Trabalho Final: Predição do Tipo de Rocha

Frederico Schardong

1 Enunciado do trabalho

O enunciado do desafio escolhido propõem a classificação do tipo de rocha a partir de informações litoestratigráficas adquiridas através da perfuração do solo. Os mais de 90 poços foram perfurados no fundo do mar na costa da Noruega[1]. As informações litoestratigráficas disponíveis são resumidas na Figura 1.

Log Name	Log Description
FORCE_2020_LITHOFACIES_CONFIDENCE	Qualitiative measure of interpretation confidence
FORCE_2020_LITHOFACIES_LITHOLOGY	Interpreted Lithofacies
RDEP	Deep Reading Restitivity measurement
RSHA	Shallow Reading Restitivity measurement
RMED	Medium Deep Reading Restitivity measurement
RXO	Flushed Zone Resistivity measurment
RMIC	Micro Resisitivity measurment
SP	Self Potential Log
DTS	Shear wave sonic log (us/ft)
DTC	Compressional waves sonic log (us(ft)
NPHI	Neutron Porosity log
PEF	Photo Electric Factor log
GR	Gramma Ray Log
RHOB	Bulk Density Log
DRHO	Density Correction log
CALI	Caliper log
BS	Borehole Size
DCAL	Differential Caliper log
ROPA	Average Rate of Penetration
SGR	Spectra Gamma Ray log
MUDWEIGHT	Wheight of Drilling Mud
ROP	Rate of Penetration
DEPTH_MD	Measured Depth
x_loc	X location of sample
y_loc	Y location of sample
z_loc	Z (TVDSS) Depth of sample

Figure 1: Dados para cada poço.

As informações disponíveis para cada poço são incompletas, bem como não existe padronização quanto a profundidade de cada poço e a quantidade de leituras para cada um. Os organizadores deste dataset incluiram uma medida de confiança na classe atribuída para cada registro do dataset, disponível na coluna `FORCE_2020_LITHOFACIES_CONFIDENCE`, sendo 1 alto, 2 médio e 3 baixo.

[1] `https://xeek.ai/challenges/force-well-logs/overview`

Neste desafio, classificar rochas de forma errada apresenta penalidades distintas, desta forma, a matriz exibida na Figura 2 mostra a penalidade para as classificações possíveis. Por fim, a seguinte função é usada para calcular a pontuação de uma solução, onde A são os pesos da matriz mostrada na Figura 2, $\hat{y}i$ a classe correta e yi a classe prevista.

$$S = -\frac{1}{N}\sum_{i=0}^{N} A_{\hat{y}i,yi} \tag{1}$$

label \ prediction	Sandstone	Sandstone/Shale	Shale	Marl	Dolomite	Limestone	Chalk	Halite	Anhydrite	Tuff	Coal	Crystalline Basement
Sandstone	0	2	3.5	3	3.75	3.5	3.5	4	4	2.5	3.875	3.25
Sandstone/Shale	2	0	2.375	2.75	4	3.75	3.75	3.875	4	3	3.75	3
Shale	3.5	2.375	0	2	3.5	3.5	3.75	4	4	2.75	3.25	3
Marl	3	2.75	2	0	2.5	2	2.25	4	4	3.375	3.75	3.25
Dolomite	3.75	4	3.5	2.5	0	2.625	2.875	3.75	3.25	3	4	3.625
Limestone	3.5	3.75	3.5	2	2.625	0	1.375	4	3.75	3.5	4	3.625
Chalk	3.5	3.75	3.75	2.25	2.875	1.375	0	4	3.75	3.125	4	3.75
Halite	4	3.875	4	4	3.75	4	4	0	2.75	3.75	3.75	4
Anhydrite	4	4	4	4	3.25	3.75	3.75	2.75	0	4	4	3.875
Tuff	2.5	3	2.75	3.375	3	3.5	3.125	3.75	4	0	2.5	3.25
Coal	3.875	3.75	3.25	3.75	4	4	4	3.75	4	2.5	0	4
Crystalline Basement	3.25	3	3	3.25	3.625	3.625	3.75	4	3.875	3.25	4	0

Figure 2: Matriz de penalidades.

2 Resolução do Trabalho

Nesta seção é apresentado como os dados faltantes foram preenchidos, como a normalização foi realizada e quais as técnicas usadas para prever o tipo das rochas com base nas 20 dimensões de dados.

2.1 Preenchimento de Dados

O primeiro desafio neste conjunto de dados é lidar com a falta de dados. As 1170511 leituras nos poços, cada uma com 20 informações litoestratigráficas distintas resultaria em 23410220 dados. Entretanto, existe um total de 10074726 dados faltando, quase a metade do total.

A Figura 3 mostra todos os dados de todos os poços juntos, alinhados pela profundidade, que é o eixo Y da figura, enquanto cada uma das 20 dimensões é exibida por um gráfico distinto, especificado no eixo X.

Três diferentes estratégias foram testadas para preencher os dados faltantes, são elas:

1. Simples: substituir valores faltando por
   (a) Um valor fixo (0)
   (b) O valor mais frequênte de cada coluna

(c) A média de cada coluna

2. Iterativo [1], onde os dados são estimados com base nos seus vizinhos de forma *round-robin*

3. Utilizando o algoritmo KNN [3], que usa `k-Nearest Neighbors`

A mesma representação gráfica utilizada para visualizar os dados crús utilizado na Figura 3 é utilizada para visualizar o resultado de cada uma das técnicas acima. A Figura 5 mostra o resultado das três técnicas chamadas de `simples`, a Figura 4 mostra o método iterativo e a Figura 6 mostra o resultado do método KNN utilizando diferentes quantidades de vizinhos.

