UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Matheus Henrique Schaly

Aplicação de Aprendizado de Máquina na Classificação de Litofácies

Florianópolis
2021

Matheus Henrique Schaly

Aplicação de Aprendizado de Máquina na Classificação de Litofácies

Trabalho de Conclusão de Curso do Curso de Graduação em Ciências da Computação do Campus Florianópolis da Universidade Federal de Santa Catarina para a obtenção do título de bacharel em Ciências da Computação.
Orientador: Prof. Dr. Mauro Roisenberg

Florianópolis
2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Schaly, Matheus Henrique
Aplicação de aprendizado de máquina na classificação de litofácies / Matheus Henrique Schaly ; orientador, Mauro Roisenberg, 2021.
p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2021.

Inclui referências.

1. Ciências da Computação. 2. Aprendizado de Máquina. 3. Classificação Automática de Litofácies. I. Roisenberg, Mauro. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

Matheus Henrique Schaly

Aplicação de Aprendizado de Máquina na Classificação de Litofácies

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "bacharel em Ciências da Computação" e aprovado em sua forma final pelo Curso de Graduação em Ciências da Computação.

Florianópolis, 30 de 08 de 2021.

Prof. Dr. Jean Everson Martina
Coordenador do Curso

Banca Examinadora:

Prof. Dr. Mauro Roisenberg
Orientador

Profa. Dra. Jerusa Marchi
Avaliadora
Universidade Federal de Santa Catarina

Prof. Dr. Elder Rizzon Santos
Avaliador
Universidade Federal de Santa Catarina

Este trabalho é dedicado a todos que fizeram parte desta minha jornada universitária.

AGRADECIMENTOS

Agradeço ao meu pai Arno Rui Schaly, minha mãe Márcia Helena Schaeffer, minha tia Maerly Cristine Schaeffer Fertig e a toda minha família, pela paciência, carinho, orientação e amor durante os mais diversos momentos da minha vida universitária.

Agradeço a todos os meus amigos que me acompanharam e apoiaram durante a graduação, em especial, Francisco Luiz Vicenzi e Maurício Macário de Farias Júnior.

Agradeço a todos os professores do curso de Ciências da Computação e do Departamento de Informática e Estatística, em especial ao meu orientador Mauro Roisenberg, que são os responsáveis por construir um curso de excelência.

Agradeço à Universidade do Vale do Itajaí que me guiou no início da minha graduação, a Universidade Federal de Santa Catarina que me permitiu uma experiência imensamente enriquecedora e a University of Ottawa por ter me acolhido como aluno intercambista.

A todos, muito obrigado.

"It's in responsibility that most people find
the meaning that sustains them through life.
It's not in happiness. It's not in impulsive pleasure."
(Jordan B. Peterson)

RESUMO

Classificação de litofácies é uma tarefa realizada por geólogos que consiste em analisar uma série de registros elétricos e físico-químicos obtidos através de sensores que percorrem a parede de um poço perfurado e, a partir das leituras destes, identificar que unidades litológicas (litofácies) caracterizam o ambiente de formação e os aspectos composicionais das rochas. O objetivo do trabalho é propor um modelo eficaz de aprendizado de máquina (do inglês Machine Learning (ML)), incluindo a parte de manipulação dos dados, para a classificação de litofácies em poços geológicos. O ramo de ML vem se tornando uma ferramenta cada vez mais importante em vários campos da ciência, neste trabalho aplicaremos ML no ramo das geociências. As técnicas padrão de ML podem levar a problemas de ambiguidade já que duas litofácies diferentes podem apresentar os mesmos valores dos sensores. Acreditamos que levar em consideração a sequência de padrões sedimentares possa ajudar no processo de desambiguação da classificação. Para isso poderia ser utilizado uma versão modificada de uma rede neural recorrente (do inglês Recurrent Neural Network (RNN)). Sabemos que a classificação acurada de litofácies é de grande importância para obter informações geológicas úteis para a exploração e produção de hidrocarbonetos. Além disso, a classificação automática de litofácies torna o processo de estudo da litologia dos poços mais rápido e menos oneroso. A classificação de litofácies é realizada estudando as propriedades litológicas das rochas encontradas em poços, que são características dos sedimentos atuais acumulados em determinadas condições físicas e geográficas. As propriedades litológicas podem incluir raio gama, resistividade, efeito fotoelétrico, perfil neutrônico, entre outras. Dado um banco de dados contendo as características e a classificação das litofácies, é esperado que o modelo proposto consiga, de maneira eficaz, realizar automaticamente a classificação de tais litofácies. A eficácia do método será medida através da métrica de classificação acurácia, assim como de uma métrica customizada chamada *score*.

Palavras-chave: Aprendizado de máquina, classificação automática de litofácies.

ABSTRACT

Lithofacies classification is a task performed by geologists that consists of analyzing a series of electrical and physicochemical records obtained through sensors that run along the wall of a drilled well and, from these, identify which lithological units (lithofacies) characterize the environment of formation and the compositional aspects of the rocks. This work proposes an effective model of ML, including data manipulation, for the classification of lithofacies in geological wells. The branch of ML has become an increasingly important tool in various fields of science, in this work we will apply ML in the field of geosciences. Standard ML techniques can lead to ambiguity problems since two different lithofacies can have the same sensor values. We believe that taking into account the sequence of sedimentary patterns can help in the classification disambiguation process. To achieve that, a modified version of a recurrent neural network (RNN) could be used. Accurate classification of lithofacies is of great importance to obtain useful geological information for the exploration and production of hydrocarbons. The automatic classification of lithofacies makes the process of studying the lithology of the wells faster and less costly. The classification of lithofacies is performed by studying the lithological properties of the rocks found without wells, which are characteristic of the current sediments accumulated under certain physical and geographical conditions. Lithological properties can include gamma ray, resistivity, photoelectric effect, neutron profile, among others. Given a database containing the characteristics and the classification of lithofacies, it is expected that the proposed model will be able to effectively carry out the classification of such lithofacies automatically. The method's effectiveness will be measured through the accuracy rating metric, as well as a custom metric called score.

Keywords: Machine learning, automatic classification of lithofacies.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

1D-CNN	Unidimensional Convolutional Neural Network
AUC	Area Under the ROC Curve
BOS	bafflestone filoidalgal (calcário)
BRNN	Bidirectional Recurrent Neural Network
BS	bafflestone filoidalgal (calcário)
CA	Classification Accuracy
CALI	calibrador
CNN	Convolutional Neural Network
CSiS	siltito grosso não marinho
D	dolomita
DCAL	calibrador diferencial
DCNN	Deep Convolutional Neural Network
DeltaPHI	porosidade de nêutron e diferença de porosidade de densidade
DEN	densidade
DL	Deep Learning
DNN	Deep Neural Network
DRHO	correção de densidade
DTC	tempo de trânsito da onda compressiva
DTS	tempo de trânsito da onda cisalhante
FSiS	siltito fino não marinho
GR	raios gama
GRU	Gated Recurrent Unit
IA	Inteligência Artificial
ILDlog10	base de registro de resistividade verdadeira aparente
KNN	K-Nearest Neighbors
LSTM	Long Short-Term Memory
M	argilito
ML	Machine Learning
MS	lamito (calcário)
MSS	siltito marinho e xisto
NCS	siltito grosso não marinho
NEU	porosidade de nêutrons
NFS	siltito fino não marinho
NPHI	porosidade a partir do perfil neutrônico
NS	arenito não marinho
PAB	calcário
PE	efeito fotoelétrico
PEF	absorção fotoelétrica

PG	packstone-grainstone
PHI	média de porosidade de nêutron e densidade
PS	packstone grainstone (calcário)
RDEP	medição de resistividade de leitura profunda
ReLU	Rectified Linear Unit
RHOB	densidade aparente
RMIC	medição de resistividade de leitura micro
RNN	Recurrent Neural Network
ROP	taxa de penetração
ROPA	taxa média de penetração
RXO	medição de resistividade de zona lavada
SGR	raio gama espectral
SiSh	siltito marinho e xisto
SP	potencial espontâneo
SS	arenito não marinho
SVM	Support Vector Machine
TVDSS	True Vertical Depth SS
VSH	fração de volume de xisto
W	wackestone
WS	wackestone (calcário)

SUMÁRIO

1 INTRODUÇÃO

Há diversas definições existentes para o termo fácies. Definimos fácies como qualquer parte restrita não comparável de uma unidade estratigráfica projetada que exibe caráter significativamente diferente de outras partes da unidade (MOORE, 1949). Biofácies são fácies identificadas por características paleontológicas (conteúdo fóssil) sem levar em conta o caráter litológico. Litofácies são fácies identificadas com base em características litológicas (BOGGS, 2001). Usaremos litofácies como base de dados no presente trabalho.

A classificação de litofácies é atribuir uma classe de rocha a uma amostra específica com base nas características medidas. A fonte ideal para classificação de litofácies são amostras de núcleo de rochas extraídas de poços. No entanto, devido aos custos associados, nem sempre as amostras de núcleo podem ser obtidas. Além disso, o método convencional é um processo tedioso e demorado, pois consiste em classificar litofácies manualmente por intérpretes humanos. Portanto, um método para classificar fácies a partir de medidas indiretas (por exemplo, gerar perfis utilizando sensores presos a cabo de aço) é necessário. Várias abordagens distintas para a questão da classificação de fácies utilizando dados de poços já foram propostas (MANDAL; REZAEE, 2019). Neste trabalho será investigado um conjunto de 118 perfis de poços que possui 27 atributos e 12 classes.

Nos últimos anos, aprendizado de máquina (do inglês Machine Learning (ML)) se tornou uma ferramenta interdisciplinar cada vez mais importante, que avançou vários campos da ciência, como biologia, química, medicina e farmacologia. Especificamente, o método de rede neural profunda (do inglês Deep Neural Network (DNN)) encontrou ampla aplicação. Enquanto a geociência foi mais lenta na adoção, a bibliometria mostra adoção do aprendizado profundo (do inglês Deep Learning (DL)) em todos os aspectos da geociência (DRAMSCH, 2020).

Aprendizado de máquina está profundamente enraizado na estatística aplicada, criando modelos computacionais que usam inferência e reconhecimento de padrões em vez de conjuntos explícitos de regras (DRAMSCH, 2020). Aprendizado de máquina é o campo de estudo que fornece aos computadores a capacidade de aprender sem serem explicitamente programados (SAMUEL, 1959). Aprendizagem supervisionada consiste na tarefa de um algoritmo de ML em aprender uma função que mapeia uma entrada para uma saída com base em exemplos de pares de entrada e saída (RUSSELL; NORVIG, 2010). Uma função é inferida a partir de dados de treinamento rotulados que consistem em um conjunto de exemplos de treinamento (MOHRI; ROSTAMIZADEH; TALWALKAR, 2012).

O DL é uma forma de ML que permite que os computadores aprendam com a experiência e entendam o mundo em termos de uma hierarquia de conceitos. A hierarquia de conceitos permite que o computador aprenda conceitos complicados construindo-os a partir de outros mais simples (GOODFELLOW; BENGIO; COURVILLE, 2016).

Recentemente, as técnicas de DL foram desenvolvidas e amplamente adotadas para extrair informações de vários tipos de dados. Considerando as diferentes características dos dados de entrada, existem vários tipos de arquiteturas para DL, como a rede neural recorrente (do inglês Recurrent Neural Network (RNN)), rede neural convolucional (do inglês CNN), e DNN. Diferentemente da DNN, a RNN pode lidar com dados que possuem informações temporais. Portanto, em áreas de pesquisa que contêm dados sequências, como texto, áudio e vídeo, RNNs são dominantes. Contudo, RNNs são incapazes de aprender as informações relevantes dos dados de entrada quando o intervalo de entrada é grande. A diferença entre uma RNN e uma LSTM está na função de portão. Portanto, ao introduzirmos funções de portão na estrutura da célula de uma RNN passamos a ter uma LSTM. Com a LSTM podemos lidar bem com o problema das dependências de longo prazo. Desde a introdução da RNN quase todos os resultados interessantes baseados em RNNs foram alcançados pela LSTM. A LSTM se tornou o foco do DL (YU, Y. *et al.*, 2019).

O problema de classificação automática de litofácies deve ser explorado a fim de diminuir os custos envolvidos na classificação manual de litofácies. Existem competições envolvendo a classificação automática acurada de perfis de poços por meio de algoritmos de ML. No presente trabalho avaliaremos a competição, já encerrada, chamada Force 2020 Machine Learning Competition[1] (2020, s.d.), na qual o modelo de ML vencedor da competição utilizou o algoritmo XGBoost. Nossa solução ao problema de classificação de litofácies utilizará um algoritmo de ML supervisionado. Acreditamos que a sequência de padrões sedimentares possa ajudar no processo de classificação. Portanto, podemos, mais especificamente, criar uma nova topologia de LSTM que venha a considerar este aspecto sequencial do nosso banco de dados. Além disso, criaremos um *pipeline* para a manipulação dos dados para organizar e melhorar os dados de entrada ao modelo.

1.1 OBJETIVOS

As técnicas de ML normalmente utilizadas para o problema da classificação automática de fácies constituem em modelos estáticos no sentido de que a previsão da classe para uma dada profundidade do poço é resultado apenas dos valores dos sensores correspondentes àquela profundidade, o que em tese pode resultar um problemas de ambiguidades já que duas litofácies diferentes poderiam apresentar os mesmos valores dos sensores. A desambiguação poderia ser feita levando em consideração não apenas os valores instantâneos dos sensores em dada profundidade, mas dos valores lidos em profundidades acima e abaixo do referido ponto, o que levaria para o modelo o processo sedimentar que levou a deposição daquele tipo de rocha naquela profundidade.

