Relatório do Projeto 1

Projeto desenvolvido durante no 2º período do 2º semestre do ano letivo 2022/2023

1st Leonardo Amado Brito
Instituto Superior Técnico
Mestrado em Engenharia e Ciência de Dados
Lisboa, Portugal
leonardo.amado.brito@tecnico.ulisboa.pt

2nd Gustavo Caria
Instituto Superior Técnico
Mestrado em Matemática Aplicada e Computação
Lisboa, Portugal
gustavocaria@tecnico.ulisboa.pt

I. Introdução

Os *Wireless Sensor Networks* (WSN) são sistemas complexos formados por nós de sensores interconetados. Cada nó é multifacetado, geralmente composto por um transcetor de rádio com uma antena interna ou externa, um microcontrolador, um circuito eletrónico para interface com os sensores e uma fonte de energia.

Um aspeto chave no funcionamento dos WSN é a localização dos nós dos sensores. A localização pode ser alcançada usando módulos GPS, mas esta abordagem acarreta um custo significativo e consumo de energia. Portanto, uma alternativa prevalente envolve o uso de alguns nós equipados com GPS para inferir as posições dos nós restantes com base em informações de conectividade.

Os algoritmos de localização costumam ser computacionalmente caros. Este compromisso limita a viabilidade de realizar várias simulações para otimização de parâmetros, incluindo decisões sobre o número de sensores, número de nós de ancoragem ou alcance de transmissão necessário para garantir um erro de localização aceitável.

Em resposta a este desafio, foi proposta uma abordagem para agilizar este processo. Através do uso de uma Rede Neuronal (NN), o nosso objetivo foi prever o Erro de Localização do Sensor (ESLE) com base em um determinado conjunto de parâmetros. Utilizámos um conjunto de dados que inclui variáveis como Densidade de Nós, Rácio de Ancoragem, Alcance de Transmissão, *Step Size*, Iterações e o ESLE resultante.

O objetivo deste projeto foi desenvolver um modelo de rede neural capaz de estimar com precisão o ESLE, dado os parâmetros presentes em nosso conjunto de dados. Tivemos em conta que o conjunto de dados pode conter teve que ser limpo e analisado antes de ser usado.

Detalhamos, neste relatório, as nossas decisões em relação à preparação de dados, configuração experimental, construção do modelo, processo de avaliação e validação. Também falamos do ficheiro *TestMe*, que produz um vetor de previsão de ELSE para cada entrada no arquivo de teste, bem como o Erro Quadrático Médio da Raiz (RMSE) da previsão de ESLE que servirá para a avaliação do modelo produzido.

Ao longo deste projeto, nosso foco não foi apenas entregar um modelo ótimo, mas também fornecer um relato das experiências e daquilo que pensámos e decidimos detalhadamente.

II. Análise Estatística e Pré-Processamento de Dados

A análise estatística e pré-processamento de dados são etapas fundamentais no processo de análise e modelação de dados. Têm o objetivo de obter uma análise inteligente dos dados que nos permitirá prepará-los melhor para serem utilizados pelo modelo. Está composta em:

- Limpeza de dados;
- Deteção e tratamento de *outliers*;
- Redução de dimensionalidade;
- Normalização;

A codificação de variáveis categóricas não foi necessária porque não as tínhamos no *dataset*.

A. Sumário estatístico das variáveis

Inicialmente quisemos conhecer os nossos dados. O objetivo passou por ter uma análise estatística básica. Também tivemos o cuidado de abrir o *dataset* e olhar para todos os dados que tínhamos, visto que o *dataset* era curto. Através das tabelas abaixo tivemos uma perceção do tipo de valores, valores mínimos e máximos, a média, etc. Este tipo de análise foi essencial para perceber melhor as condições que levam que observação pode vir a ser um *outlier*.