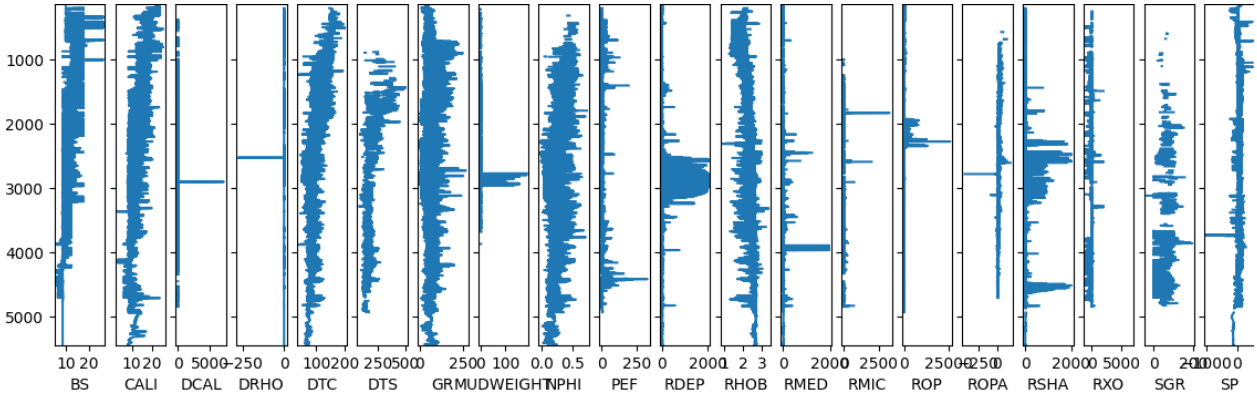


Figure 3: Dados de todos os poços, sendo X cada categoria litoestratigráfica e Y a profundidade do poço.

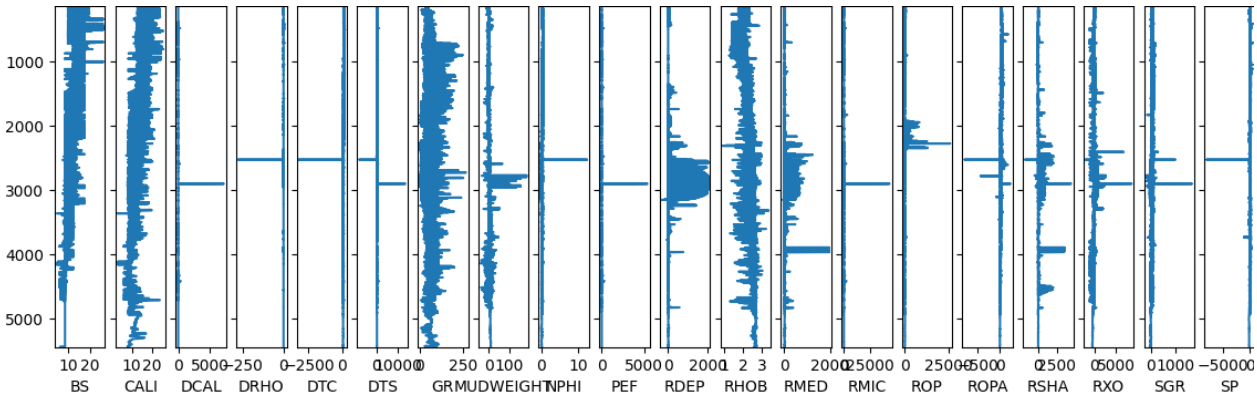
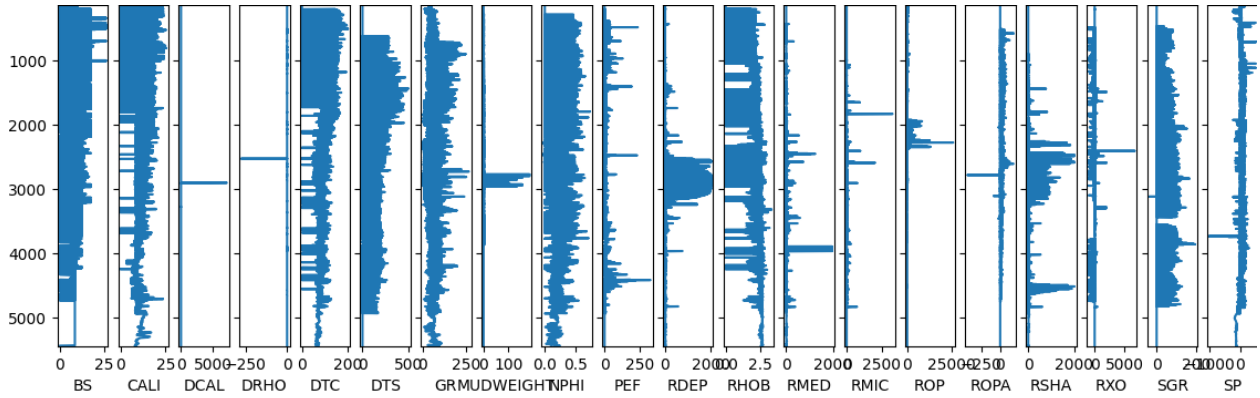
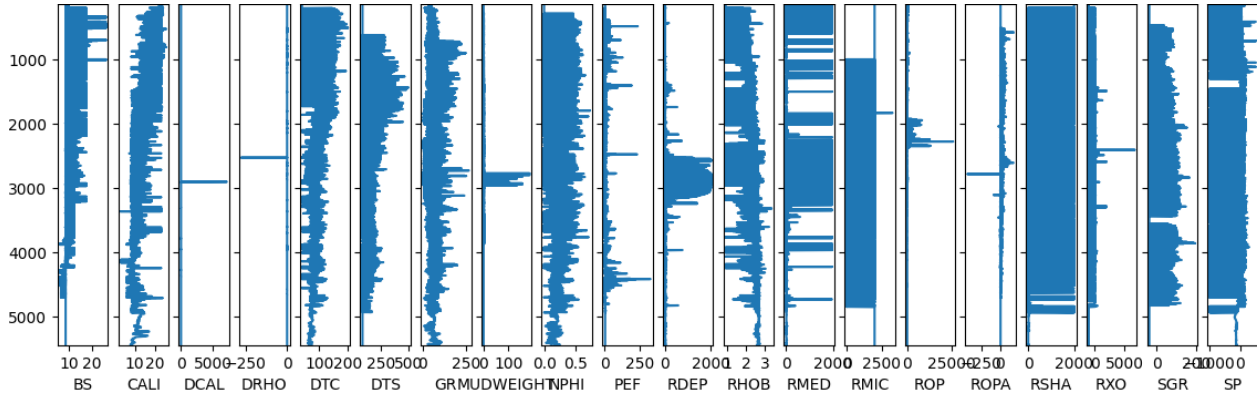