[1] Link para o site da competição: https://xeek.ai/challenges/force-well-logs/overview. Link para o GitHub da competição: https://github.com/bolgebrygg/Force-2020-Machine-Learning-competition/tree/master/lithology_competition

Os objetivos são divididos em:

1.1.1 Objetivo Geral

Utilizar e avaliar um modelo de ML que seja capaz de modelar a sequência temporal de formação das diversas camadas de rocha. Mais especificamente, utilizaremos as redes LSTM no problema da classificação automática de litofácies.

1.1.2 Objetivos Específicos

a) Estudo sobre litofácies e classificação automática de litofácies utilizando ML.

b) Levantamento da literatura buscando técnicas que já foram utilizadas para esta tarefa e tarefas similares.

c) Utilização dos dados da Force 2020 Machine Learning Competition.

d) Construção de um *pipeline* para a manipulação dos dados que serão utilizados como entrada para o algoritmo de ML proposto.

e) Desenvolvimento de um algoritmo de classificação de ML que leve em consideração a sequência dos dados.

f) Avaliação dos resultados obtidos com o modelo criado e comparação do modelo criado com outros modelos já existentes da competição Force 2020 Machine Learning Competition.

1.2 MÉTODO DE PESQUISA

Iniciamos o trabalho com o estudo teórico de litofácies e a importância da utilização de modelo de ML para a classificação automática de litofácies. Além disso, também realizamos um estudo sobre Inteligência Artificial (IA) e seus subcampos, partindo de ML, passando pela RNN e chegando a LSTM.

Em seguida realizamos o levantamento da literatura, na área de classificação de litofácies, onde fizemos uma análise crítica dos trabalhos que tentam solucionar o problema de classificação de litofácies utilizando modelos de ML.

Finalmente apresentamos o banco de dados utilizado, implementamos o algoritmo de ML proposto, e seguimos com a conclusão e comparação dos resultados obtidos pelo algoritmo proposto e outros algoritmos da competição.

1.3 ESTRUTURA DO TRABALHO

O capítulo 2 aborda alguns conhecimentos necessários para o entendimento do trabalho, relacionado-os com o problema em questão.

O capítulo 3 apresenta trabalhos já existentes na área de classificação de litofácies por modelos de ML. Neste capítulo também é realizado a análise crítica entre os trabalhos apresentados.

O capítulo 4 detalha o conjunto de dados e como foi realizado o pré-processamento dos dados. Também são descritos o ambiente e o linguagem de programação utilizada

O capítulo 5 apresenta e explica os modelos utilizados para os experimentos, e avalia a qualidade dos modelos. Em seguida, selecionamos o melhor modelo, e comparamos ele aos resultados obtidos pelos 5 primeiros colocados na competição.

Por fim, no capítulo 6 realizamos a conclusão do trabalho e sugerimos possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Aqui serão apresentados os conceitos principais para a realização deste trabalho. Começaremos apresentando a definição de fácies sedimentares, litofácies, e uma breve descrição dos atributos utilizados para a classificação de litofácies. Em seguida, será feito uma breve introdução aos conceitos de IA, ML, DL, RNN e LSTM.

2.1 FÁCIES SEDIMENTARES

Um ambiente de deposição (ou ambiente sedimentar) (Figura 1) é um tipo específico de local no qual os sedimentos são depositados, como um canal de riacho, um lago ou o fundo do oceano profundo (COLLEGE, s.d.). Rochas sedimentares podem ser formadas apenas onde os sedimentos são depositados por tempo suficiente para se compactar e cimentar em camadas ou estratos duros. A sedimentação normalmente ocorre em áreas onde o sedimento permanece intacto por muitos anos em bacias sedimentares. Enquanto algumas dessas bacias são pequenas, outras ocupam milhares de quilômetros quadrados e geralmente possuem vários ambientes locais deposicionais diferentes. Fatores físicos, químicos e biológicos influenciam esses ambientes e as condições que eles produzem determinam em grande parte a natureza dos sedimentos que se acumulam. Vários ambientes locais diferentes (sedimentares) podem, portanto, existir lado a lado dentro de uma bacia, à medida que as condições mudam lateralmente; as rochas sedimentares que, em última instância são ali produzidas, podem estar relacionadas a esses ambientes deposicionais. Essas rochas sedimentares diferentes, mas contemporâneas e justapostas, são conhecidas como fácies sedimentares (BRITANNICA, s.d.).

Por exemplo, uma fácies de praia geralmente pode ser distinguida de uma fácies plana de maré, ambas as quais foram depositadas ao mesmo tempo adjacentes uma à outra. Em comparação com a fácies da praia, a fácies plana da maré terá um tamanho médio de grão de sedimento menor, mais fósseis de bioturbação, conterá camadas cruzadas e ondulações criadas por correntes de maré e terá mais moluscos ou outros fósseis de águas rasas preservados em seu lugar original, em forma ininterrupta. Não haverá uma fronteira nítida entre as duas fácies preservadas no registro sedimentar. Em vez disso, a fronteira entre eles será uma zona com camadas de sedimentos que se interpenetram e se misturam lateralmente de uma fácies para outra (COLLEGE, s.d.).

Abaixo (Figura 2) está um diagrama simplificado de três fácies sedimentares adjacentes entre si: uma fácies plana de praia e maré (combinadas), uma porção marinha ou perto da costa de uma plataforma continental e uma plataforma carbonática ou recife de alto mar. Os sedimentos da fácies plana da praia e da maré são principalmente areia, a fácies da baía é principalmente lama, e a fácies do recife é composta principalmente por conchas e corais que são feitos de minerais carbonáticos. Se esses sedimentos forem enterrados e litificados em rochas sedimentares, as areias da praia se transformam em

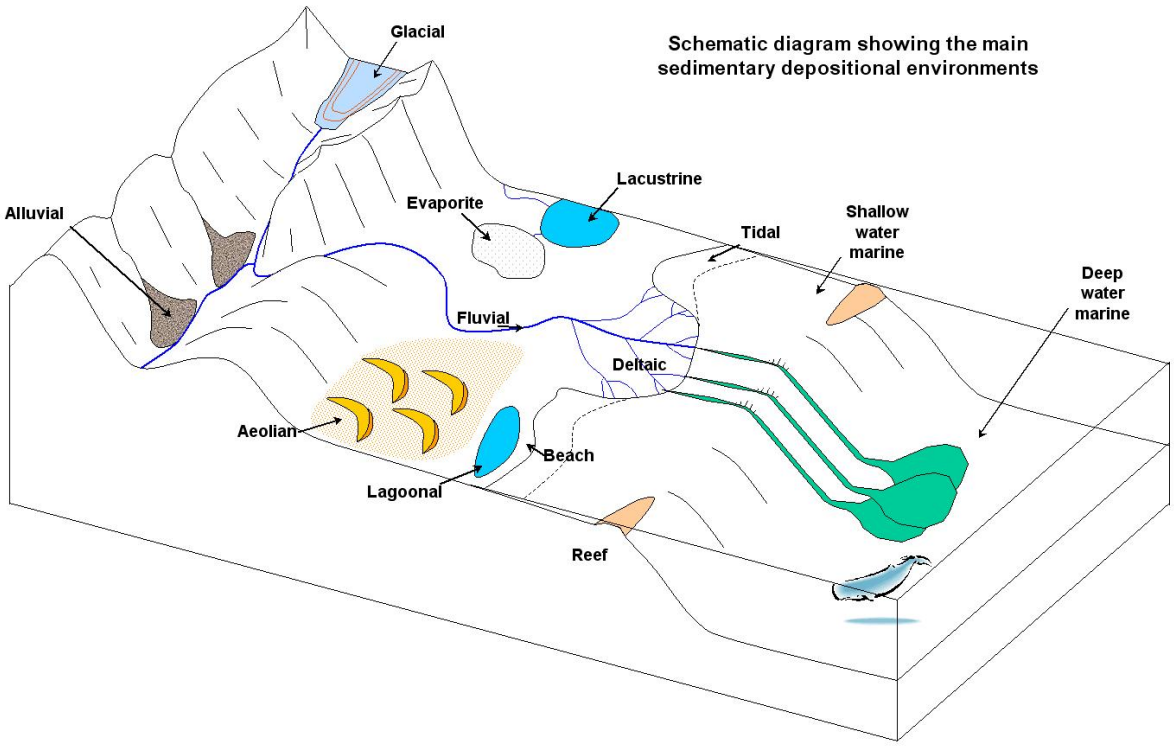


Figura 1 – Ambientes Deposicionais

Fonte – (COLLEGE, s.d.)

arenito, a lama da baía se transforma em xisto e os sedimentos do recife se transformam em calcário (COLLEGE, s.d.).

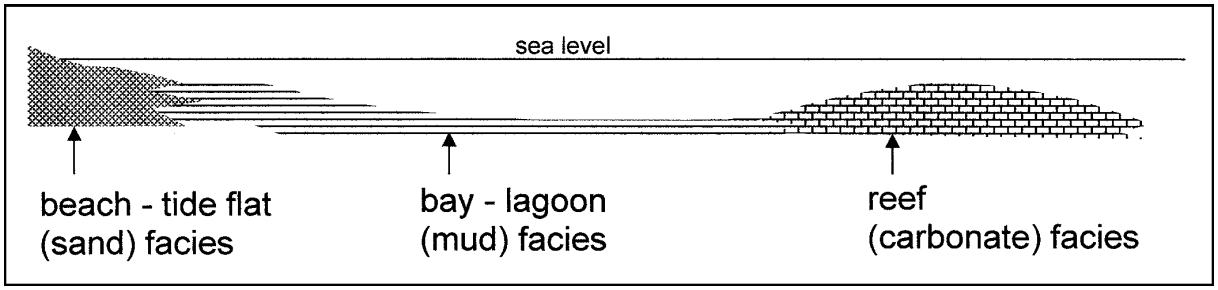


Figura 2 – Fácies Sedimentares

Fonte – (BRITANNICA, s.d.)

2.1.1 Litofácies

Existem várias maneiras de descrever ou designar fácies sedimentares. Ao observar as características físicas (ou litológicas) primárias, é possível reconhecer litofácies. Os atributos biológicos (ou mais corretamente, paleontológicos) - os fósseis - definem biofácies. Ambos são o resultado direto da história deposicional da bacia. Ao atribuir modos de origem a diferentes fácies (ou seja, interpretando as litofácies ou biofácies), pode-se visualizar um sistema genético de fácies (BRITANNICA, s.d.).

2.2 ATRIBUTOS DE LITOFÁCIES

Levantamentos de sensores presos a cabos de aço determinam propriedades físicas dentro e além da parede de um poço por dispositivos conectados a um cabo ou cabo de aço. As condições geológicas de subsuperfície e as características de engenharia podem ser derivadas direta ou indiretamente de uma ampla variedade de propriedades mensuráveis disponíveis por levantamento de cabos de aço. Os dados de vários métodos são frequentemente combinados para avaliar uma única característica geológica ou de engenharia (INTERIOR BUREAU OF RECLAMATION, 1998). A Force 2020 Machine Learning Competition utiliza 27 atributos de registro de poços, além de um atributo de confiança de interpretação, e outro atributo contendo a classe das litofácies, totalizando 29 atributos. A seguir apresentaremos uma breve descrição sobre cada um dos atributos (SCHLUMBERGER, s.d.).

- medida qualitativa de confiança de interpretação: 1 para alta, 2 para média, 3 para baixa.
- litofácies interpretadas: classe da litologia.
- poço: nome do poço.
- grupo: nome do grupo litoestratigráfico.
- formação: nome da formação litoestratigráfica.
- medição de resistividade de leitura profunda (RDEP)/média/rasa/micro: uma medição da resistividade da formação feita no tubo de perfuração a uma frequência na faixa de 100 kHz a 10 GHz, mais comumente 2 MHz. Na prática, vários transmissores podem ser usados para obter diferentes profundidades de investigação e obter compensação de poço.
- medição de resistividade de zona lavada (RXO): o volume próximo à parede do poço no qual todos os fluidos móveis foram deslocados pelo filtrado de lama.

- potencial espontâneo (SP): potencial elétrico de ocorrência natural (estático) na Terra. Os potenciais espontâneos são geralmente causados pela separação de carga na argila ou outros minerais, pela presença de uma interface semipermeável que impede a difusão de íons através do espaço dos poros das rochas, ou pelo fluxo natural de um fluido condutor (água salgada) através das rochas.

- tempo de trânsito da onda cisalhante (DTS) e tempo de trânsito da onda compressiva (DTC) (seg/ft): um tipo de registro acústico que exibe o tempo de viagem das ondas em relação à profundidade. Os perfis sônicos são normalmente registrados puxando uma ferramenta em um cabo de aço até o furo de poço. A ferramenta emite uma onda sonora que viaja da fonte para a formação e de volta para um receptor.

- porosidade a partir do perfil neutrônico (NPHI): referindo-se a um registro de porosidade com base no efeito da formação em nêutrons rápidos emitidos por uma fonte. Uma vez que o hidrogênio é encontrado principalmente nos fluidos dos poros, o registro da porosidade do nêutron responde principalmente à porosidade. O registro é calibrado para ler a porosidade correta assumindo que os poros são preenchidos com água doce e para uma dada matriz (calcário, arenito ou dolomita). O registro da porosidade do nêutron é fortemente afetado por argila e gás.