TABLE I
Sumário estatístico (1)

Variável	Média	Desv. Padrão	Min	Max
Anchor Ratio	0.207	0.065	0.140	0.300
Transmission Range	20.555	3.410	17.250	28.750
Node Density	0.015	0.007	0.010	0.030
Step Size	1.033	0.125	0.900	1.200
Iterations	37.537	24.105	6.000	90.000
ESLE	1.220	1.536	0.000089	25.000

TABLE II
SUMÁRIO ESTATÍSTICO (2)

Variável	25%	75%
Anchor Ratio	0.150	0.300
Transmission Range	17.250	23.000
Node Density	0.010	0.020
Step Size	0.900	1.200
Iterations	20.000	60.000
ESLE	0.783	1.361

B. Boxplots

Os *boxplots* são úteis para identificar a dispersão dos dados, verificar a presença de assimetrias ou desvios da distribuição normal, comparar visualmente a distribuição de diferentes grupos ou categorias de dados e a forma da distribuição dos dados. Logo, são ótimos para perceber se há dados fora da norma.

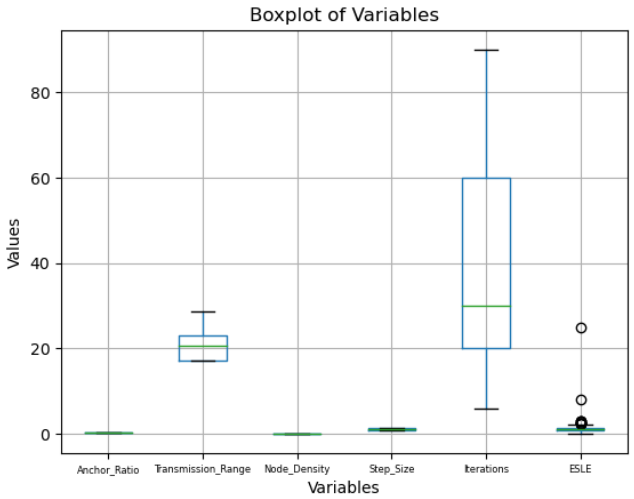

Fig. 1. *Boxplot* geral do dataset

Imediatamente, tirámos duas conclusões:

- a variável de interesse *ESLE* parece ter *outliers*
- há grandes indicações de que teremos de fazer normalização dos dados, tendo em conta a disparidade de escalas das variáveis, e.g. escala de *Iterations* é $[6, 90]$ e escala de *Anchor_Ratio* é $[0.14, 0.3]$

C. Remoção dos outliers

Foi encontrada uma ligação entre os *outliers* e o valor dos mesmos nos *Iterations*. (número de iterações no algoritmo utilizado para estimar a localização do sensor)

O número de iterações no algoritmo indica quantas vezes o processo iterativo é repetido. Cada iteração permite que o algoritmo faça ajustes e melhore a precisão das posições estimadas dos sensores. Geralmente, aumentar o número de iterações pode resultar em uma maior precisão de localização, uma vez que o algoritmo tem mais oportunidades para aprimorar as estimativas, mas também tem um custo computacional associado.

D. Histogramas

Depois, observámos os histogramas e gráficos de densidade de cada variável:

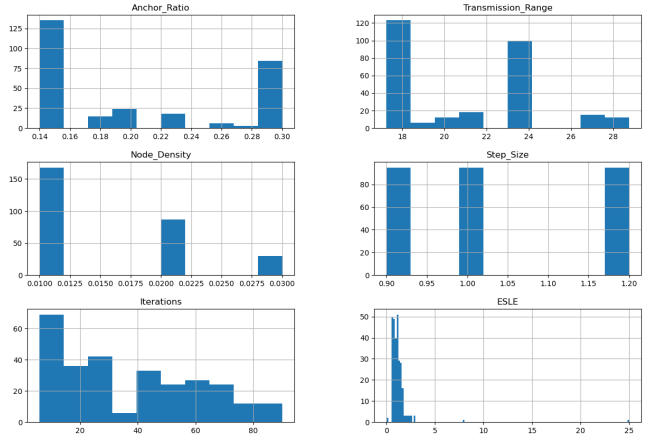

Fig. 2. histogramas das variáveis

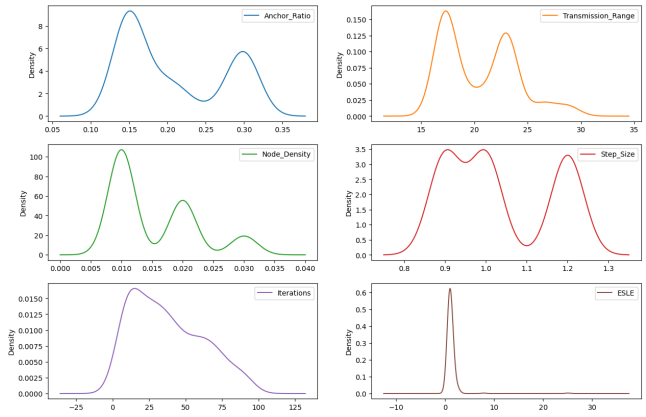

Fig. 3. gráficos de densidade de cada variável

Dos histogramas e dos gráficos de densidade tirámos a conclusão de que provavelmente conseguiríamos ajustar as variáveis *Iterations* e *ESLE* a alguma distribuição conhecida. As restantes variáveis não parecem promissoras nesse aspeto, mas mesmo assim tentámos avaliá-las também.

De facto, usámos o *package Fitter* para comparar as variáveis a mais de 100 variáveis conhecidas, e os únicos resultados promissores foram os expectáveis:

Primeiro apresentamos as duas variáveis que esperávamos conseguir aproximar (face ao formato dos seus histogramas).

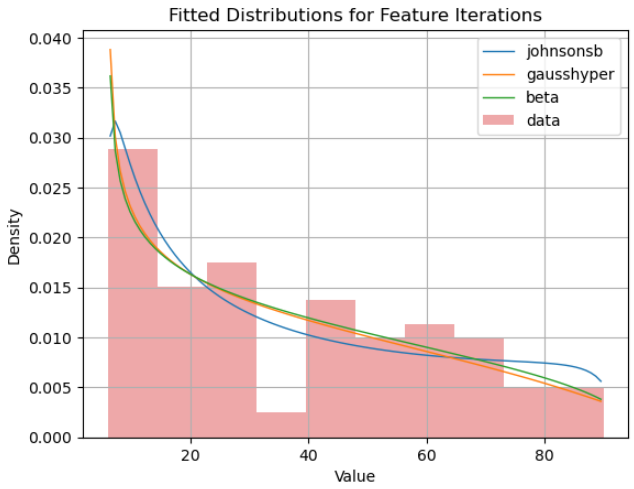

Fig. 4. As 3 distribuições mais ajustadas (na legenda, estão ordenadas da mais ajustada à menos ajustada) ao histograma de densidade da variável *Iterations*

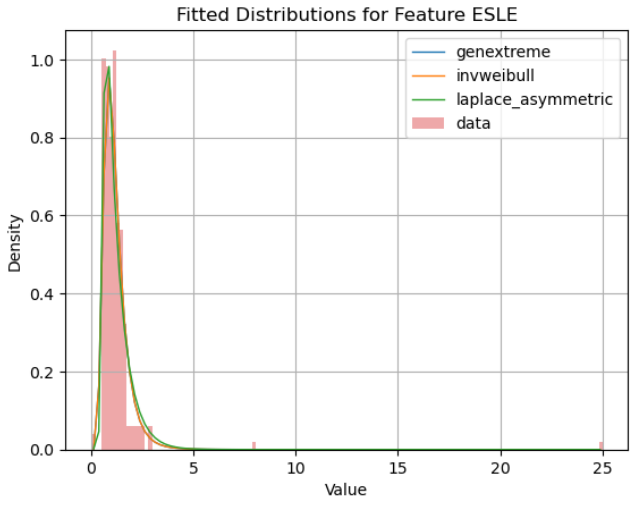

Fig. 5. As 3 distribuições mais ajustadas (na legenda, estão ordenadas da mais ajustada à menos ajustada) ao histograma de densidade da variável *ESLE*