Figure 4: Dados de todos os poços, preenchimento de dados interativo [1]

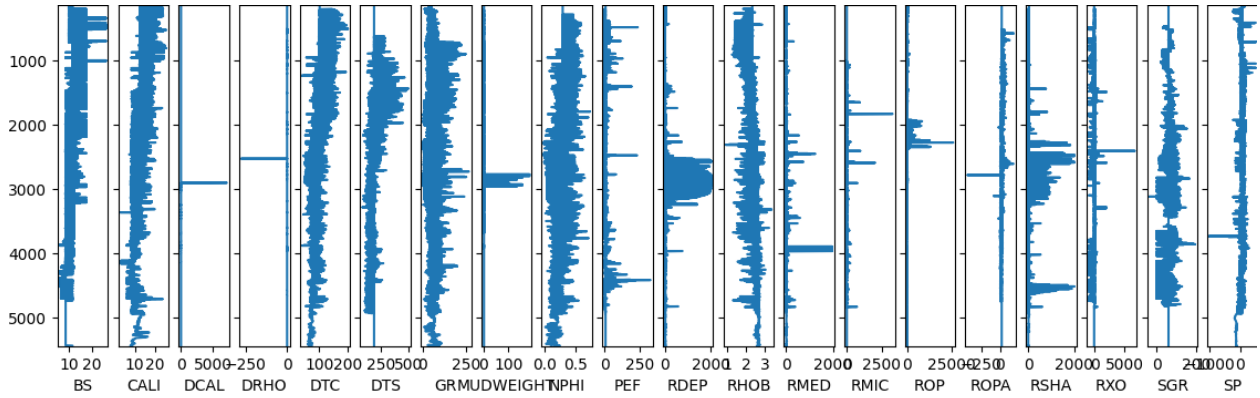
.



(a) Dados de todos os poços, preenchimento de dados `constante`, ou seja, substituindo dados faltando por 0.

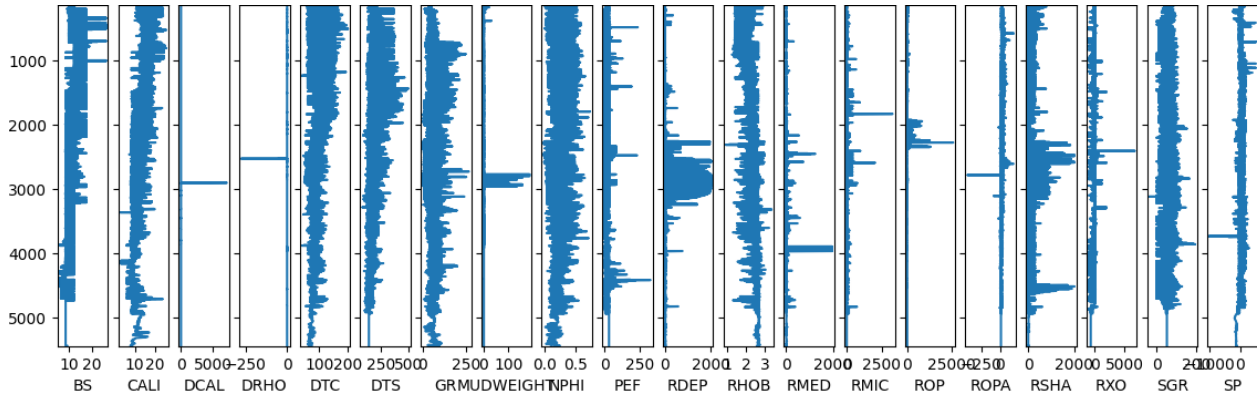


(b) Dados de todos os poços, preenchimento de dados `mais frequênte`, ou seja, copiando o dado mais frequênte de cada coluna.