- absorção fotoelétrica (PEF): um registro das propriedades de absorção fotoelétrica. Como os fluidos têm números atômicos muito baixos, eles têm muito pouca influência, de modo que o fator fotoelétrico é uma medida das propriedades da matriz da rocha. Arenitos têm baixo fator fotoelétrico, enquanto dolomitos e calcários têm alto fator fotoelétrico. Argilas, minerais pesados e minerais contendo ferro têm alto fator fotoelétrico. Assim, o registro é muito útil para determinar a mineralogia.

- raios gama (GR): um registro da radioatividade natural total. Os xistos e as argilas são responsáveis pela maior parte da radioatividade natural, de modo que o registro de raios gama costuma ser um bom indicador dessas rochas. No entanto, outras rochas também são radioativas, notadamente alguns carbonatos e rochas ricas em feldspato. O perfil também é usado para correlação entre poços, para correlação de profundidade entre orifícios abertos e revestidos e para correlação de profundidade entre execuções de perfilagem.

- densidade aparente (RHOB): uma medição da densidade aparente da formação, com base na redução no fluxo de raios gama entre uma fonte e um detector devido ao espalhamento Compton. A medição responde à densidade média do material entre a fonte e o detector.

- correção de densidade (DRHO): uma correção para variações na densidade ou espessura da crosta terrestre. As correções isostáticas são comumente aplicadas aos dados de gravidade e são feitas de acordo com um modelo específico para isostasia.

- calibrador (CALI) e calibrador diferencial (DCAL): uma representação do diâmetro medido de um poço ao longo de sua profundidade.
- bafflestone filoidalgal (calcário) (BOS): O próprio tamanho do poço, incluindo o poço aberto ou parte não revestida do poço. O furo de poço pode se referir ao diâmetro interno da parede do furo de poço, a face da rocha que limita o furo perfurado.
- taxa média de penetração (ROPA): a velocidade média na qual a broca pode quebrar a rocha sob ela e, assim, aprofundar o furo de poço.
- raio gama espectral (SGR): é a última variante do registro de raios gama. A energia do raio gama captado pelo detector é proporcional ao brilho do pulso de luz que ele produz, e esse brilho, por sua vez, determina o tamanho do pulso elétrico produzido pelo fotomultiplicador. A energia dos raios gama é determinada por qual elemento os emitiu. As medições de raios gama espectrais oferecem várias vantagens, como na digitação de argila (PETROWIKI, s.d.).
- peso da lama de perfuração: a massa por unidade de volume de um fluido de perfuração.
- taxa de penetração (ROP): a velocidade na qual a broca pode perfurar a rocha sob ela e, assim, aprofundar o furo de poço.
- profundidade medida: O comprimento do furo de poço, como se determinado por uma régua de medição.
- localização X da amostra: localização da amostra na coordenada X.
- localização Y da amostra: localização da amostra na coordenada Y.
- profundidade Z (TVDSS) da amostra: localização da amostra na profunzidade Z. Profundidade vertical verdadeira SS (do inglês True Vertical Depth SS (TVDSS)).

2.3 APLICAÇÃO DE ML NA CLASSIFICAÇÃO DE LITOFÁCIES

Para resolver o problema de alto custo e tempo despendido na classificação de litofácies, vários estudos incorporaram algoritmos de ML, alimentado por dados e de baixo custo, usando apenas registros de poços para classificar fácies. Propriedades físicas de perfis de poço são usadas como atributos, enquanto fácies interpretadas de amostras são usadas como categoria verdadeira. Classificar fácies com base exclusivamente em características de perfis de poço é um desafio devido às suas diferenças nas resoluções, bem como valores de características sobrepostos para diferentes fácies. Embora as abordagens de estudos anteriores sejam robustas e capazes de prever as fácies com certo grau de precisão, as informações geológicas e as sequências de fácies estão ausentes, o que faz com

que os modelos prevejam sequências irrealísticas de fácies. É reconhecido que as fácies em camadas vizinhas são correlacionadas e os padrões de empilhamento de fácies são significativos para a interpretação geológica. Um modelo de ML baseado em sequência é, portanto, mais apropriado do que a abordagem tradicional de classificação multiclasse usada em estudos anteriores. Ele detecta naturalmente a sequência aprendendo com as fácies anteriores antes de fazer uma previsão (JAIKLA *et al.*, 2019).

Sendo assim, este trabalho pretende resolver o problema de custo associado a classificação manual de litofácies usando um algoritmo de ML capaz de levar em consideração a sequência das camadas de rocha. Em seguida será apresentado em maiores detalhes os conceitos básicos de algoritmos de ML que levam em consideração as sequências dos dados.

2.4 INTELIGÊNCIA ARTIFICIAL

Antes de entrar em detalhes sobre RNN, vamos primeiramente definir IA e algumas de suas subáreas.

Inteligência artificial é a automação de atividades que nós associamos com o pensamento humano, atividades como tomada de decisão, resolução de problemas e aprendizado. A IA é um dos campos mais recentes na ciência e engenharia. O termo originou-se em 1956 e seus estudos começaram logo após a Segunda Guerra Mundial. Atualmente, a IA engloba uma grande variedade de atividades, que vão do geral (aprendizagem e percepção) ao específico, como jogar xadrez, provar teoremas matemáticos, escrever poesia, dirigir um carro em uma rua movimentada e diagnosticar doenças. Essa variedade de atividades levou ao desenvolvimento da hierarquia de subáreas de IA, ilustradas na Figura 3 junto com exemplos representativos de cada subárea (RUSSELL; NORVIG, 2010). A seguir, a hierarquia será apresentada, partindo do termo mais abrangente ML, seguido por DL, RNN e chegando ao termo mais específico LSTM.

2.4.1 Aprendizado de Máquina

As dificuldades enfrentadas por sistemas que dependem do conhecimento manualmente codificado sugerem que os sistemas de IA precisam ser capazes de adquirir seu próprio conhecimento, extraindo padrões de dados brutos. Esse recurso é conhecido como ML. A introdução de ML permitiu aos computadores lidar com problemas que envolvem o conhecimento do mundo real, assim como tomar decisões que parecem subjetivas. Um algoritmo de ML é um algoritmo capaz de aprender com dados. De acordo com (MITCHELL, 1997) dizemos que um programa de computador aprende com experiência E com relação a alguma classe de tarefas T e medida de desempenho P, se seu desempenho nas tarefas em T, conforme medido por P, melhora com a experiência E (GOODFELLOW; BENGIO; COURVILLE, 2016). Por exemplo, um modelo que aprende a tarefa de classifi-

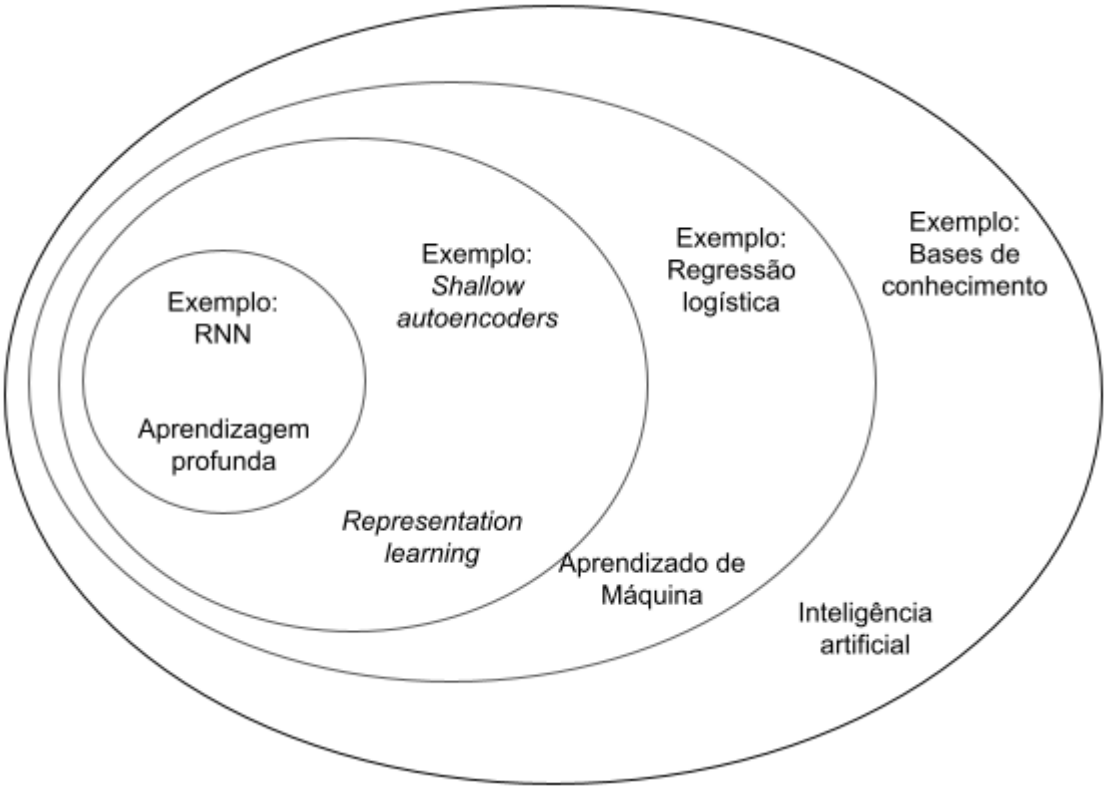


Figura 3 – Hierarquia das subáreas da inteligência artificial.

Fonte – adaptado de (GOODFELLOW; BENGIO; COURVILLE, 2016)

car rochas pode melhorar seu desempenho, medido pela acurácia da classificação, obtendo experiência a partir de dados de rochas já previamente categorizadas.

2.4.1.1 Aprendizagem Profunda

Os algoritmos tradicionais de ML funcionam bem em uma ampla variedade de problemas importantes. Eles não conseguiram, no entanto, resolver os problemas centrais de IA, como reconhecer a fala ou reconhecer objetos. O desenvolvimento de DL foi motivado em parte pela falha dos algoritmos tradicionais em generalizar bem essas tarefas de IA. O DL é um tipo particular de ML que atinge grande poder e flexibilidade ao representar o mundo como uma hierarquia aninhada de conceitos, com cada conceito definido em relação a conceitos mais simples e representações mais abstratas computadas em termos de conceitos menos abstratos (GOODFELLOW; BENGIO; COURVILLE, 2016).

Perceptron (Figura 4) é um tipo de neurônio artificial. Um perceptron recebe várias entradas binárias x_1, x_2, ... x_m, e produz uma única saída binária. Rosenblatt introduziu pesos, w_1, w_2, ... w_m, que são números reais que expressam a importância das respectivas entradas para a saída. A saída do neurônio, 0 ou 1, é determinada pelo resultado da soma ponderada $\sum_{j=1}^{m} w_j x_j$ ser menor ou maior que um determinado valor limite. Assim como os pesos, o limite é um número real e é também um parâmetro do neurônio. Em termos

algébricos temos a Equação 1 abaixo.

$$saída = \begin{cases} 0 & \text{se } \sum_{j=1}^{m} w_j x_j \leq \text{limite} \\ 1 & \text{se } \sum_{j=1}^{m} w_j x_j > \text{limite} \end{cases} \quad (1)$$

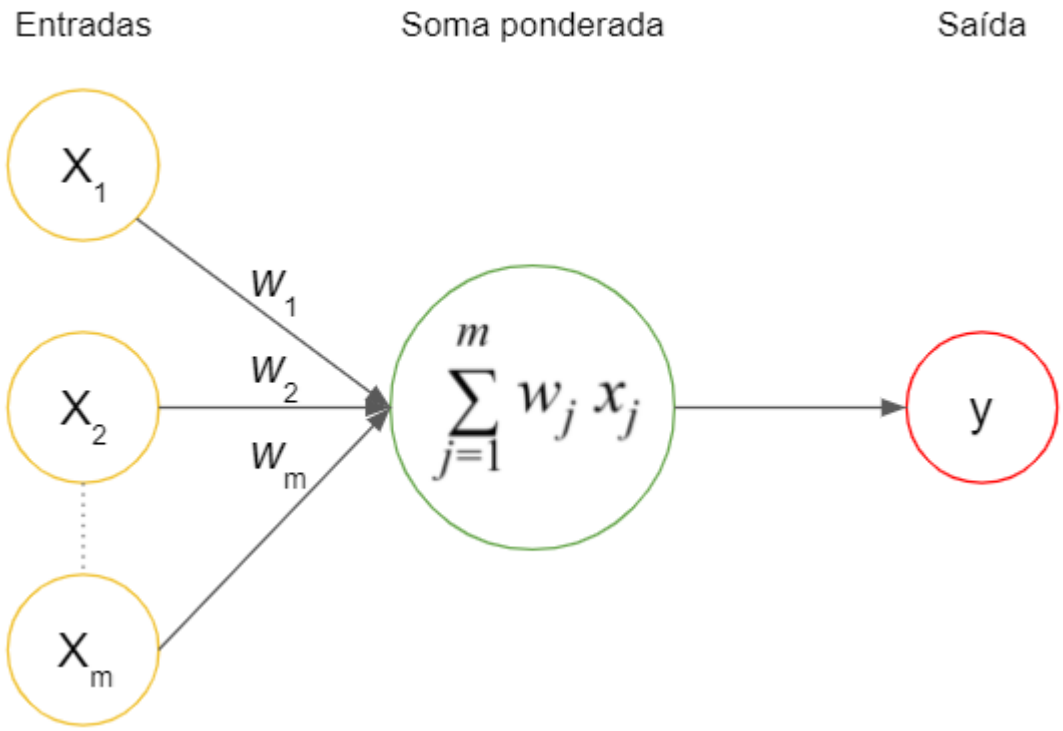


Figura 4 – Perceptron.

