reduzida a dimensionalidade dos dados para duas componentes. Para sua aplicação, foi utilizado o método *Standard Scaler*, da biblioteca *Scikit-learn*, para a normalização dos dados. A distribuição dos dados resultante do PCA pode ser observada na Figura 2. Com o propósito de comparação da performance do modelo, os dados originais foram preservados.

Figura 2: Distribuição dos dados em duas Componentes Principais



Fonte: Adaptado de Lohweg, 2012.

Visualizando a distribuição dos dados em duas dimensões, nota-se uma nuvem de confusão sobrepondo os dados uns nos outros, entre os valores de -2 a 2, na primeira componente, e entre os valores de 1 a -2, na segunda componente. Com isso, levanta-se a hipótese de que o modelo não saberá lidar com a sobreposição dos dados de classes diferentes, não performando tão bem em relação a estes dados.

4 Resultados

Feito a aplicação do modelo em ambos datasets, com e sem PCA, pode-se comparar os resultados e entender como o modelo performou com o mesmo problema, em dimensões diferentes.

Utilizando 10 camadas escondidas, 2 camadas de saída, 20 épocas de treinamento e um *learning rate* de 0.001, chegou-se aos resultados mostrados na Figura 3 abaixo.

Figura 3: Acurácia e Perda no treinamento sem PCA

```
Epoch: [2 / 20] Accuracy: 0.882 Loss: 0.000132410497844558534266332094908591443527257069945335388183593750000000000000
Epoch: [4 / 20] Accuracy: 0.819 Loss: 0.000125504505525128698699074969447053717885864898562431335449218750000000000000
Epoch: [6 / 20] Accuracy: 0.954 Loss: 0.000127561078281032420477653088752845178532879799604415893554687500000000000000
Epoch: [8 / 20] Accuracy: 0.981 Loss: 0.000055997306691250757157158474131009029406413901597261428833007812500000000000
Epoch: [10 / 20] Accuracy: 0.988 Loss: 0.000034599537411300951609237352979420165866031311452388763427734375000000000000
Epoch: [12 / 20] Accuracy: 0.988 Loss: 0.000023650489771197748554770398654589769193989923223853111267089843750000000000
Epoch: [14 / 20] Accuracy: 0.992 Loss: 0.000018194050670757316059663113994027128228481160476803779602050781250000000000
Epoch: [16 / 20] Accuracy: 0.994 Loss: 0.000014930889637756275962468185958265110002685105428099632263183593750000000000
Epoch: [18 / 20] Accuracy: 0.996 Loss: 0.000012695806415715489299579112059124241795871057547628879547119140625000000000
Epoch: [20 / 20] Accuracy: 0.999 Loss: 0.000011020254424201851364546571321323398251479375176131725311279296875000000000
```

Fonte: Elaborado pelo autor, 2025.

Utilizando 200 camadas escondidas, 2 camadas de saída, 300 épocas de treinamento e um learning rate de 0.001, chegou-se aos resultados mostrados na Imagem 4 abaixo.