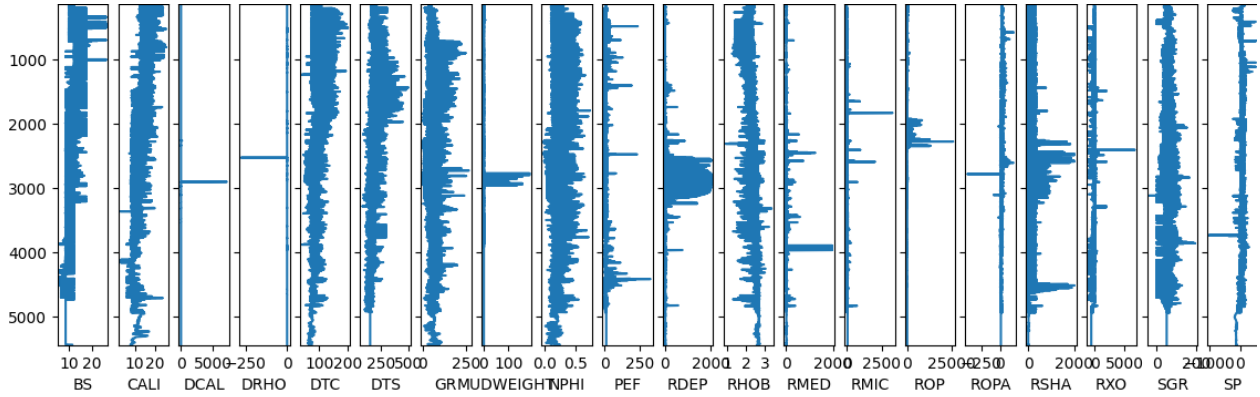


(c) Dados de todos os poços, preenchimento de dados `média`, ou seja, copiando a média de cada coluna.

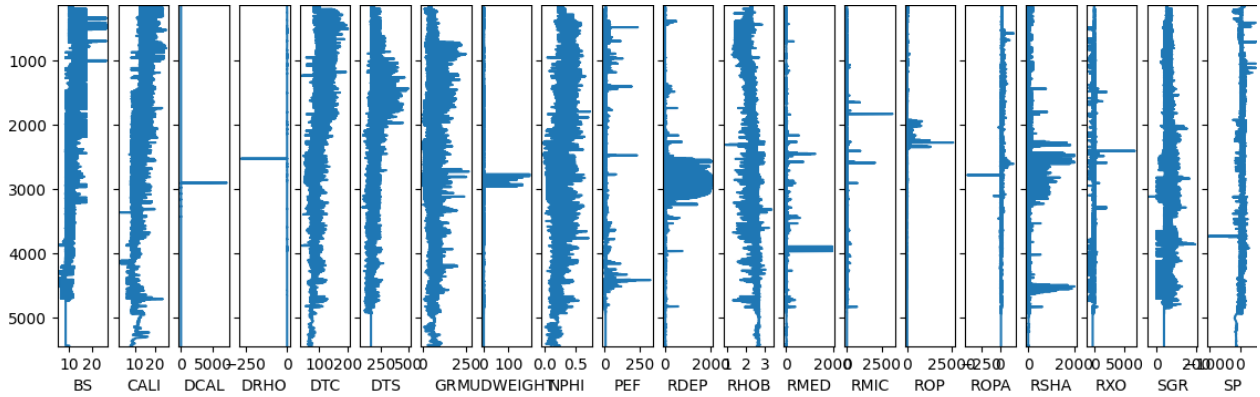
Figure 5: Dados faltando preenchidos com valor fixo (0), mais frequênte e média.



(a) Dados de todos os poços, preenchimento de dados `KNN` com $n = 5$.



(b) Dados de todos os poços, preenchimento de dados `KNN` com $n = 50$.



(c) Dados de todos os poços, preenchimento de dados `KNN` com $n = 500$.

Figure 6: Dados faltando preenchidos com o algoritmo KNN [3]

O desempenho das diferentes técnicas listadas acima foi medido através de testes utilizando uma rede *multi-layer perceptron* com os parâmetros fixos, apenas variando o dado de entrada onde os dados faltantes eram preenchidos com as diferentes técnicas. A técnica que apresentou melhor desempenho, que foi o `KNN` com 500 vizinhos.

Após preencher os dados faltantes, os poços foram agrupados em um único poço, reunindo as medições das profundidades a cada 5 metros, respeitando os diferentes níveis de confiança para cada rocha encontrada dentro dos intervalores de 5 metros. Por exemplo, todas as leituras no dataset entre 135 e 140 metros de profundidade foram reunidas em uma única profundidade 140, separando os tipos de rocha e os níveis de confiança deste intervalo de medições, como pode ser visto na Tabela 1. A técnica usada para agrupar os valores de cada uma das 20 dimensões foi calcular a média destes valores. Esta técnica reduziu os 23410220

dados para 213118.

Profundidade	Tipo de Rocha	Confiança
135	65030	1
140	30000	1
	65030	1
145	30000	1
150	30000	1
155	30000	1
160	30000	1
165	30000	1
	65000	2
	65030	1

Table 1: Resultado do agrupamento dos poços a cada 5 metros. Mostrando apenas as primeiras medições, excluindo as 20 dimensões de dados.

2.2 Normalização dos dados

Depois de completar os dados faltando nas 20 dimensões, o próximo passo consiste em normalizá-los. A Figura 7 mostra a distribuição dos dados em todas as dimensões antes de qualquer normalização (dados crús).

A normalização dos dados de entrada consistiu de três etapas. Na primeira etapa foi aplicada a técnica de `Yeo-Johnson` [2] em todas as dimensões, menos as `DTS, MUDWEIGHT, RXO, DCAL`, pois não apresentou mudança significativa para estes dados, como pode ser visto na Figura 8. Esta técnica tenta normalizar a distribuição dos dados, não se registringindo a manter o intervalo fixo (como `min max`). Na segunda etapa, os dados das dimensões não tratados na primeira etapa e os dados das dimensões `DRHO, ROPA, SP, RHOB` foram submetidos a normalização por quartil com o objetivo de normaliza-los, o resultado pode ser visto na Figura 9. Por fim, na última etapa todas as 20 dimensões foram normalizadas com `min max`, de modo a não mudar a distribuição dos dados, mas apenas transportá-los para o intervalo $[0, 1]$, como pode ser visto na Figura 10.