Fonte – autor

Não há consenso entre o número de camadas que uma rede neural deve ter para ser considerada uma rede neural profunda. Contudo, a maioria dos pesquisadores concorda que o aprendizado profundo envolve duas ou mais camadas ocultas (SUGIYAMA, 2019). Sendo assim, um perceptron não é considerado um modelo de DL. Já uma *deep feedforward network* também chamada de *feedforward neural network* ou *multilayer perceptrons* (Figura 6) é considerada DL pois possui duas ou mais camadas ocultas. Similarmente a um perceptron, a *deep feedforward network* tem como objetivo aproximar uma função f^*. Por exemplo, para um classificador $y = f^*(x)$ que mapeia uma entrada x para uma categoria y. Uma *deep feedforward network* define um mapeamento $y = f(x; w)$ e aprende o valor dos parâmetros w que resultam na melhor aproximação da função (GOODFELLOW; BENGIO; COURVILLE, 2016).

O nodo da rede neural realiza a soma dos pesos w_i e dos atributos de entrada x_i e passa o resultado da soma como parâmetro para a função de ativação ϕ. A ReLU (Figura 5) é a função de ativação padrão recomendada para uso com a maioria das *deep*

feedforward network. Aplicar essa função à saída de uma transformação linear produz uma transformação não linear (GOODFELLOW; BENGIO; COURVILLE, 2016).

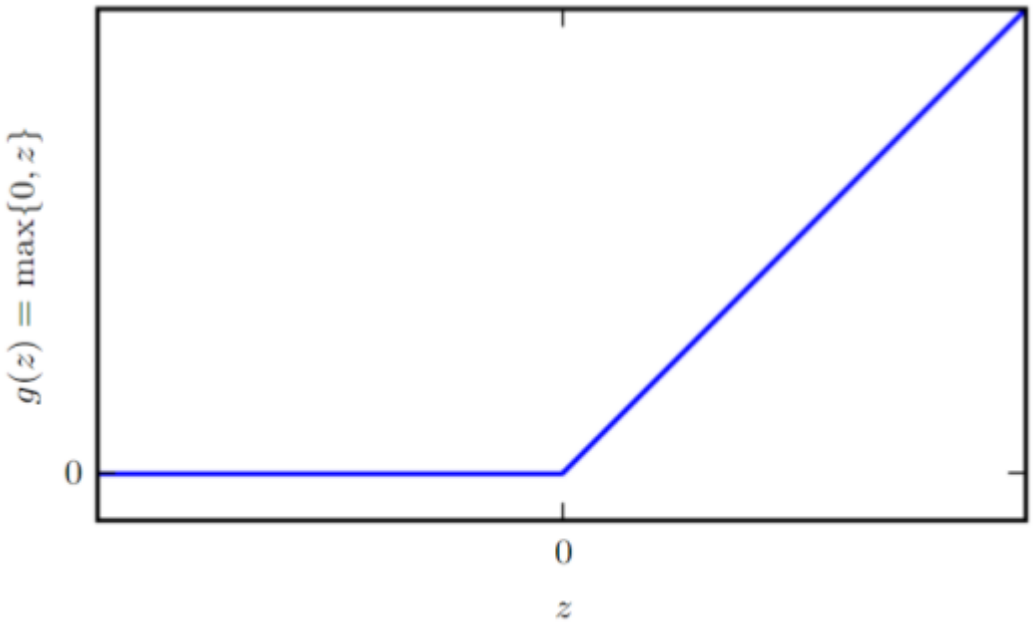


Figura 5 – Função de ativação ReLU.

Fonte – (GOODFELLOW; BENGIO; COURVILLE, 2016)

Uma *deep feedforward neural network* é o exemplo mais típico de um modelo de DL. As *deep feedforward neural network* (Figura 6) são chamadas de redes pois são normalmente representadas pela composição de muitas funções diferentes (GOODFELLOW; BENGIO; COURVILLE, 2016). O modelo está associado a um grafo acíclico direcionado que descreve como as funções são compostas juntas. Por exemplo, podemos ter três funções $f^{(1)}$, $f^{(2)}$ e $f^{(3)}$ conectadas em uma cadeia, para formar $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. Nesse caso, $f^{(1)}$ é chamada de primeira camada da rede, $f^{(2)}$ é chamada de segunda camada e assim por diante. Essas estruturas em cadeia são as estruturas mais comumente usadas de redes neurais e caracterizam a vantagem da *deep feedfoward neural network* sobre o perceptron (GOODFELLOW; BENGIO; COURVILLE, 2016).

O comprimento total da rede fornece a profundidade do modelo. O nome "aprendizagem profunda" surgiu dessa terminologia. A camada final de uma rede *feedforward* é chamada de camada de saída. Durante o treinamento da rede neural, dirigimos $f(x)$ para corresponder a $f^*(x)$. Os dados de treinamento nos fornecem exemplos aproximados e ruidosos de $f^*(x)$ avaliados em diferentes pontos de treinamento. Cada exemplo x é acompanhado por um rótulo $y \approx f^*(x)$. Os exemplos de treinamento especificam diretamente o que a camada de saída deve fazer em cada ponto x; ela deve produzir um valor que seja próximo de y. O comportamento das outras camadas não é especificado diretamente pelos dados de treinamento. O algoritmo de aprendizado deve decidir como usar essas camadas

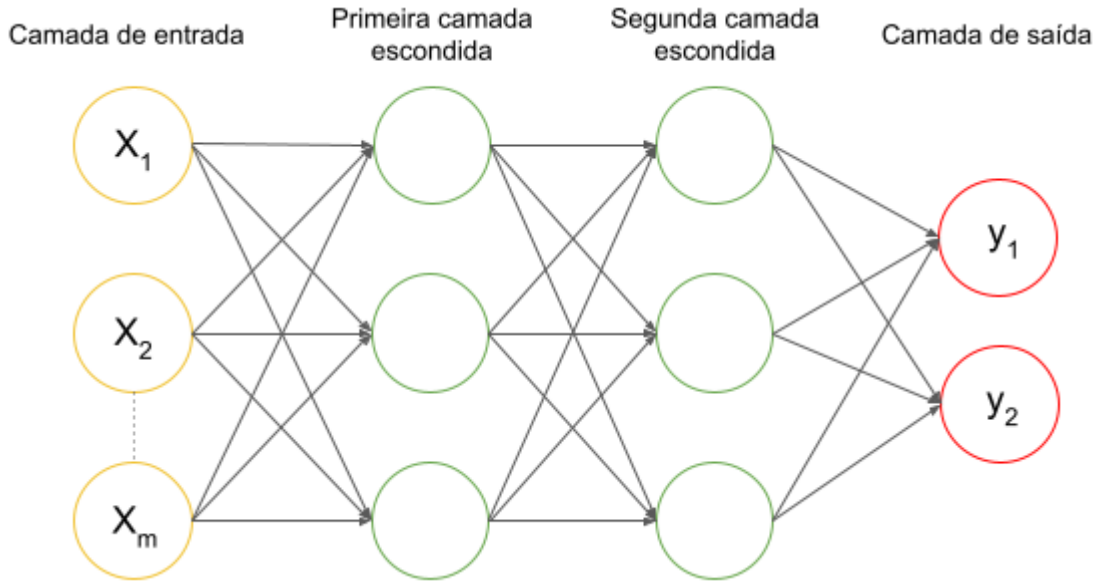


Figura 6 – Deep Feedforward Neural Network.

Fonte – autor

para produzir a saída desejada, mas os dados de treinamento não dizem o que cada camada individual deve fazer. Em vez disso, o algoritmo de aprendizagem deve decidir como usar essas camadas para melhor implementar uma aproximação de f^*. Como os dados de treinamento não mostram a saída desejada para cada uma dessas camadas, elas são chamadas de camadas ocultas (GOODFELLOW; BENGIO; COURVILLE, 2016).

Os modelos descritos acima são chamados de *feedforward* pois a informação flui através da função que está sendo avaliada de x através dos cálculos intermediários usados para definir f e, finalmente, para a saída y. Ou seja, não há conexões de *feedback* nas quais as saídas do modelo são realimentadas. Quando as redes neurais *feedforward* são estendidas para incluir conexões de *feedback*, elas são chamadas de RNN (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.4.1.1.1 Rede Neural Recorrente

As RNNs têm sido amplamente adotadas em áreas de pesquisa relacionadas com dados sequenciais, como texto, áudio e vídeo. Nas RNNs, as camadas recorrentes ou camadas ocultas consistem em células recorrentes cujos estados são afetados tanto pelos estados passados quanto pela entrada atual a partir de conexões de *feedback*. Normalmente RNNs são redes que consistem em células recorrentes padrão, como células sigma (equação 2) e células tanh (equação 3). A Figura 7 mostra um esquema da célula sigma recorrente padrão. A expressão matemática da célula sigma recorrente padrão é definida pela equação 4, onde x_t, h_t, and y_t denotam a entrada, a informação recorrente e a saída da célula no

tempo t, respectivamente; w_h e w_x são os pesos; e b é o viés (YU, Yong *et al.*, 2019).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3}$$

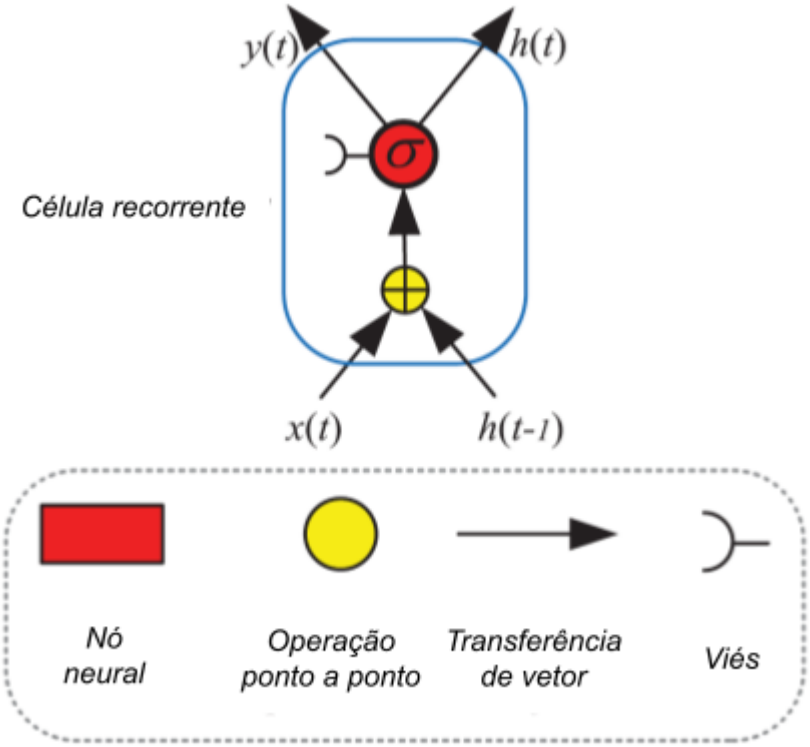


Figura 7 – Esquema da célula sigma recorrente padrão.

Fonte – adaptado de (YU, Yong *et al.*, 2019)

$$\begin{aligned} h_t &= \sigma(w_h h_{t-1} + w_x x_t + b) \\ y_t &= h_t \end{aligned} \tag{4}$$

As células recorrentes padrão obtiveram algum sucesso em alguns problemas. No entanto, RNNs que consistem em células sigma ou células tanh são incapazes de aprender as informações relevantes dos dados de entrada quando o intervalo de entrada é grande: à medida que o intervalo entre as entradas relacionadas aumenta, é difícil aprender as informações de conexão (YU, Yong *et al.*, 2019).

Para lidar com o problema das "dependências de longo prazo", Hochreiter e Schmidhuber (1997) propuseram a célula LSTM. Desda sua introdução, quase todos os resultados empolgantes baseados em RNNs foram alcançados pela LSTM. A LSTM se tornou

o foco do DL. A capacidade de memorização da célula recorrente padrão foi aumentada ao introduzir um "portão" na célula. Desde este trabalho pioneiro, as LSTMs foram modificadas e popularizadas por muitos pesquisadores. As variações incluem LSTM sem um portão de esquecimento, LSTM com um portão de esquecimento e LSTM com uma conexão de olho mágico (YU, Yong *et al.*, 2019). Em seguida, apresentamos o modelo LSTM com um portão de esquecimento.

A Figura 8 apresenta as conexões internas de uma LSTM com portão de esquecimento. Com base nas conexões mostradas na Figura 8, a célula LSTM pode ser expressa matematicamente pela equação 5, onde c_t denota o estado da LSTM, w_i, $w_{\tilde{c}}$, w_o, w_f são os pesos do portão de entrada, da célula de memória, do portão de saída e do portão de esquecimento respectivamente, e o operador "$\cdot$"denota a multiplicação ponto a ponto de dois vetores. Ao atualizar o estado da célula, o portão de entrada pode decidir quais novas informações podem ser armazenadas no estado da célula, o portão de saída decide quais informações podem ser enviadas com base no estado da célula e o portão de esquecimento pode decidir quais informações serão descartadas do estado da célula. Quando o valor do portão de esquecimento, f_t, é 1, ele mantém essa informação, por outro lado, um valor de 0 significa que o portão se livra de todas as informações (YU, Yong *et al.*, 2019).