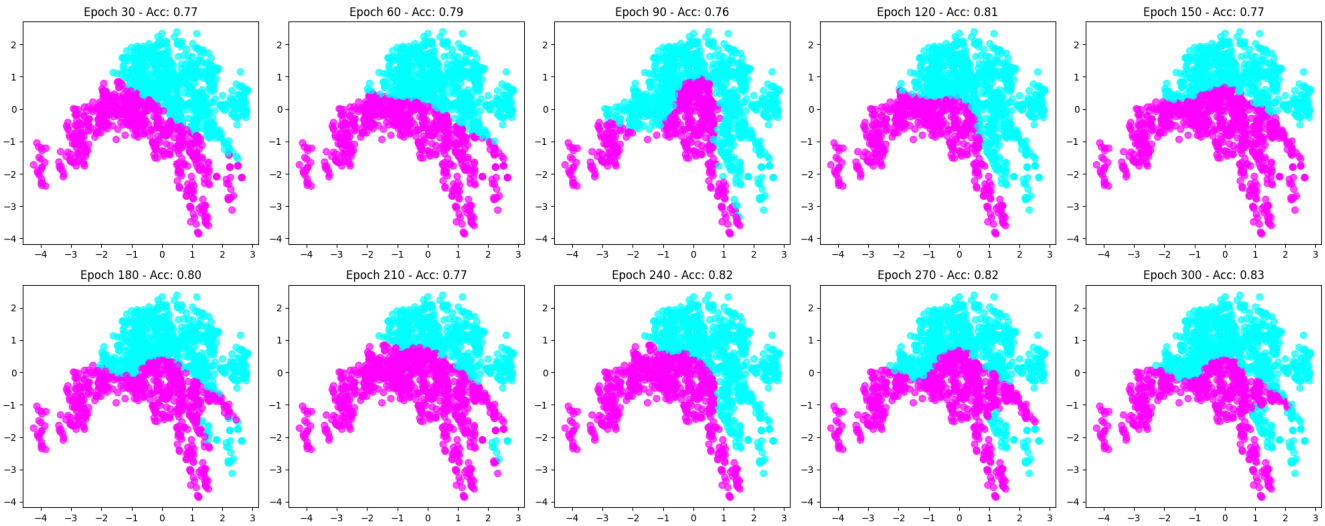
Figura 4: Acurácia e Perda no treinamento com PCA

```
Epoch: [30 / 300] Accuracy: 0.771 Loss: 0.000000000000012973425379124469227056952849111882057096876812163355197071723523555
Epoch: [60 / 300] Accuracy: 0.789 Loss: 0.000004843578645820401410663352814722770744992885738611221313476562500000000000000
Epoch: [90 / 300] Accuracy: 0.759 Loss: 0.000850223553316481169342400114885549555765464901924133300781250000000000000000000
Epoch: [120 / 300] Accuracy: 0.811 Loss: 0.000126120865732372500938110460033669824042590335011482238769531250000000000000000
Epoch: [150 / 300] Accuracy: 0.772 Loss: 0.000112516770845788463897840792071747273439541459083557128906250000000000000000000
Epoch: [180 / 300] Accuracy: 0.802 Loss: 0.000487490425099027196824408303399422948132269084453582763671875000000000000000000
Epoch: [210 / 300] Accuracy: 0.769 Loss: 0.000022005916066862239011516291786740850966452853754162788391113281250000000000000
Epoch: [240 / 300] Accuracy: 0.816 Loss: 0.000104757097994695555932126440623619600955862551927566528320312500000000000000000
Epoch: [270 / 300] Accuracy: 0.822 Loss: 0.000614298834370813643876441112468000937951728701591491699218750000000000000000000
Epoch: [300 / 300] Accuracy: 0.826 Loss: 0.000733515634135856035326350710334963878267444670200347900390625000000000000000000
```

Fonte: Elaborado pelo autor, 2025.

Com os resultados das Figuras 3 e 4, nota-se que o modelo realmente performou melhor com os dados nas 4 dimensões originais que nas 2 dimensões reduzidas com o PCA, como levantado na hipótese anteriormente. Para entender melhor como ele lidou com os dados sobrepostos, visualiza-se abaixo, na Figura 5, os resultados gráficos da classificação dos dados com PCA.

Figura 5: Resultados gráficos do processo de classificação em duas Componentes Principais



Fonte: Adaptado de Lohweg, 2012.

Nota-se, portanto, a confusão do modelo ao classificar os dados na nuvem de confusão, o que explica a acurácia consideravelmente menor para estes dados, indo de 77% para aproximadamente 83%, com uma perda de 10e-3 (após a terceira casa decimal); enquanto que, para os dados originais, o modelo foi de 88% para 99%, alcançando uma perdas 10e-4 (após a quarta casa decimal).

Referências

NG, Andrew; KATANFOROOSH, Kian. *Section 4 (Week4). Xavier Initialization and Regularization*. Stanford University, 2024. Disponível em: `https://cs230.stanford.edu/section/4/`. Acesso em: 30 maio 2025.

UCI MACHINE LEARNING REPOSITORY. Banknote Authentication Data Set. Disponível em: https://archive.ics.uci.edu/ml/datasets/banknote+authentication. Acesso em: 12 jun. 2025.