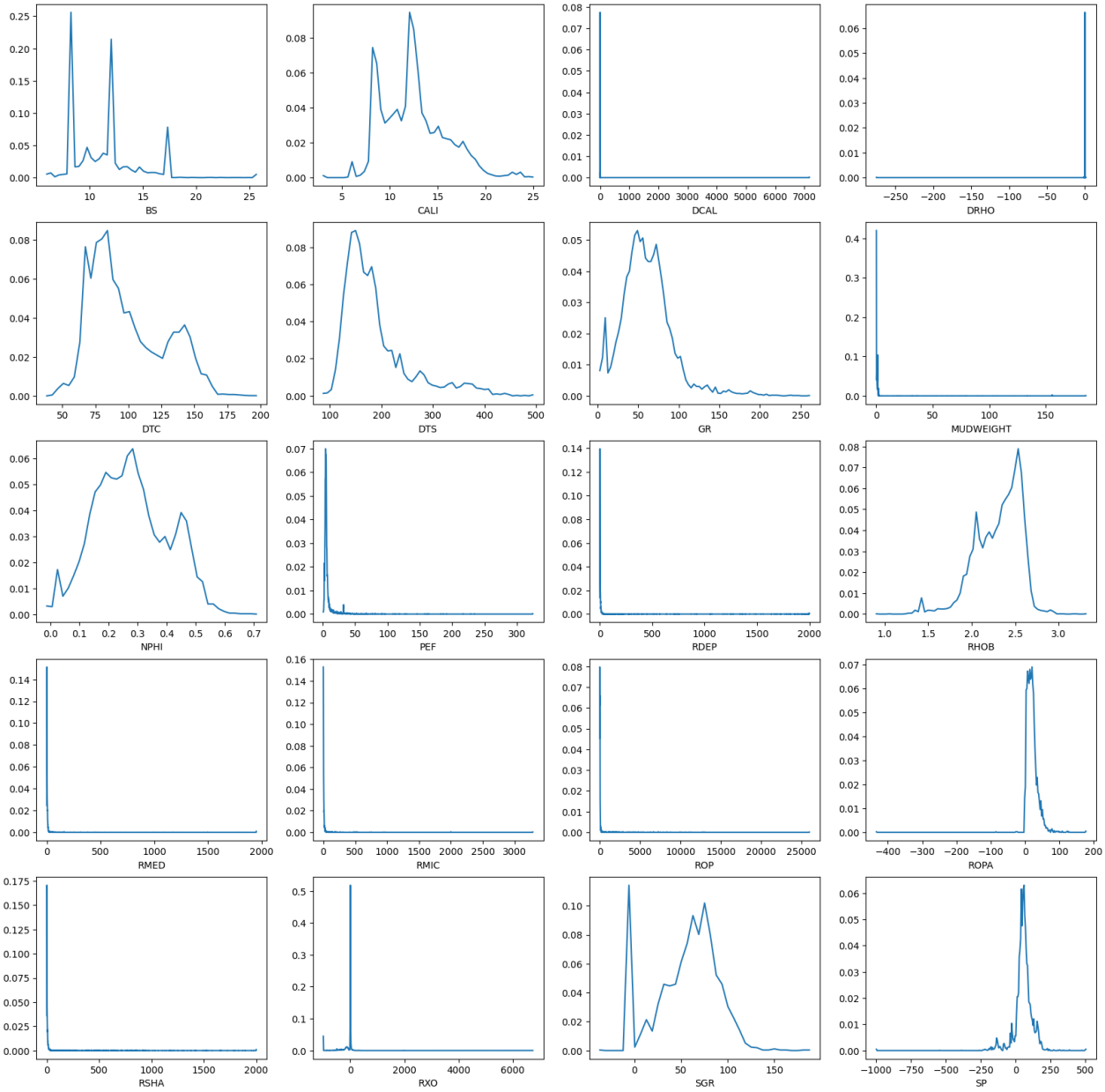


Figure 7: Distribuição dos dados em cada uma das 20 dimensões - antes da normalização.