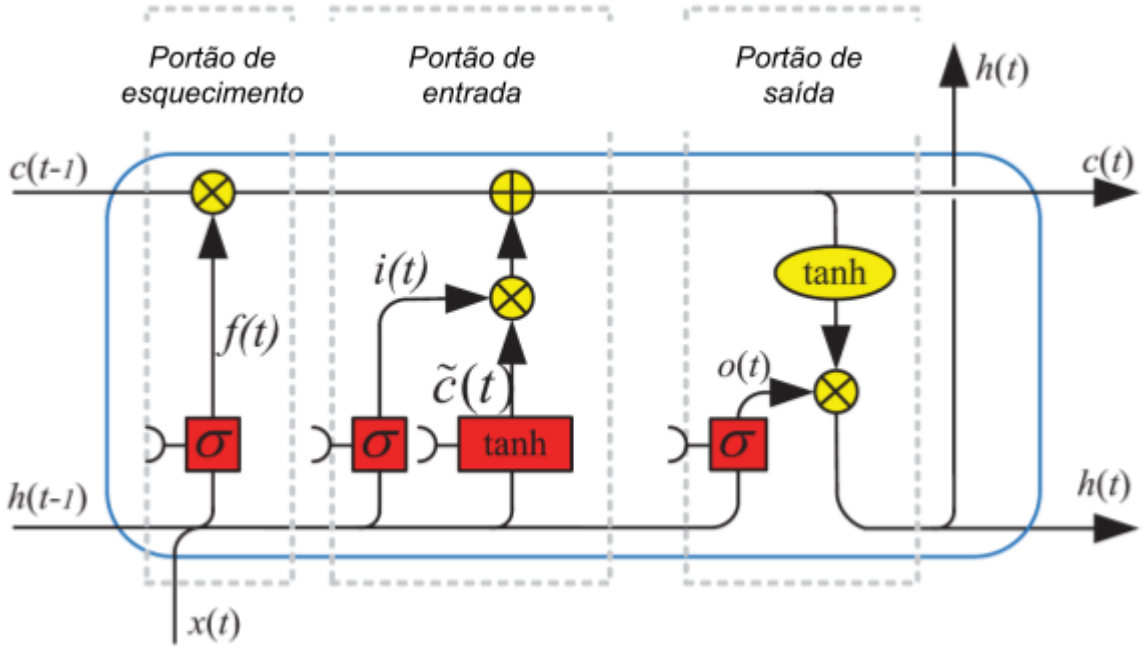


Figura 8 – Arquitetura da LSTM com um portão de esquecimento..

Fonte – adaptado de (YU, Yong *et al.*, 2019)

$$
\begin{aligned}
f_t &= \sigma(w_{fh}h_{t-1} + w_{fx}x_t + b_f) \\
i_t &= \sigma(w_{ih}h_{t-1} + w_{ix}x_t + b_i) \\
\tilde{c}_t &= tanh(w_{\tilde{c}h}h_{t-1} + w_{\tilde{c}x}x_t + b_{\tilde{c})} \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \\
o_t &= \sigma(w_{oh}h_{t-1} + w_{ox}x_t + b_o) \\
h_t &= o_t \cdot tanh(c_t)
\end{aligned}
\tag{5}
$$

Em suma, este capítulo apresentou os principais conceitos relacionados ao domínio da classificação de litofácies, assim como os conceitos sobre algoritmos de ML. A seguir, tais conceitos serão utilizados para a realização da revisão sistemática da literatura sobre classificação de litofácies utilizando algoritmos de ML.

3 REVISÃO SISTEMÁTICA DA LITERATURA

Neste capítulo são apresentados quatro artigos que aplicaram técnicas para a classificação de litofácies através de modelos de ML. Os trabalhos foram selecionados a fim de expandir a visão sobre o que vem sendo aplicado nessa área do conhecimento. Ao final do capítulo é realizada a comparação entre os trabalhos.

A busca por artigos foi realizada no Google Scholar utilizando as palavras chaves *facies*, *classification*, *well logs*, e *machine learning*. A pesquisa retornou 1.120 artigos. Procuramos por artigos que possivelmente utilizavam técnicas de ML e realizavam a classificação de litofácies. Através de uma leitura dos resumos de 38 desses artigos, observando se eles realmente usavam técnicas de ML para a classificação e realmente classificavam litofácies, foram selecionados 17 artigos para uma leitura mais profunda. Por fim, foram selecionados 4 artigos para serem apresentados a seguir. Tais artigos apresentavam maior similaridade com a proposta do presente trabalho, assim como gráficos de comparação com outros modelos e uma boa explicação da arquitetura da rede neural utilizada.

3.1 LITHOLOGICAL FACIES CLASSIFICATION USING DEEP CONVOLUTIONAL NEURAL NETWORK

No trabalho apresentado por (IMAMVERDIYEV; SUKHOSTAT, 2019) uma arquitetura baseada em CNN unidimensional (1D-CNN), que é treinada usando vários algoritmos de optimização, é proposta para a classificação de litofácies. Os algoritmos de optimização testados foram o Adagrad, Adadelta e Adamax.

A arquitetura do modelo 1D-CNN consiste em uma camada de entrada, quatro camadas convolucionais com ReLU como função de ativação não linear, duas camadas *maxpool* e três camadas totalmente conectadas. Uma camada *maxpool* aplica, para cada uma das regiões representadas pelo filtro convolucional, o máximo dessa região e cria uma nova matriz de saída onde cada elemento é o máximo de uma região na entrada original (WIKI, s.d.). A última camada é a camada de saída, que atribui a categoria aos dados de entrada (Figura 9). Os dados de entrada foram divididos em conjuntos de treinamento e validação (20% do conjunto de dados) para conduzir os experimentos. O tamanho do lote foi determinado como 10 e a função de perda foi escolhida como a entropia cruzada categórica. Cada resultado experimental foi obtido ao longo de 4000 épocas para fornecer comparações consistentes.

No conjunto de dados considerado, existem dados de 10 poços contendo um total de 4149 amostras. O conjunto de dados contém 9 tipos de litofácies: arenito não marinho (NS), siltito grosso não marinho (NCS), siltito fino não marinho (NFS), siltito marinho e xisto (MSS), argilito (M), wackestone (W), dolomita (D), packstone-grainstone (PG), e calcário (PAB). Os 6 atributos do conjunto de dados são: efeito fotoelétrico, raio gama, resistividade, diferença de porosidade de neutrondensidade, porosidade de densidade média

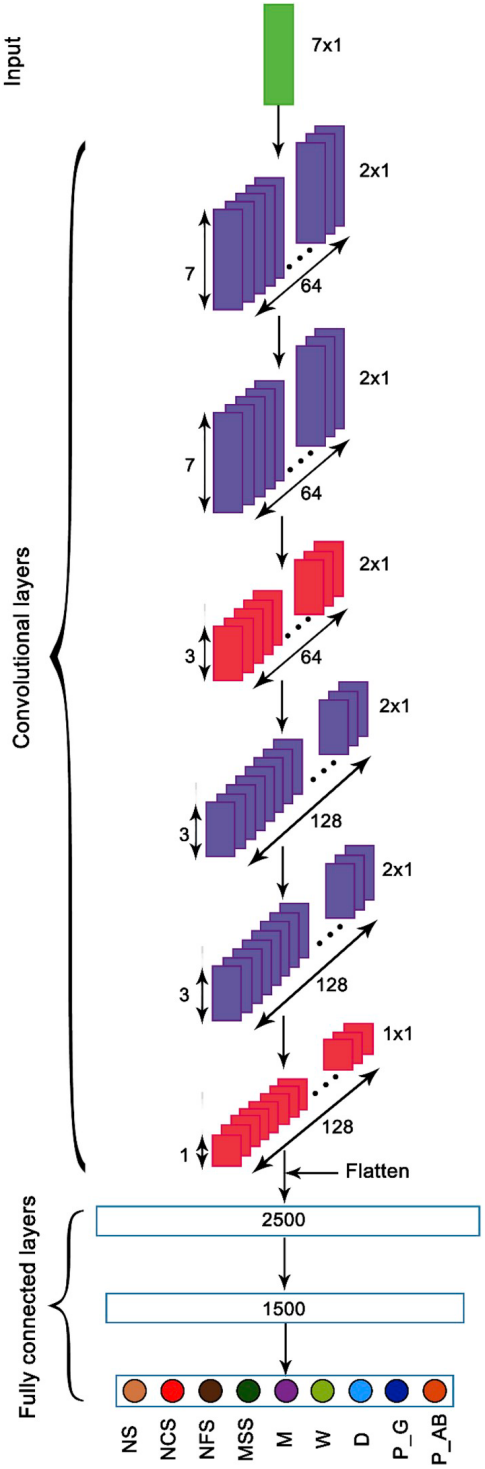


Figura 9 – A arquitetura geral da 1D-CNN.

Fonte – (IMAMVERDIYEV; SUKHOSTAT, 2019)

de nêutrons, e variáveis de restrição geológica.

Uma análise comparativa do modelo proposto usando otimizadores Adagrad, Adadelta e Adamax com CNN, LSTM, máquina de vetores de suporte (do inglês SVM), e k-vizinhos mais próximos (do ingles KNN) com base em acurácia e métricas de *F-score*

são mostradas na Figura 10.

Method	Evaluation metrics	Facies									
		NS	NCS	NFS	MSS	M	W	D	P_G	P_AB	Total
RNN	Accuracy (%)	75.86	65.00	62.05	58.70	18.18	42.50	46.15	49.72	80.00	56.39
	F-measure (%)	53.01	67.29	64.78	55.10	10.53	43.97	43.64	57.69	52.46	55.22
LSTM	Accuracy (%)	81.25	69.36	68.92	50.85	44.00	50.39	72.73	58.13	83.87	63.13
	F-measure (%)	60.47	74.09	68.00	54.05	27.85	53.94	40.00	63.92	72.22	62.22
SVM	Accuracy (%)	78.00	75.23	76.06	73.47	58.82	62.07	83.33	76.34	**88.89**	73.73
	F-measure (%)	75.00	78.22	73.47	71.29	57.14	63.16	75.47	76.34	**93.02**	73.64
KNN	Accuracy (%)	73.08	74.31	74.66	64.62	56.25	58.87	83.33	77.27	76.74	71.33
	F-measure (%)	71.70	76.60	73.15	71.19	52.94	61.86	75.47	70.83	78.57	71.32
1D-CNN (Adagrad)	Accuracy (%)	**84.09**	**77.63**	**76.82**	**74.55**	**70.69**	**67.89**	**86.67**	**78.15**	**88.89**	**76.87**
	F-measure (%)	75.51	**80.38**	**76.57**	71.93	**70.69**	**67.58**	**80.00**	**78.81**	**93.02**	**76.78**
1D-CNN (Adadelta)	Accuracy (%)	79.59	75.77	76.47	**74.55**	61.36	62.07	84.62	77.50	85.97	74.58
	F-measure (%)	75.00	78.54	74.02	**77.36**	55.10	64.87	**80.00**	75.30	89.91	74.44
1D-CNN (Adamax)	Accuracy (%)	78.57	75.23	76.43	73.68	58.62	61.39	84.62	76.61	81.25	73.37
	F-measure (%)	**77.19**	78.10	73.29	75.00	59.13	58.77	77.19	76.92	88.64	73.20

Figura 10 – Comparação dos resultados da classificação de fácies usando a abordagem proposta com RNN, LSTM, SVM e KNN.

Fonte – (IMAMVERDIYEV; SUKHOSTAT, 2019)

3.2 FACIESNET: MACHINE LEARNING APPLICATIONS FOR FACIES CLASSIFICATION IN WELL LOGS

Nesse trabalho apresentado por (JAIKLA *et al.*, 2019) é desenvolvido um modelo de classificação de fácies usando redes neurais recorrentes bidirecionais (do inglês Bidirectional Recurrent Neural Network (BRNN)) que incorporam sequências de fácies na previsão. Além de BRNNs, experimentou-se outra arquitetura adicionando camadas de decodificação e codificação de redes neurais convolucionais profundas (do inglês Deep Convolutional Neural Network (DCNN)) para extrair informações latentes antes de alimentá-las nas camadas de BRNNs.

Para a arquitetura de BRNN os experimentos incluíram o treinamento do conjunto de dados em modelos com 1, 2 e 3 camadas de BRNNs com 16, 32, 64 e 128 estados ocultos de unidades recorrentes bloqueadas (do inglês Gated Recurrent Unit (GRU)).

Já para a arquitetura com DCNNs e BRNN, a arquitetura que tem a maior acurácia e acurácia equilibrada no conjunto de teste consiste em 5 camadas de codificação e decodificação DCNNs seguidas por 2 camadas de BRNNs com 128 estados ocultos usando a *dice loss function*. Tal arquitetura foi chamada de FaciesNet (Figura 11).

O conjunto de dados possui 4 poços contendo um total de 170 amostras. O conjunto de dados contém 5 tipos de litofácies: arenito cimentado, heterolítico, lamito, arenito limpo, e arenito sujo. Os 6 atributos do conjunto de dados são: GR, fração de volume de xisto (VSH), densidade (DEN), DTC, DTS, e porosidade de nêutrons (NEU).

