Exercício 2 - Deep Feedforward Neural Networks

Caroline Pereira de Sena[1]

[1]Universidade Tecnológica Federal do Paraná

carolinesena@alunos.utfpr.edu.br

1. Exploração e pré-processamento dos dados

O dataset tinha alguns dados faltantes, porém as classes ("True"e "False") presentes na coluna "Transported"são aproximadamente balanceadas, com 4378 e 4315 valores respectivamente. Outro fator importante do dataset é a presença de valores categóricos, que devem ser transformados para um equivalente numérico, para permitir o treinamento da rede. Como há valores faltantes, eles devem ser preenchidos utilizando alguma estratégia. Considerando ainda que as classes estão balanceadas, não é necessário aplicar uma técnica para balanceamento.

A transformação de valores categóricos foi feita manualmente e algumas colunas foram transformadas para se extrair informação extra. As colunas CryoSleep e VIP eram compostas for valores "True"e "False"que foram substituídos por 1 e 0, respectivamente. A coluna HomePlanet continha os valores "Europa", "Earth"e "Mars", que foram substituídos por 1, 2 e 3, respectivamente. O mesmo foi feito para a coluna Destination, que continha os valores "TRAPPIST-1e", "PSO J318.5-22"e "55 Cancri e". A coluna Cabin foi foi utilizada para gerar uma nova coluna chamada CabinType, de forma que para cada valor presente na coluna Cabin (de formato x/y/z), um novo valor foi criado utilizando apenas a primeira e a última parte (xz), que posteriormente também foram substituídos por valores de 1 a 16. A coluna Name foi utilizada para gerar a coluna NumberRelatives, com a contagem do número de pessoas com o mesmo sobrenome no dataset.

Por fim, as colunas Transported, PassengerId, Cabin e Name, foram removidas dos datasets de treino e teste, e o primeiro foi dividido entre conjuntos de treino e teste (10%).

2. Modelo Inicial

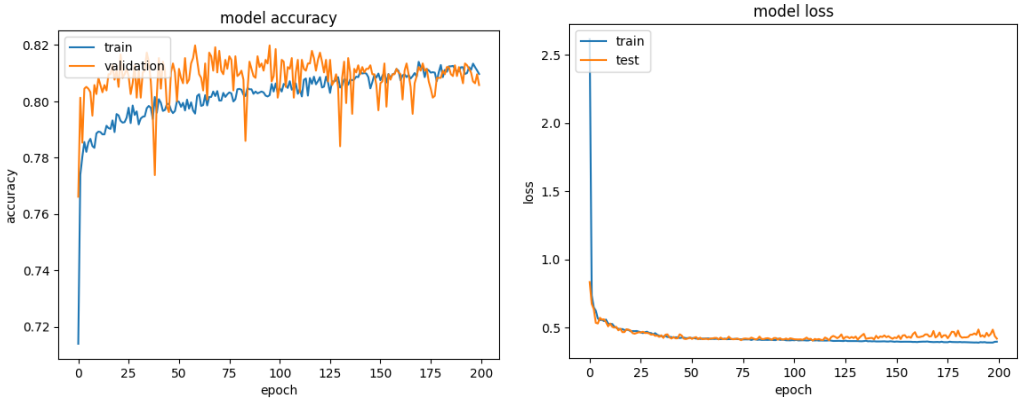
Para o modelo inicial, foram utilizadas 5 camadas densas, com função de ativação ReLU e otimizador Adam, de acordo com o trecho de código abaixo.

```
        num_classes = 2

        model = Sequential()
        model.add(Dense(12, input_shape=(12,), activation='relu'))
        model.add(Dense(32, activation='relu'))
        model.add(Dense(32, activation='relu'))
        model.add(Dense(32, activation='relu'))
        model.add(Dense(num_classes, activation='softmax'))
        model.compile(loss='binary_crossentropy', optimizer='adam',
    metrics=['accuracy'])
        history = model.fit(x_train, y_train, validation_split=0.2,
    shuffle=True, epochs=200, batch_size=40, verbose=2)

```

As métricas e curvas obtidas são mostradas abaixo.

	Accuracy	Loss
Treino	0.8097	0.3965
Validação	0.8058	0.4189
Teste	0.7828	0.4703

Tabela 1. Métricas do modelo inicial



(a) (b)

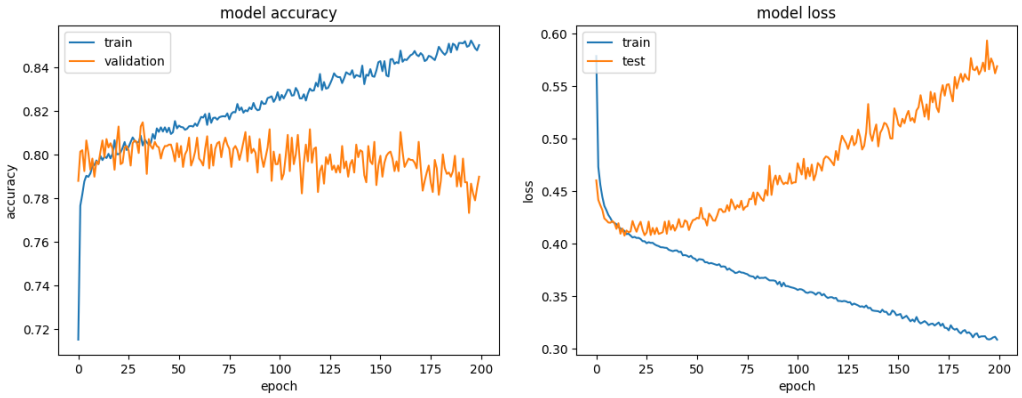
Figura 1. Curvas de treinamento: (a) Acurácia, (b) Loss