As restantes variáveis não conseguiram ser bem aproximadas por distribuições conhecidas. Assim sendo, apresentamos apenas os resultados de uma variável como exemplo ilustrativo:

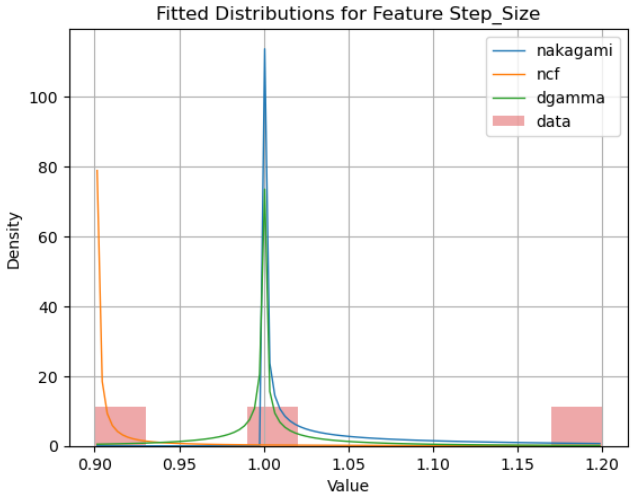

Fig. 6. As 3 distribuições mais ajustadas (na legenda, estão ordenadas da mais ajustada à menos ajustada) ao histograma de densidade da variável *Step_Size*

Como se pode ver, claramente nenhuma das distribuições encontradas se ajusta bem à variável *Step_Size*. De qualquer das formas, face ao aspeto do histograma da variável, seria de esperar o caso.

Portanto, decidimos ajustar **apenas** a variável *Iterations* à distribuição *jhonsonsb* e a variável *ESLE* à distribuição *genextreme*. E apresentamos o resultado das transformações aplicadas às duas variáveis:

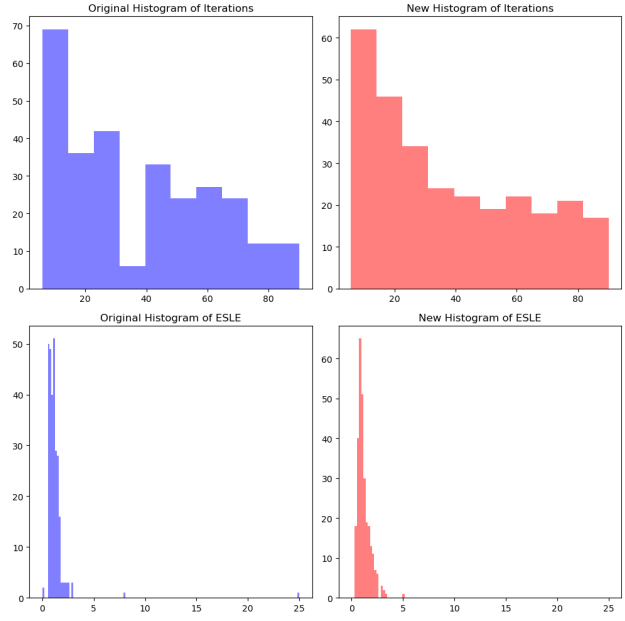

Fig. 7. A azul, as variáveis *Iterations* (acima) e *ESLE* (abaixo) originais e, a vermelho, o resultado do seu ajuste às distribuições mais próximas.