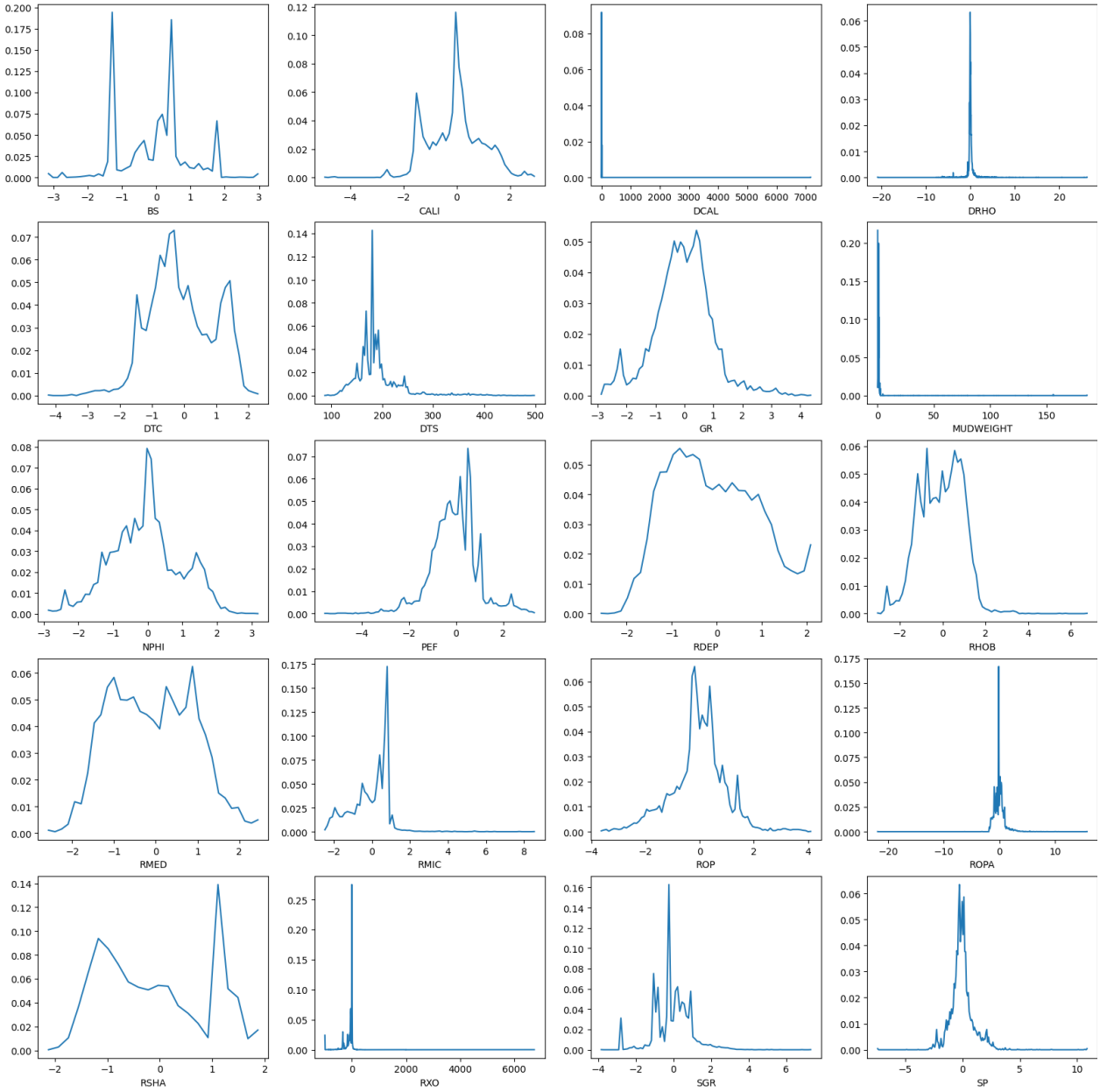


Figure 8: Primeira etapa de normalização, utilização de `Yeo-Johnson` [2].