3. Treinamento com uso de Scaler

Foi adicionado o Standard Scaler do Scikit Learn, para fazer a normalização dos dados, porém as curvas obtidas demonstram que apesar da melhora na acurácia de treinamento, a validação e teste não tiveram o mesmo desempenho.

	Accuracy	Loss
Treino	0.8501	0.3084
Validação	0.7898	0.5690
Teste	0.7609	0.5724

Tabela 2. Métricas do modelo com Scaler



(a) (b)

Figura 2. Curvas de treinamento: (a) Acurácia, (b) Loss

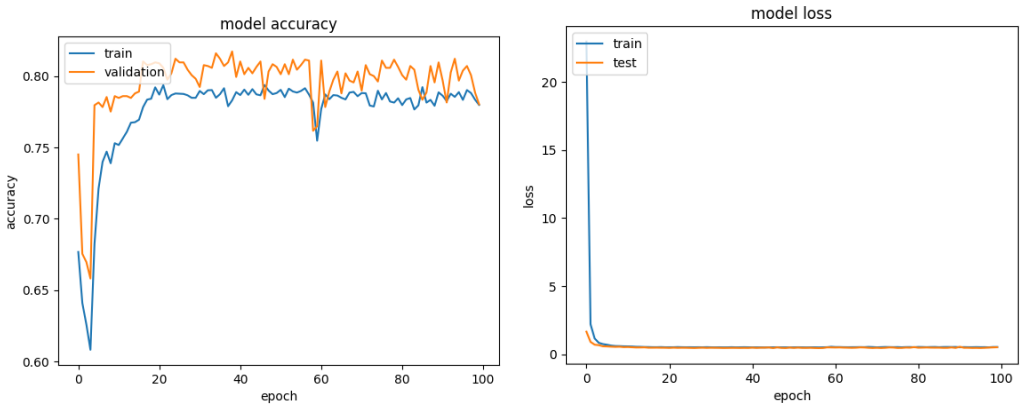
4. Grid Search e modelo final com camadas de *Dropout*

Foram adicionadas camadas de Dropout ao modelo anterior e foi feita também aplicação de Grid Search, que sugeriu a utilização da função de ativação linear, camadas com 16 unidades e otimizador Adam, com taxa de aprendizado de 0.01, conforme é mostrado no trecho abaixo.

```
        num_classes = 2

        model = Sequential()
        model.add(Dense(16, input_shape=(12,), activation='relu'))
        model.add(Dense(16, activation='linear'))
        model.add(Dropout(0.4))
        model.add(Dense(16, activation='linear'))
        model.add(Dropout(0.4))
        model.add(Dense(16, activation='linear'))
        model.add(Dropout(0.4))
        model.add(Dense(num_classes, activation='softmax'))


        adam = keras.optimizers.Adam(learning_rate=0.01)
        model.compile(loss='binary_crossentropy', optimizer=adam,
    metrics=['accuracy'])
        history = model.fit(x_train, y_train, validation_split=0.2,
    shuffle=True, epochs=100, batch_size=40, verbose=2)
```

	Accuracy	Loss
Treino	0.7563	0.5622
Validação	0.7859	0.5080
Teste	0.7621	0.5523

Tabela 3. Métricas do modelo com Scaler



(a) (b)

Figura 3. Curvas de treinamento: (a) Acurácia, (b) Loss

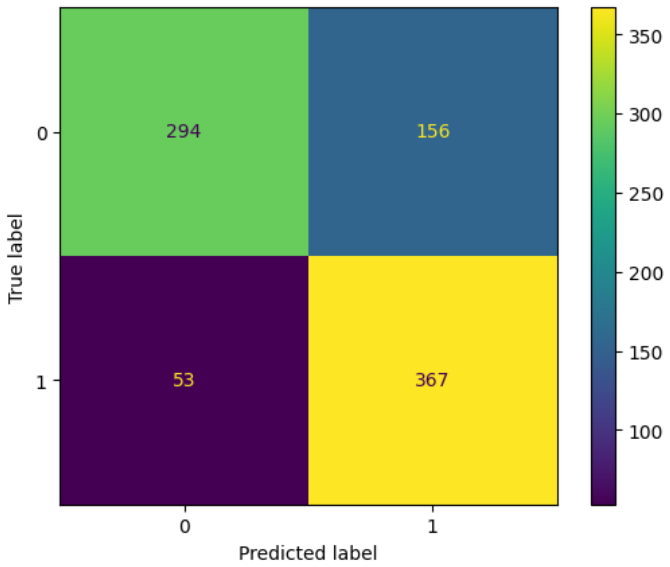


Figura 4. Confusion Matrix