A forma como geramos estes dados foi simples:

1) encontrámos a melhor distribuição que se ajustasse às variáveis (e os seus parâmetros)
2) gerámos 285 (número de observações total) observações novas para as calculadas distribuições com os parâmetros que encontrámos
3) como estas observações foram geradas aleatoriamente, a sua ordem original foi perdida, pelo que tivemos que as reordenar de acordo com os dados originais
4) para verificar se a reordenação foi bem feita, visualizámos o *scatterplot* da variável original *vs* a variável ajustada à distribuição (o gráfico segue-se abaixo)

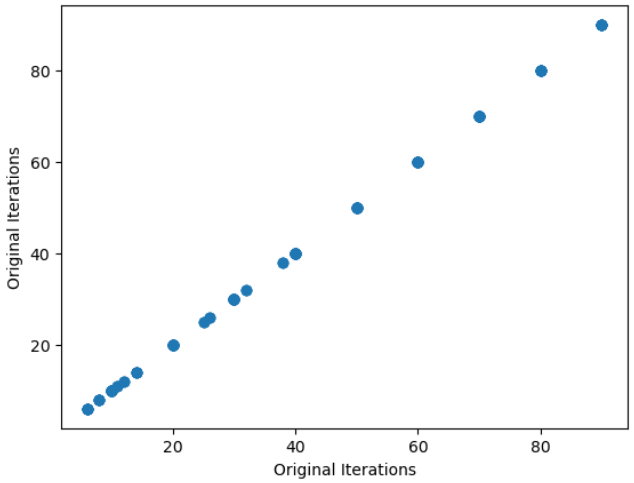

Fig. 8. *scatterplot* da variável *Iterations* contra si mesma, resultando na esperada linha reta

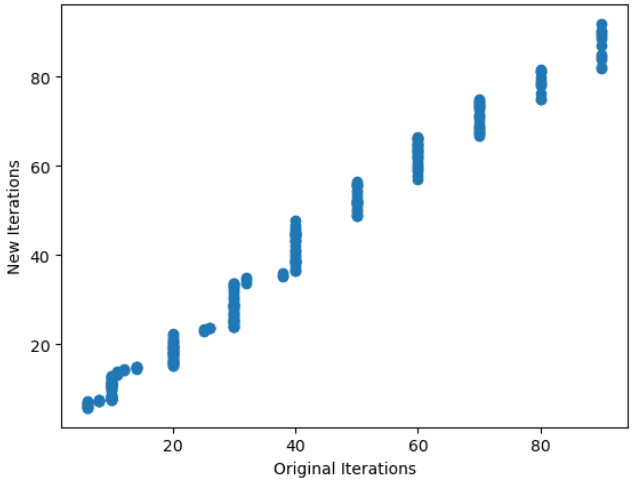

Fig. 9. *scatterplot* da variável *Iterations* contra a sua versão ajustada **e alinhada**, resultando em próximo de uma linha reta, mostrando que os dados foram alinhados (nunca poderia ficar realmente uma linha reta, visto que a variável foi ajustada e portanto os dados não são os mesmos

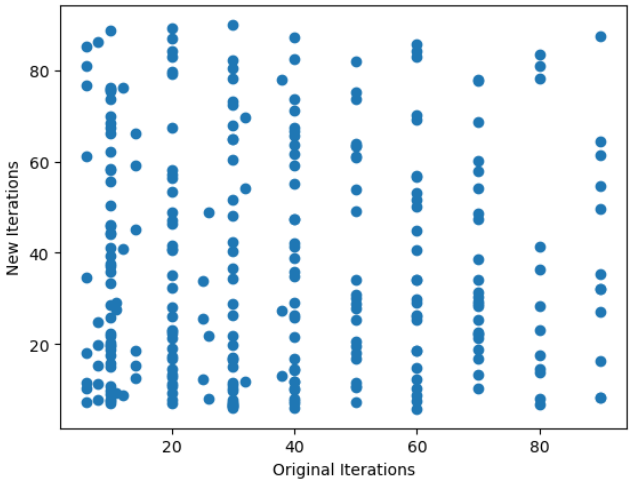

Fig. 10. *scatterplot* da variável *Iterations* contra a sua versão ajustada mas **ainda não ordenada**, resultando numa completa aleatoriedade, visto que aqui os resultados ainda não tinham sido ordenados

E. Data Cleaning:

Em termos de *data cleaning*, não encontrámos nada do suposto, i.e. dados incompletos, inconsistentes ou intencionais, pelo que não pudemos aplicar nada neste sentido.