Foi realizado uma análise comparandos os modelos de Naïve Bayes, árvore de decisão, floresta aleatória, BRNN e FaciesNet, com base em acurácia e acurácia equilibrada

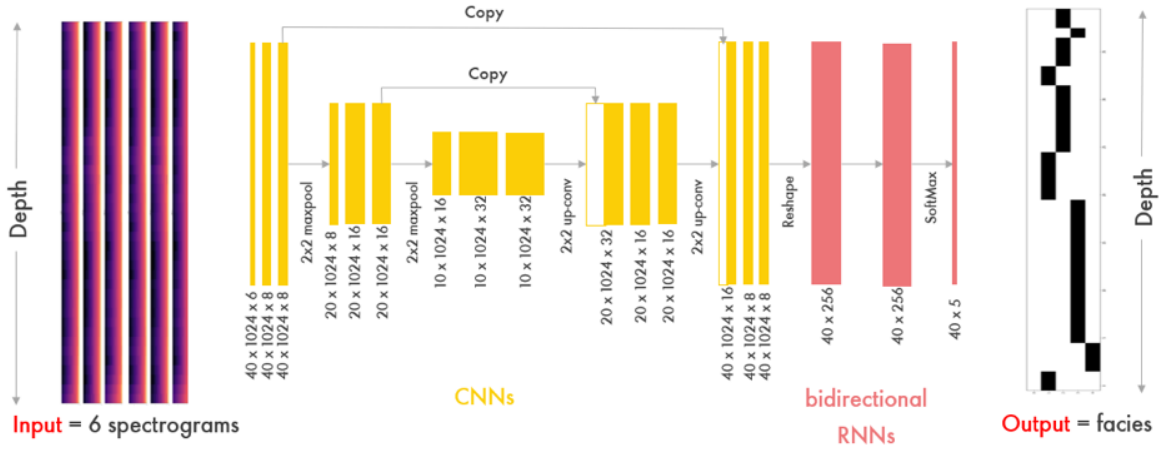


Figura 11 – Arquitetura FaciesNet.

Fonte – (JAIKLA *et al.*, 2019)

(Figura 12). Além disso, uma análise comparativa entre Naïve Bayes e FaciesNet, com base em precisão, *recall* e *F1-score* são mostradas na Figura 13.

Model	Accuracy	Balanced accuracy
Naive Bayes	83.45%	56.97%
Decision Tree	83.69%	51.55%
Random Forest	84.88%	51.21%
BRNNs	64.11%	24.43%
FaciesNet	74.85%	40.01%

Figura 12 – Acurácia e acurácia balanceada da rede.

Fonte – (JAIKLA *et al.*, 2019)

3.3 CHARACTERIZING ROCK FACIES USING MACHINE LEARNING ALGORITHM BASED ON A CONVOLUTIONAL NEURAL NETWORK AND DATA PADDING STRATEGY

No trabalho apresentado por (WEI *et al.*, 2019) é proposto uma arquitetura usando CNN com estratégias de preenchimento de dados. Inspirados pelo uso de CNN em imagens multicanal, foi testado três estratégias de preenchimento para expandir os conjuntos de dados bem medidos 1-D para 2-D para melhor capturar seus recursos inerentes.

O modelo possui duas camadas convolucionais. A primeira camada convolucional tem oito filtros 3x3 e a segunda dezesseis camadas 3x3. Cada camada convolucional é seguida por uma função de ativação de ReLU e uma camada de *maxpool*. Finalmente, a saída é conectada por uma camada totalmente conectada (Figura 14). Os três tipos de

FaciesNet	Precision	Recall	F1 score
Cemented sandstone	0.8125	0.2718	0.4063
Heterolithic	0.1895	0.2000	0.1956
Mudstone	0.6209	0.5125	0.5621
Sandstone	0.8485	0.9133	0.8797
Dirty sandstone	0.1320	0.1029	0.1157
Naive Bayes	Precision	Recall	F1 score
Cemented sandstone	0.8913	0.8542	0.8723
Heterolithic	0	0	0
Mudstone	0.5745	1	0.7298
Sandstone	0.9315	0.9401	0.9358
Dirty sandstone	0	0	0

Figura 13 – Comparação de precisão, *recall*, *F1-score*, da FaciesNet com Naïve Bayes.

Fonte – (JAIKLA *et al.*, 2019)

estratégias de preenchimento de dados usados foram: preenchimento igual, preenchimento de descolamento e preenchimento aleatório.

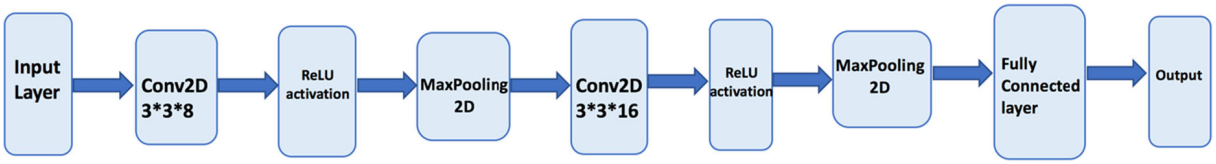


Figura 14 – Arquitetura da CNN.

Fonte – (WEI *et al.*, 2019)

No conjunto de dados, existem dados de 8 poços contendo um total de 4149 amostras. O dados possuem 9 tipos de litofácies: arenito não marinho (SS), siltito grosso não marinho (CSiS), siltito fino não marinho (FSiS), siltito marinho e xisto (SiSh), lamito (calcário) (MS), wackestone (calcário) (WS), D, packstone grainstone (calcário) (PS), e bafflestone filoidalgal (calcário) (BS). Os 5 atributos dos dados são: GR, média de porosidade de nêutron e densidade (PHI), porosidade de nêutron e diferença de porosidade de densidade (DeltaPHI), efeito fotoelétrico (PE), e base de registro de resistividade verdadeira aparente (ILDlog10).

Os resultados de precisão das três estratégias diferentes de preenchimento são apresentados na Figura 15. Para comparação, a Figura 16 compara diferentes algoritmos sobre o mesmo conjunto de dados.

	No. of filters of first convolutional layer	No. of filters of second convolutional layer	After 100 Epochs train accuracy (%)	Test set accuracy (%)
Equal padding	8	16	58.62	52.46
Shift padding	8	16	61.87	59.26
Random padding	8	16	40	22.92

Figura 15 – Resultados das três diferentes estratégias de preenchimento, onde a acurácia é dada pelo *F1-score*.

Fonte – (WEI *et al.*, 2019)

Team	F1 score	Algorithm	Language
LA Team (Mosser, de la Fuente)	0.641	Boosted trees	Python
Ispl (Bestagini, Tuparo, Lipari)	0.640	Boosted trees	Python
…	…	…	…
ShiangYong	0.570	ConvNet	Python
StoDIG	0.561	ConvNet	Python
…	…	…	…
BrendonHall	0.427	Support vector machine	Python

Figura 16 – Resultados da competição SEG ML 2016 (https://github.com/seg/2016-ml-contest).

Fonte – (WEI *et al.*, 2019)

3.4 COMPARISON OF DIFFERENT MACHINE LEARNING ALGORITHMS FOR LITHOFACIES CLASSIFICATION FROM WELL LOGS

Nesse trabalho apresentado por (DELL'AVERSANA, 2019) é realizado a comparação de 6 algoritmos de aprendizagem supervisionada. Os algoritmos utilizados foram: indução de regra CN2, Naïve Bayes, SVM, árvore de decisão, floresta aleatória, e impulso adaptativo.

O fluxo de trabalho incluiu as seguintes etapas principais: 1) análise de dados estatísticos; 2) treinamento de 6 algoritmos de classificação; 3) avaliação quantitativa do desempenho de cada algoritmo individual; 4) classificação simultânea de litofácies usando todos os 6 algoritmos; 5) comparação e relatórios de resultados.

O conjunto de dados possui 2 poços (poço A e poço B) contendo aproximadamente 21000 amostras cada poço. O conjunto de dados contém 5 tipos de litofácies: xisto prevalente, arenitos intercalados / xisto, arenitos intercalados / siltito, hidrocarboneto de média saturação, hidrocarboneto de baixa saturação, e hidrocarboneto de alta saturação. Os 6 atributos do conjunto de dados são: sônico, RDEP, DEN, NEU, PEF, GR, e SP.

Para avaliar a performance dos 6 algoritmos, foi utilizado as métricas: área abaixo da curva (do inglês Area Under the ROC Curve (AUC)), precisão de classificação (do inglês Classification Accuracy (CA)), *F1-score*, precisão, e *recall*. A Figura 17 mostra os

resultados dos diferentes modelos para o poço A, e a Figura 18 para o poço B.

METHOD	AUC	CA	F1	PRECISION	RECALL
Tree	0.971	0.943	0.943	0.944	0.943
Random Forest	1.000	0.990	0.990	0.990	0.990
CN2 Inducer	0.955	0.867	0.868	0.869	0.867
AdaBoost	0.979	0.965	0.965	0.965	0.965
SVM	0.988	0.981	0.980	0.981	0.981
Naive Bayes	0.998	0.956	0.956	0.958	0.956

Figura 17 – Resultados da avaliação do desempenho dos modelos para o poço A.

Fonte – (DELL'AVERSANA, 2019)

METHOD	AUC	CA	F1	PRECISION	RECALL
Tree	0.928	0.848	0.947	0.848	0.848
Random Forest	0.992	0.917	0.918	0.919	0.917
CN2 Inducer	0.909	0.767	0.768	0.769	0.767
AdaBoost	0.914	0.859	0.859	0.859	0.859
SVM	0.944	0.744	0.744	0.744	0.744
Naive Bayes	0.928	0.959	0.959	0.959	0.959

Figura 18 – Resultados da avaliação do desempenho dos modelos para o poço B.

Fonte – (DELL'AVERSANA, 2019)

3.5 COMPARAÇÃO ENTRE OS TRABALHOS

Os resultados de (IMAMVERDIYEV; SUKHOSTAT, 2019) mostraram que as melhores previsões de resposta da rede foram obtidas para o caso Adagrad com um *F1-score* total de 76.78. Além disso, o modelo 1D-CNN mostrou resultados mais precisos em comparação com SVM, KNN, RNN e LSTM. O modelo proposto também superou o SVM em mais de 50% dos resultados. O pior desempenho é observado para RNN. A aplicação da abordagem proposta para a classificação de fácies mostrou resultados significativos para fácies de origem marinha (dolomita, bafflestone de algas filoides e packstone-grainstone) e de origem continental (siltito grosso e arenito). O modelo 1D-CNN (Adagrad) proposto apresentou uma melhora estatisticamente significativa na classificação de fácies.

Embora o modelo proposto por (JAIKLA *et al.*, 2019) tenha menor acurácia e acurácia equilibrada do que as abordagens de outros estudos, o FaciesNet pode diferenciar

entre fácies reservatório e não reservatório, que são arenito limpo e arenito sujo, bem como argilito e heterolítico. Além disso, ele dá previsões geológicas significativas e não sofre ao usar dados heterogêneos e desequilibrados. A arquitetura BRNN de maior precisão de teste de 64,11% foi a de 3 camadas de BRNN com 128 estados ocultos para cada camada. Entretanto, a arquitetura que teve a maior acurácia e acurácia equilibrada no conjunto de teste foi a FaciesNet consistindo em 5 camadas de codificação e decodificação DCNNs seguidas por 2 camadas de BRNNs com 128 estados ocultos usando a função de perda de dados. A acurácia da FaciesNet foi de 74,85%, e a precisão balanceada foi de 40,01%.

No trabalho de (WEI *et al.*, 2019) entre todos os resultados de classificação, o método de preenchimento de deslocamento igual atingiu 59,26% na *F1-score*, melhor do que os melhores resultados da CNN de 57% entre todos os concursos SEG ML 2016. Ao preencher os dados, a matriz de recursos 2D é útil para algoritmos CNN para detectar conexões entre os recursos e capturar as diferenças sutis entre fácies no processo de classificação. As estratégias de preenchimento adicionam dimensões aos dados, o que significa adicionar mais liberdade ao conjunto de dados e capturar melhor a correlação entre seus diferentes recursos. No entanto, o resultado final ainda não bate os melhores resultados alcançados pelo algoritmo de árvore impulsionada.

Por fim, no trabalho de (DELL'AVERSANA, 2019), os algoritmos de conjunto como floresta aleatória e impulso adaptativo parecem fornecer classificações/previsões ligeiramente mais confiáveis que Naïve Bayes, árvore de decisão, indução de regra CN2. O método SVM também demonstrou bom desempenho. No poço A a floresta aleatória atingiu o melhor resultado com 0.990 considerando o *F1-score*, enquanto a indução de regra CN2 atingiu o pior resultado com 0.868 de *F1-score*. Já no poço B o Naïve Bayes atingiu o melhor resultado com 0.959 de *F1-score*, por outro lado, o pior resultado foi atingido pela SVM com *F1-score* de 0.744.

Três dos quatro trabalhos apresentados constroem seus modelos tendo como base arquiteturas de CNN. Em relação a RNN e LSTM, o trabalho de (IMAMVERDIYEV; SUKHOSTAT, 2019) mostrou que os piores resultados foram obtidos pela RNN e LSTM. Além disso, as arquiteturas mostradas por (JAIKLA *et al.*, 2019), reafirmam a hipótese de que incrementar uma RNN com uma CNN (culminando em uma FaciesNet) fez com que a acurácia passasse de 64.11% para 74.85% e a acurácia balanceada passasse de 24.43% para 40.01%. O trabalho de (WEI *et al.*, 2019) apresenta que o modelo com maior sucesso na competição SEG ML 2016 foram árvores impulsionadas. As CNNs não ficaram entre as 10 primeiras colocadas, e não há nenhuma citação de experimento com RNN no quadro de resultados da competição. Por fim, o trabalho de (DELL'AVERSANA, 2019) fez uma apresentação de modelos que não utilizam redes neurais. Tais modelos obtiveram os melhores resultados de *F1-score*. Entretanto, como os trabalhos apresentados possuem conjuntos de dados diferentes, não é possível comparar os resultados obtidos por modelos em um determinado conjunto de dados com resultados obtidos por modelos utilizando um

outro conjunto de dados. Nenhum dos trabalhos correlatos utilizou o mesmo conjunto de dados utilizado no desenvolvimento do presente trabalho.