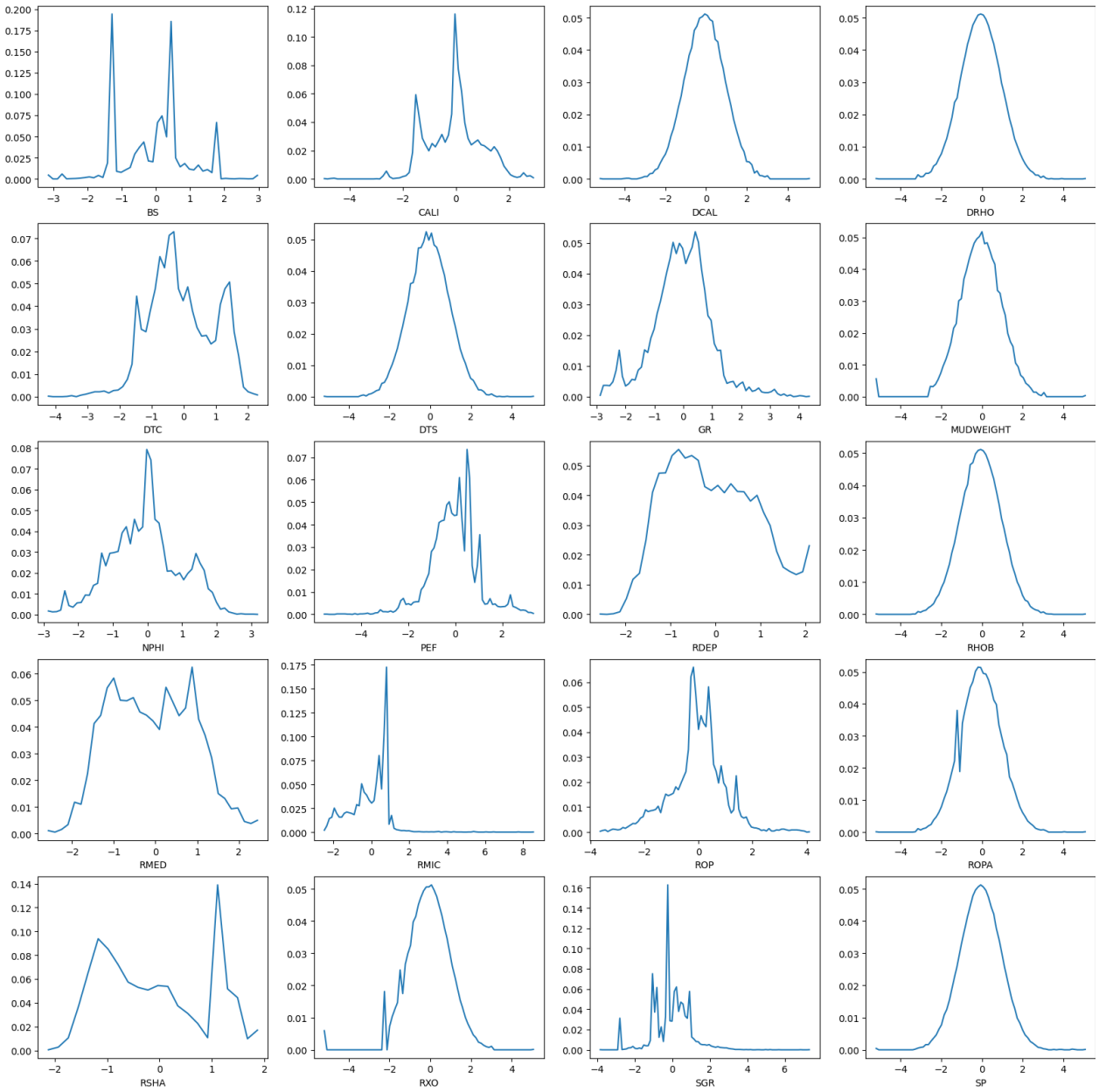


Figure 9: Segunda etapa de normalização, aplicação de quartil em algumas dimensões.

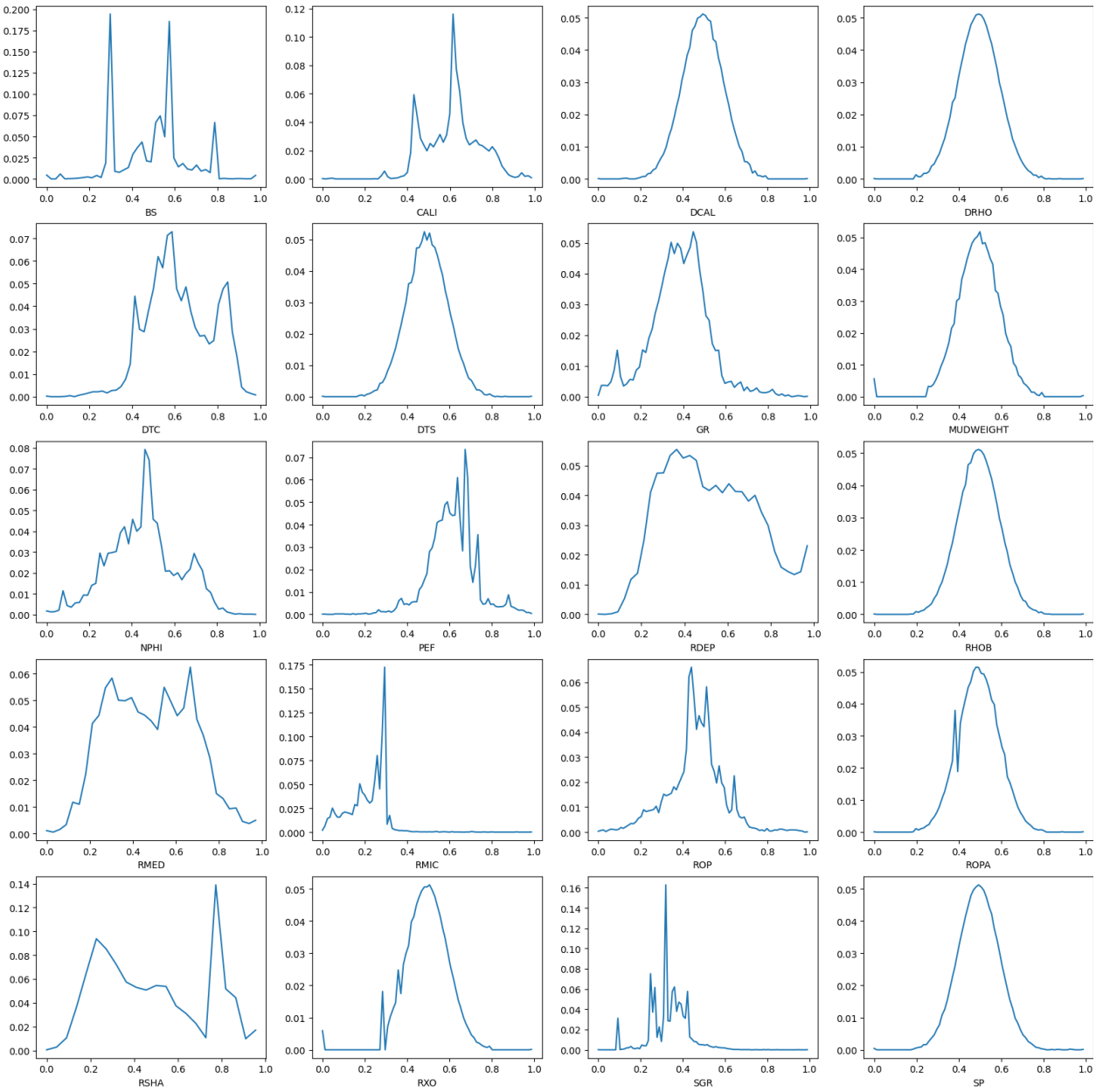


Figure 10: Terceira etapa de normalização, aplicação de `min max` em todas as dimensões.

2.3 Modelo Proposto

Nesta seção as três redes/modelos distintos que foram propostos serão descritos. Todos eles consistiram de uma busca pelos parâmetros que produziram os maiores `f1-score`. Para testar os parâmetros, os dados de treinamento (dados de 98 poços) foram divididos em 80% de treinamento e 20% de teste, e `4-fold validation` foi utilizado.

2.3.1 Multi-Layer Perceptron

O primeiro modelo adotado foi o classificador *multi-layer perceptron* do tipo *feed forward* pois foi discutido em profundidade na disciplina. Para encontrar a configuração que produz os melhores resultados, dezenas de configurações diferentes foram testadas. A biblioteca `scikit-learn` [6], utilizada para implementar a rede,

fornece a classe `GridSearchCV` que recebe como parâmetro as possíveis configurações de uma rede neural e executa cada possível combinação de configuração, retornando o `f1-score` de cada configuração testada.