A nossa maior dúvida era se haviam dados com ruído, pelo que fizemos uma análise para perceber se haviam *outliers*, e também verificámos se os nossos dados se ajustavam a alguma distribuição conhecida, e foi o caso da variável de interesse *ESLE*.

F. Transformação e preparação dos dados para o modelo

Os dados já devidamente tratados foram divididos, inicialmente, em dois: os dados de *input* e os dados de *output*. Fixámos a *seed* (para que a divisão dos dados que façamos seja sempre igual), e dividimos os dados de input e de output em dois (cada um), ou seja, em dados de treino e de teste. A utilização de dados de treino e teste separados permite avaliar a capacidade do modelo de generalizar para dados não vistos anteriormente, fornecendo uma estimativa da performance do modelo. Usámos 80% para treino e 20% para teste. No final, normalizámos os dados entre 0 e 1 por causa da função de ativação (regressão logística), cujos valores variam também entre 0 e 1. Uma breve nota para o facto de termos tido o cuidado de usar os mesmos valores usados no valor mínimo e máximo para a normalização de dados do treino nas outras normalizações de dados, tal como no teste e no ficheiro *TestMe*. A normalização ajuda a garantir que os dados de entrada usados para previsão estejam em uma escala semelhante aos dados usados durante o treino. Ao aplicar a mesma técnica de normalização que foi utilizada nos dados de treino, é possível manter a consistência nos passos de pré-processamento dos dados.

Ao normalizar o conjunto de dados para previsão, é importante utilizar os mesmos parâmetros de normalização que foram calculados a partir dos dados de treino. Isso garante que

o processo de normalização seja consistente e que os dados sejam transformados da mesma maneira que durante o treino. Aplicar a técnica de normalização aprendida a partir dos dados de treino ajuda a garantir que as previsões sejam feitas em dados comparáveis ao que o modelo foi treinado. Claro que o processo de normalização foi unicamente aplicado aos dados de *input*.

III. Modelos

A. Multilayer Perceptron

O código cria um objeto do modelo uma Multilayer Perceptron (MLP) para regressão. O melhor modelo para os dados não ajustados tem uma única *hidden layer* com 10 neurónios. O parâmetro *alpha* é um método para prevenir o *overfitting* ao adicionar uma penalização à função de perda e foi definido com 0.01. A regularização L1 ou L2 pode ser adicionada através do parâmetro *alpha*. A taxa de *learning rate* é definida como 0.1. A função de ativação utilizada é a função logística.

O algoritmo de otimização utilizado é o *Stochastic Gradient Descent* (SGD), apesar de ter sido testado com o *Adam*. A opção *early_stopping* está ativada, o que significa que o treino será interrompido se não houver melhoria no desempenho nos dados de validação. O número máximo de iterações (épocas) permitidas durante o treino é definido como 50.

Uma fração de 20% dos dados de treino é usada como conjunto de validação para acompanhar o desempenho. A taxa de aprendizagem é ajustada de forma adaptativa durante o treino. O tamanho do lote (batch size) é definido como 128, o que significa que 128 amostras são usadas para calcular o gradiente e atualizar os pesos a cada passo de treino.

O treino é interrompido se não houver melhoria no desempenho nos dados de validação por 50 iterações consecutivas. A tolerância definida é 1e-4, o que significa que se a melhoria no desempenho for menor do que essa tolerância, o treino será interrompido.

Durante a fase de treino, foram feitas mudanças no *batch size* entre 32 e 512 para valores 2^n. Para os restantes hiperparâmetros utilizámos o *GridSearchCV* para ajustar os hiperparâmetros, tais como o *learning rate*, o número de camadas e número de neurónios. O *GridSearch* consiste em especificar uma lista de valores possíveis para cada hiperparâmetro do modelo e realizar uma busca exaustiva por todas as combinações possíveis desses valores. Para cada combinação de hiperparâmetros, o modelo é treinado e avaliado usando validação cruzada ou alguma outra métrica de avaliação. O objetivo é identificar a combinação de hiperparâmetros que resulta no melhor desempenho do modelo.