Notamos, a partir dos trabalhos apresentados, que as RNNs são modelos que podem ser utilizados para a classificação de litofácies. Entretanto, os resultados das RNNs foram inferiores as demais arquiteturas propostas. O modelo proposto utilizará conceitos similares aos apresentados por (JAIKLA *et al.*, 2019) isto é, um tipo de RNN inspirado na ideia de levar em consideração sequências de camadas. Entretanto, utilizaremos um tipo diferente de RNN, no caso uma LSTM, assim como um diferente conjunto de dados. Portanto, realizaremos a verificação de desempenho de mais um tipo de RNN em relação as arquiteturas propostas na Force 2020 Machine Learning Competition, validando ou não sua ineficácia na classificação de litofácies sobre o conjunto de dados da competição.

4 DESENVOLVIMENTO

Neste capítulo, explicamos como os conceitos teóricos apresentados na seção de fundamentação teórica são colocados em prática na elaboração do método proposto[1]. Primeiramente demonstramos o ambiente utilizado na implementação, tais como linguagem e bibliotecas. Em seguida, apresentamos o conjunto de dados utilizados como entrada para o modelo proposto. Por fim, mostramos as métricas utilizadas para a avaliação dos modelos produzidos.

4.1 AMBIENTE

A implementação foi desenvolvida em Python, utilizando como IDE o Jupyter Notebook. As principais bibliotecas utilizadas foram *numpy* e *pandas* para a manipulação e visualização dos dados; *sklearn* para o pré-processamento dos dados; *keras* para a construção da rede neural; e *matplotlib* para a construção de gráficos.

4.2 CONJUNTO DE DADOS

Neste trabalho usaremos o conjunto de dados fornecido pela competição Force 2020 Machine Learning Competition. Os dados são provenientes da costa da Noruega. O conjunto de dados é separado em conjunto de treino, teste, e validação. O conjunto de treino foi liberado no início da competição. O conjunto de teste era utilizado para computar o placar dos participantes da competição durante a competição. Para tal, os participantes submetiam suas predições sobre o conjunto de teste, para a competição, e a competição retornava a pontuação do participante. Por fim, o conjunto de validação foi liberado apenas após o final da competição, para que a competição pudesse determinar o vencedor.

O conjunto de dados consiste em um total de 1.429.694 exemplos de litofácies, sendo que 1.170.511 são pertencentes ao conjunto de treino, 136.786 ao conjunto de teste, e 122.397 ao conjunto de validação. Há um total de 118 perfis de poços, 98 para o conjunto de treino, 10 para o conjunto de teste, e 10 para o conjunto de validação. O conjunto de dados de treino e de validação possuem 27 atributos de registro de poços, além de um atributo de confiança de interpretação, e outro atributo contendo a classe das litofácies, totalizando 29 atributos. Tais atributos foram apresentados na Seção 2.2. Visto que o conjunto de teste não possui a confiança da interpretação nem a classe das litofácies, ele possui apenas 27 atributos. Portanto, dado que o conjunto de teste não possui as classes para as litofácies e que a competição já se encerrou, tal conjunto de dados não será utilizado no presente trabalho.

[1] O código produzido pode ser encontrado em https://github.com/MatheusSchaly/TCC assim como no apêndice A

O conjunto de dados possui vários atributos com inúmeros valores ausentes. Por exemplo, o atributo SGR, que é o atributo que possui a maior quantidade de valores ausentes, possui 1.101.158 valores ausentes no conjunto de treino e 122.397 valores ausentes no conjunto de validação. Representando uma ausência de 94.07% de dados no conjunto de treino e 89.48% de dados no conjunto de validação. Considerando os 27 atributos do conjunto de dados de treino e de validação, temos um total de 31.603.797 e 3.304.719 valores respectivamente. Desse total, 10.245.502 são valores ausentes no conjunto de treino e 995.942 são valores ausentes no conjunto de validação. Portanto, 32,42% dos valores são valores ausentes no conjunto de treino e 30,14% dos valores são valores ausentes no conjunto de validação. Os atributos ausentes do conjunto de treino podem ser vistos abaixo.

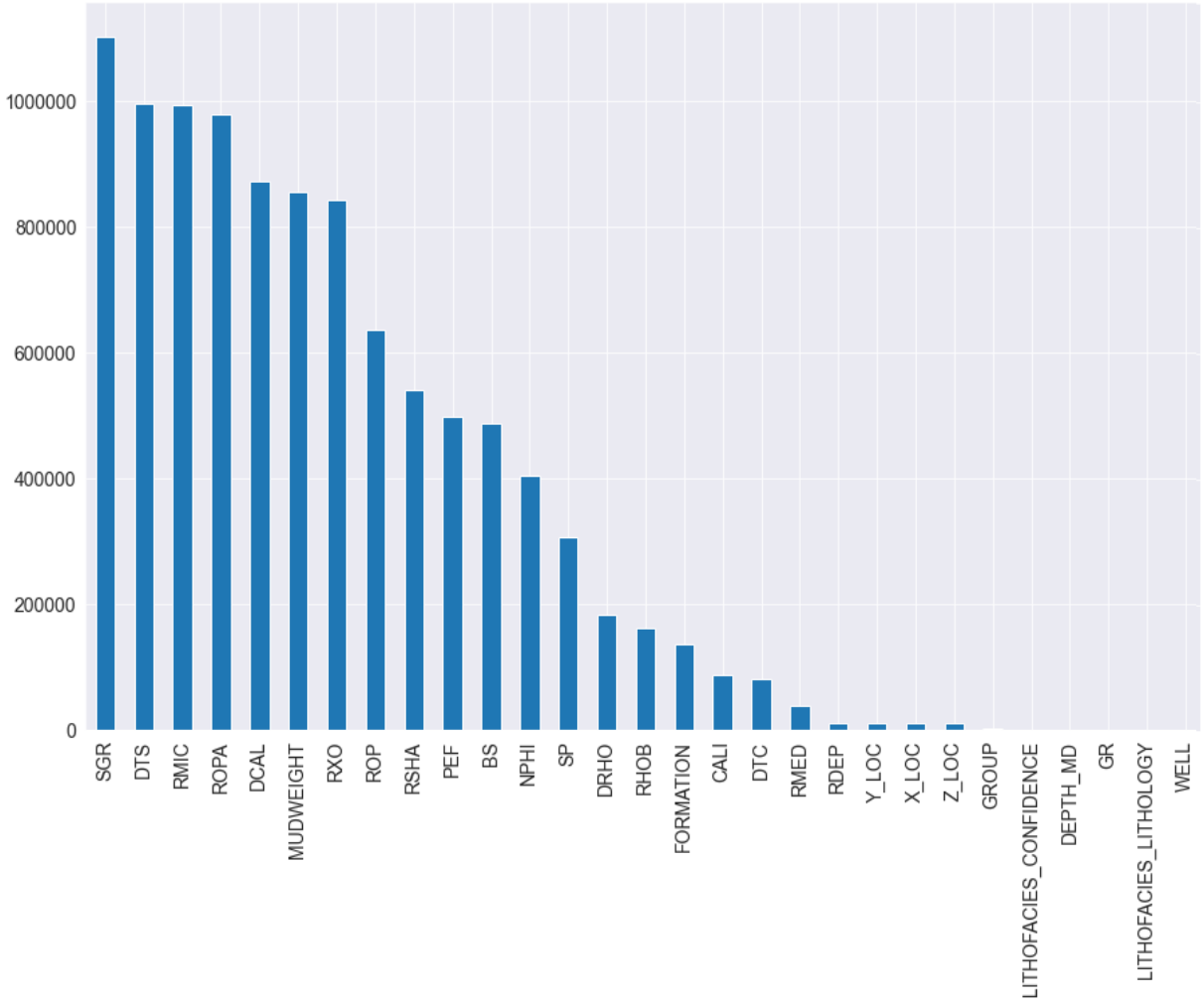


Figura 19 – Valores ausentes no conjunto de treino.

Fonte – autor

Temos também um grande desbalanceamento de classes (Figura 21). Considerando o conjunto de treino, temos que a classe xisto representa a 61.5802% das classes. No outro extremo, temos que a classe embasamento representando apenas 0.0088% das classes.

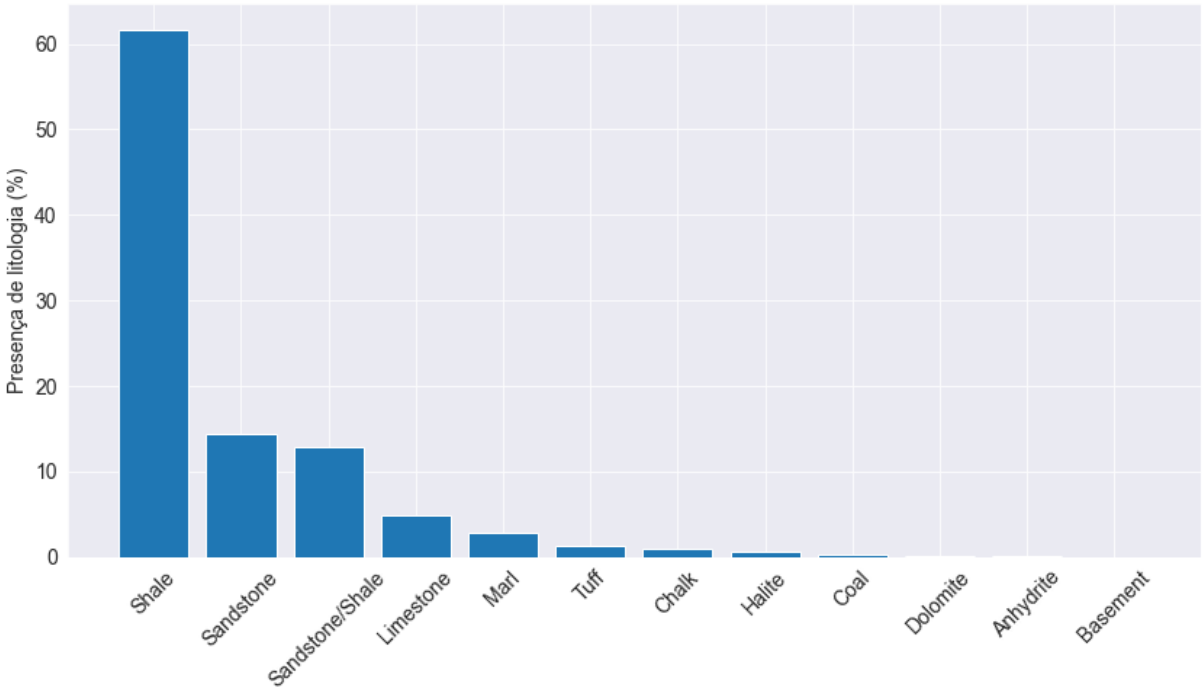


Figura 20 – Desbalanceamento de classes no conjunto de treino.

Fonte – autor

4.3 PRÉ-PROCESSAMENTO DOS DADOS

As etapas do pré-processamento dos dados (Figura 21) foram:

1. Carregamento dos dados: fazemos o carregamento dos dados de treinamento da competição em um pandas *dataframe*.

2. Seleção dos atributos: aplicamos diferentes métodos de seleção de atributos para reduzir o efeito do *curse of dimensionality*, consequentemente diminuindo o risco de *overfitting*, aumentando a velocidade do método e sua acurácia. Utilizamos os seguintes métodos de seleção de atributos:

   - Métodos de seleção de atributos limite de variância, teste qui-quadrado, limiar de correlação, *backward elimination*, eliminação de recurso recursivo, e LassoCV.
   - Quantidade de valores faltantes nos atributos.
   - Comparação com a seleção dos atributos realizados pelos top 5 participantes da competição. Os atributos descartados foram: SGR, DTS, ROPA, DCAL, medição de resistividade de leitura micro (RMIC), ROP, RXO, FORCE_2020_LITHOFACIES_CONFIDENCE, FORMATION e MUDWEIGHT. Com isso ficamos com 19 atributos.