Todos os parâmetros testados estão detalhados na Tabela 2. Os valores em negrito são os que apresentaram o maior `f1-score`.

Parâmetro	**Possíveis Valores**
Função de ativação	Tangente Hiperbólica, **Unidade Linear Retificada (ReLU)**, Sigmóide
Solucionador	Gradiente Descendente, Adam, **lbfgs**
Taxa de aprendizado	0.00001, 0.0001, 0.001, 0.01, **0.1**
Camadas escondidas	(50, 50), (50, 50, 50), (100), (100, 100), (100, 100, 100), (10, 50, 100), (100, 50, 10), (30, 60), (30, 60, 90), (30, 60, 90, 120), (60, 30), (90, 60, 30), (120, 90, 60, 30) **(100, 100)**, (150, 150), (200, 200), (100, 200), (100,300), (200,100), (300,100)

Table 2: Todas as configurações da rede *feed forward* testados. Em negrito os parâmetros que produziram o maior `f1-score`.

O `f1-score` da melhor configuração (em negrito) foi de 0.82.

2.4 SVM

Como o modelo descrito na seção anterior não leva em consideração a qualidade das leituras, duas outras técnicas cuja implementação do `scikit-learn` permite utilizar pesos foram testadas: `SVM` e `regressão logistica`.

Como no modelo anterior, o desempenho do `SVM` foi testado em relação a combinação de multiplos parâmetros, listados na tabela 3 e marcados em negrito os valores que apresentaram o maior `f1-score`.

Parâmetro	**Possíveis Valores**
γ	**Automático**, Escala
C	0.1, **1**, 10
Grau do polinômio	**2**, 3, 4, 5
Kernel	linear, **poly**, rbf, sigmóide
Peso das classes	sem peso, {1:3, 2:2, 3:1}, **{1:20, 2:10, 3:1}**, {1:100, 2:10, 3:1}

Table 3: Todas as configurações do `SVM` testados. Em negrito os parâmetros que produziram o maior `f1-score`

O `f1-score` da melhor configuração (em negrito) foi de 0.75.

2.5 Regressão Logística

Como no modelo anterior, o desempenho da `regressão logística` foi testada em relação a combinação de multiplos parâmetros, listados na tabela 4 e marcados em negrito os valores que apresentaram o maior `f1-score`.

Parâmetro	**Possíveis Valores**
Algorítmo	newton-cg, **lbfgs**, liblinear, sag, saga
Tipo de multiclasse	**auto**, ovr, multinomial
C	0.1, **1**, 10
Peso das classes	**sem peso**, {1:3, 2:2, 3:1}, {1:20, 2:10, 3:1}, {1:100, 2:10, 3:1}

Table 4: Todas as configurações da `regressão logística` testados. Em negrito os parâmetros que produziram o maior `f1-score`

O `f1-score` da melhor configuração (em negrito) foi de 0.68.

2.6 Detalhes de implementação

O trabalho foi implementado em Python 3, e as seguintes bibliotecas foram utilizadas:

- `pandas` [5] para carregar os dados do arquivo CSV e agrupar os poços;
- `scikit-learn` [6] para normalizar os dados, e também criar, treinar e testar os diferentes classificadores;
- `numpy` [7] para gerar os histogramas;
- `matplotlib` [4] para geração de gráficos.

Todas as bibliotecas podem ser instaladas através do gerenciador de pacotes padrão do Python, o `pip`, através do comando:

```
$ pip install scikit-learn numpy matplotlib pandas
```

O código fonte desta tarefa é publico[2] e possui a licensa MIT. Ele foi submetido junto com este relatório. Para reproduzir as imagens listas neste relatório e presentes na pasta resultados, basta executa o comando abaixo.

```
$ python3 main.py
```

3 Resultados

O desafio fornece dados de 10 poços para testar a capacidade preditiva da rede, porém estes dados não constam a resposta. O site também não disponibiliza a classificação destas leituras. Desta forma, 20% dos dados de treinamento foram utilizados para testar os modelos. Utilizando a melhor configuração encontrada no estágio de preparação de cada uma das três técnicas testadas, os resultados encontrados estão dispostos na Tabela 5.

	f1-score	score
Multi-Layer Perceptron	0.81	−0.42
SVM	0.74	−0.55
Logistic Regression	0.67	−0.64

Table 5: `f1-score` e **score** considerando a matriz de penalidade (Figura 2) e a Equação 1.

4 Conclusão

É possível concluir que o objetivo proposto para este trabalho de prever o tipo de rocha, levando em consideração a matriz de penalidade foi atingido. Diferentes técnicas para preencher os dados faltando foram testadas, bem como uma forma não-triviável para normalizar os dados foi desenvolvida. Foram implementadas três técnicas distintas para fazer a previsão dos tipos de rochas.

References

[1] Samuel F Buck. “A method of estimation of missing values in multivariate data suitable for use with an electronic computer”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 22.2 (1960), pp. 302–306.

[2] In-Kwon Yeo and Richard A Johnson. “A new family of power transformations to improve normality or symmetry”. In: *Biometrika* 87.4 (2000), pp. 954–959.

[2] Código fonte disponível em `https://github.com/fredericoschardong/force-well-challenge`.

[3] Olga Troyanskaya et al. “Missing value estimation methods for DNA microarrays”. In: *Bioinformatics* 17.6 (2001), pp. 520–525.

[4] John D Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[5] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[6] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[7] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. “The NumPy array: a structure for efficient numerical computation”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.