Foram testado os seguintes hiperparâmetros:

- *Hidden layer*: (10,), (10,5), (10,10), (50,), (100,), (100, 5), (100, 10), (50, 50), (100, 5);
- *Learning rate*: 0.0001, 0.001, 0.005, 0.01, 0.1;
- *Alpha*: 0.0001, 0.001, 0.01;

B. Convolucional Neural Network

Testámos uma *Convolucional Neural Network* (CNN) para uma tarefa de regressão. Primeiro, os dados de treino e teste são redimensionados para terem uma forma tridimensional, que é requerida pela CNN. Em seguida, o modelo é construído, utilizando uma camada Conv1D com ativação sigmoide, uma camada *flatten* e uma camada densa de saída. O modelo é compilado com o otimizador SGD, utilizando a função de perda *mean squared error* (MSE) e a mesma métrica para avaliação.

O treino do modelo foi realizado em um *loop* com 100 iterações. A cada iteração, o modelo é ajustado aos dados de treino, utilizando validação cruzada com uma fração de 20% para monitorizar o desempenho. É utilizado o *callback EarlyStopping* para interromper o treino caso não ocorra melhora na perda de validação após 4 épocas.

Após o treino, o modelo é utilizado para fazer previsões nos dados de teste. O MSE é calculado entre as previsões e os valores reais, fornecendo uma medida de desempenho do modelo. O processo de treino e avaliação é repetido 100 vezes, armazenando o MSE de cada iteração em uma lista. Ao final, é calculada a média dos valores de MSE, fornecendo o MSE médio nos dados de teste. Foi realizado um *loop* de 100 iterações para que o valor final possa convergir. Os valores finais foram piores que os da MLP, como demonstramos na tabela 3, o que era esperado pelo facto de uma CNN ser um modelo mais apropriado para imagens.

TABLE III
Comparison of RMSE and R² results for MLP vs. CNN Models using different datasets

	Dados sem ajuste		Dados com ajuste	
Modelo	RMSE	R²	RMSE	R²
MLP	0.22	0.60	0.34	0.57
CNN	0.30	0.38	0.46	0.38

Estes foram os melhores resultados para os dados sem ajuste às distribuições e para os dados com ajuste às distribuições, com os melhores. De qualquer das formas, o melhor modelo foi o sem ajuste, MLP.

IV. TestMe

O código carrega os dados, realiza o pré-processamento, carrega o objeto scaler e o modelo, faz a previsão nos dados utilizando o modelo e avalia o desempenho do modelo usando métricas de avaliação.

Os dados são carregados a partir de um ficheiro CSV e são realizados o pré-processamento dos dados. São removidas as linhas com valores ausentes e duplicados. Em seguida, são calculados os IQR (intervalo interquartil) para identificar valores discrepantes utilizando a regra 1.5 x IQR. As observações com valores acima ou iguais a essa regra e com menos de 20 *Iterations* são considerados *outliers* e são removidos do conjunto de dados. Os dados são separados em matrizes X e y. A matriz X contém todas as colunas, exceto a última, e a matriz y contém a última coluna que representa a variável alvo. No final, é carregado o objeto *scaler* que vai fazer a normalização dos dados. Tanto o ficheiro *scaler* como o melhor modelo modelo foram guardados num ficheiro à parte

e posteriormente carregados através da ferramenta *Pickle*. O modelo é carregado e usado para realizar a previsão do modelo nos dados fornecidos. Depois de calculados os resultados, o programa fornece o RMSE (*Root Mean Squared Error*) entre os valores verdadeiros e as previsões.

V. Conclusão

Para concluir, o trabalho foi interessante, visto que conseguimos explorar alguns tipos diferentes de redes neuronais, tal como bastante analise de dados e o seu pré-processamento. É interessante ver que não precisámos de uma arquitetura muito complicada (uma *hidden layer* apenas com 10 n´