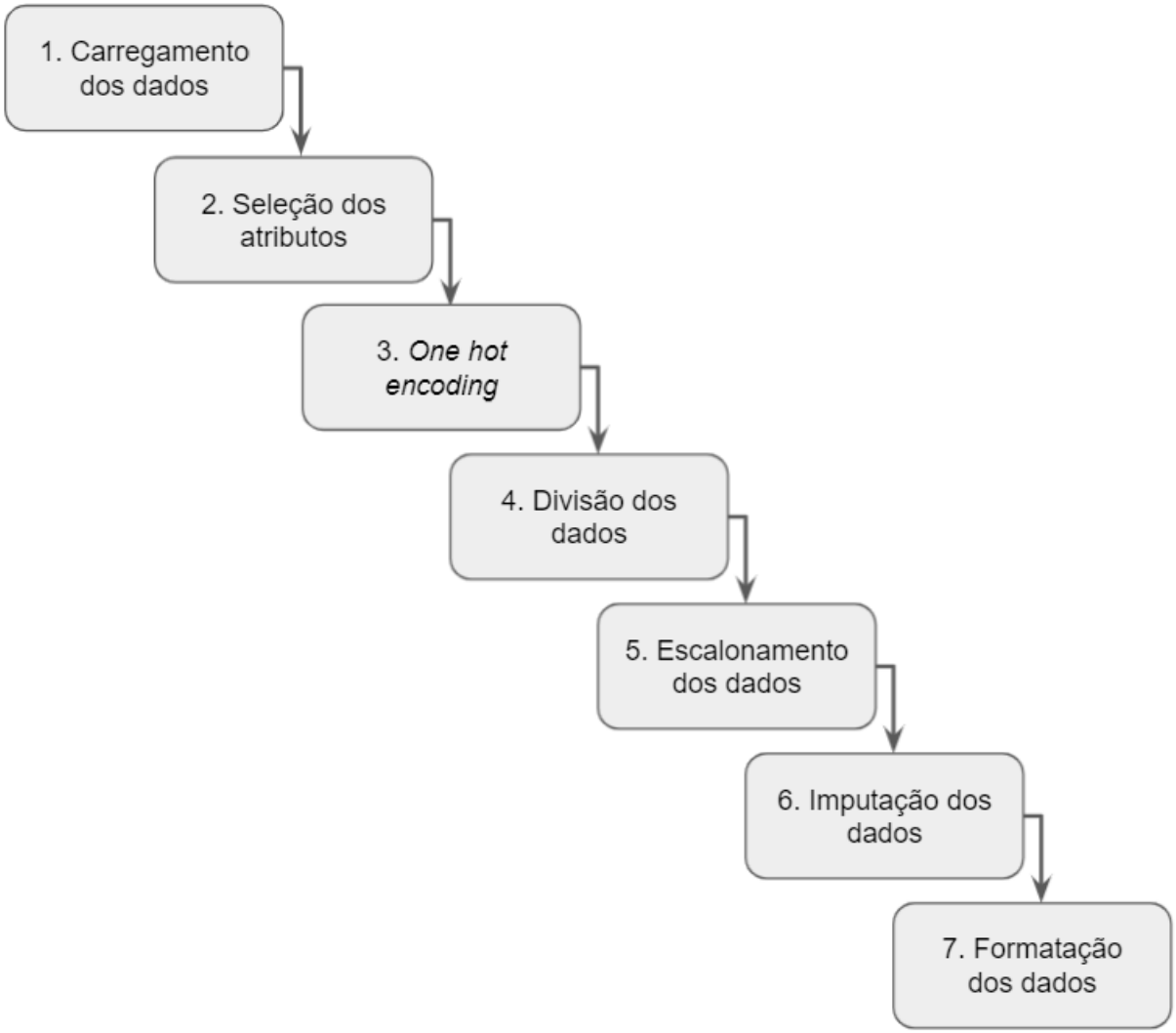


Figura 21 – Etapas de pré-processamento dos dados.

Fonte – autor

3. *One hot encoding*: dado que o modelo espera atributos numéricos, usamos o *one hot encoding* para converter as colunas categóricas em colunas numéricas. O processo de *one hot encoding* consiste em criar um atributo próprio para cada um dos valores distintos do atributo categórico e excluir o atributo categórico. Estas novas colunas terão, em suas linhas, um único valor 1 no atributo correspondente ao atributo categórico anterior e o restante dos valores é 0. Aplicamos o *one hot encoding* no atributo GROUP, o qual possuía 13 valores distintos, com isso passamos de 19 atributos para 31 atributos.

4. Divisão dos dados: dos 98 poços disponíveis para treino, realizamos uma seleção aleatória de 73 desses poços para treino e 25 para teste. A divisão dos dados fica nas seguintes 4 variáveis *X_train*, *X_test*, *y_train* e *y_test*. Onde *X_train* e *X_test* são os atributos do conjunto de treino e de teste respectivamente, e

y_train e *y_test* são as classes do conjunto de treino e de teste respectivamente. Tínhamos 31 atributos e passamos para 29 atributos ao remover os atributos FORCE_2020_LITHOFACIES_LITHOLOGY e WELL.

5. Escalonamento dos dados: Para acelerar a velocidade do modelo, assim como torná-lo mais robusto, usamos o *standard scaler* nos valores dos atributos, fazendo com que seus intervalos de valores variem entre aproximadamente -3 até aproximadamente +3.

6. Imputação (atribuição/preenchimento) dos dados: Como temos vários atributos com dados ausentes, utilizamos o modelo *Bayesian Ridge* para imputar os valores ausentes de um atributo baseado nos valores dos demais atributos.

7. Formatação dos dados: o formato de entrada necessário para nossa LSTM consiste em uma matriz 3D. As variáveis *X_train* e *X_test* possuem as dimensões: poços, exemplos de camadas do poço, e atributos. Já as variáveis *y_train* e *y_test* possuem as dimensões: poços, exemplos de camadas de poços, e categorias no formato *one hot encoded*. Contudo, para que a rede aceite a entrada, todos os poços devem ter a mesma quantidade de camadas. Para que possamos ter a mesma quantidade de camadas em cada poço, pegamos o poço com a maior quantidade de camadas e fizemos com que todos os poços tenham a mesma quantidade de camada através do *padding* com zeros ao final de cada um dos poços. Tal processo de *padding* foi feito tanto para os conjuntos de treino *X_train* e *y_train*, assim como para os conjuntos de teste *X_test* e *y_test*. Com isso, as dimensões finais dos conjuntos de dados são:

    - X_train: (73, 25131, 29), onde 73 é o número de poços, 25131 é a quantidade de camadas do poço com maior número de camadas, X é o número de atributos utilizados.
    - X_test: (25, 23879, 29), onde 25 é o número de poços, 23879 é a quantidade de camadas do poço com maior número de camadas, X é o número de atributos utilizados.
    - y_train: (73, 25131, 12), onde 73 é o número de poços, 25131 é a quantidade de camadas do poço com maior número de camadas, e 12 é o número de classes no formato *one hot encoded*.
    - y_test: (25, 23879, 12), onde 73 é o número de poços, 23879 é a quantidade de camadas do poço com maior número de camadas, e 12 é o número de classes no formato *one hot encoded*.

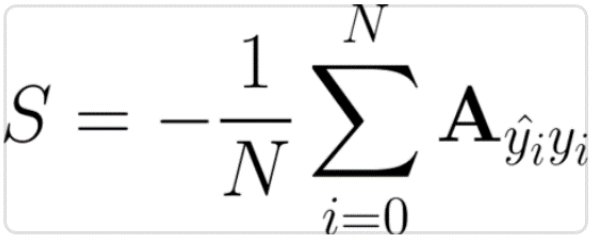
4.4 MÉTRICA DE AVALIAÇÃO

A competição possui sua própria métrica de avaliação, a qual chamaremos de *score*. Em vez de penalizar cada previsão errada das litofácies, a competição decidiu usar uma matriz de penalidade customizada (Figura 22) derivada da entrada média de uma amostra representativa de geocientistas. Isso permite que previsões petrofisicamente irracionais sejam avaliadas por um grau de "erro"(Figura 23). A Figura 23 apresenta o *score*, onde A é a matriz de penalidade abaixo, N é o número de amostras, $\hat{y}_i$ é o rótulo litológico verdadeiro e y_i é o rótulo litológico previsto. Esta métrica de avaliação não é utilizada durante o treinamento da rede, mas apenas no modelo final para mensurar sua qualidade em comparação com os outros modelos da competição.

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

Figura 22 – Matriz de penalidade.

Fonte – (2020, s.d.)



$$S = -\frac{1}{N}\sum_{i=0}^{N} \mathbf{A}_{\hat{y}_i y_i}$$

Figura 23 – Métrica de avaliação.

Fonte – (2020, s.d.)

5 EXPERIMENTOS E RESULTADOS

A estratégia para a realização dos experimentos foi comparar diferentes arquiteturas de LSTM avaliando o *score* sobre o conjunto de teste (*X_test*, *y_test*). Em seguida, selecionamos o melhor modelo, treinamos ele novamente só que agora usando tanto o conjunto de treino quanto o de teste, e avaliamos sua qualidade usando o *score*. Os passos descritos na seção de pré-processamento dos dados foram realizados em todos os experimentos.

5.1 MODELOS

Apresentamos aqui as características dos principais modelos treinados durante os experimentos. Uma arquitetura de rede neural pode possuir diversas camadas. Cada camada possui suas próprias características e parâmetros, as camadas utilizadas nas arquiteturas foram:

- *Biderectional*: Conectam duas camadas ocultas de direções opostas à mesma saída. Com isso a camada de saída pode obter informações dos estados do passado (para trás) e futuros (para frente) simultaneamente.
- LSTM: trata as camadas de forma sequencial. Dado que a LSTM está configurada para retornar sequências, a sua saída possui um formato 3D.
- *Dropout*: define aleatoriamente as unidade de entrada para 0 durante o processo de treinamento, possivelmente ajudando a prevenir o *overfitting* do modelo. Sua taxa de entradas para serem descartadas é passado como argumento.
- *Batch normalization*: aplica uma transformação que mantém a média de saída próxima a 0 e o desvio padrão da saída próximo a 1, possivelmente contribuindo para a velocidade do processo de treinamento e com a regularização, e reduzindo o erro de generalização.
- *Dense*: camada padrão de modelos de rede neural, ela serve tanto para possivelmente contribuir com a acurácia da rede quanto para mudar as dimensões dos vetores. Foi utilizado tanto a função de ativação ReLU quanto a *softmax*. A função de ativação *softmax* é escolhida para fazer com que o vetor de saída possua valores entre 0 e 1 e que sua soma seja igual a 1. Com isso podemos escolher o índice de maior valor do vetor o qual corresponde a classe da amostra da camada litológica em questão.
- *Time distributed*: trata o formato de saída 3D da camada LSTM para que a camada *dense* possa utilizá-la.

- *Masking*: para cada passo de tempo fornecido na entrada da rede, se todos os valores naquele intervalo de tempo forem iguais ao parâmetro *mask_value*, o passo de tempo será mascarado (pulado) em todas as camadas abaixo. Isto é, dado que utilizamos *padding* nos nossos dados de entrada, a camada *masking* irá desconsiderar o *padding* durante o treinamento da rede.

Além disso, também temos parâmetros para configurar no momento da compilação e treinamento do modelo, tais como:

- Otimizadores: algoritmos que otimizam a conversão do modelo. O otimizador utilizado nos modelos foi o Adam.
- Taxa de aprendizado: indica a velocidade de atualização dos pesos durante o treinamento da rede. Foi configurado para 0.002 em todos os modelos.
- Função de perda: utilizamos a *categorical crossentropy*, que é a função utilizada para problemas de classificação de multi classes, fazendo com que cada amostra de camada seja pertencente a apenas uma classe.
- Métrica de avaliação: utilizamos a métrica acurácia, que também é utilizada para tratar problemas de classificação. Tal métrica leva em consideração os valores de verdadeiro positivo, verdadeiro negativo, falso positivo e falso negativo.
- Dados de entrada: utilizamos os conjuntos de dados de treino *X_train* e *y_train* para realizar o treinamento do modelo.
- Dados de validação: utilizamos os conjuntos de dados de teste *X_test* e *y_test* para a validação do modelo.
- Épocas: número de épocas para treinar o modelo. Uma época é uma iteração sobre todos os dados de entrada fornecidos. Utilizamos 150 épocas em todos os modelos.
- Embaralhamento: se deve-se embaralhar os dados de treinamento antes de cada época. Visto que a sequência dos dados de treinamento é relevante, tal booleano foi configurado para falso.
- Tamanho do lote: Número de amostras por atualização do gradiente da rede.

Por fim, temos o parâmetro *callback*, que é passado como argumento para o treinamento do modelo, e que possui as seguintes características:

- *Early stopping*: encerra o treinamento quando uma métrica monitorada parar de melhorar durante um certo número de épocas. A métrica monitorada é a perda do conjunto de validação. O número de épocas para esperar antes de encerrar o treinamento do modelo é chamada de paciência.

A Tabela 2 abaixo apresenta as arquiteturas dos modelos treinados, onde P significa paciência, E significa épocas e B significa tamanho do lote. A função de ativação utilizada na LSTM foi a tangente hiperbólica (TanH).

Após alguns testes de modelos prévios, chagamos ao Modelo 1. O Modelo 1 possui a arquitetura mais simples dentre os demais modelos apresentados, visto que ele possui apenas 64 neurônios na LSTM e apenas uma camada *dense*. No Modelo 2 incluímos uma camada *dense* com 30 neurônios. No modelo 3 aumentamos a quantidade de neurônios para 128. O Modelo 4 não possui uma camada *bidirectional*, possui uma camada de *dropout*, *batch normalization*, e 3 camadas *dense*. O Modelo 5 é idêntico ao Modelo 1, exceto pelo fato de que o Modelo 5 não possui a camada *bidirectional*. Após tais experimentos, o Modelo 1 foi o que apresentou melhores resultados, mas ainda havia a possibilidade de tal modelo melhorar seus resultados caso ele fosse treinado por mais épocas. Portanto, no Modelo 6, também criamos um modelo idêntico ao Modelo 1, exceto que, no Modelo 6, não definimos um *early stopping*, consequentemente o Modelo 6 foi executado por 150 épocas.

P = 25, E = 150, B = 1
Masking
Bidirectional (LSTM 64 TanH)
Dropout 0.3
Time Distr. (Dense 12 Softmax)

(a) Modelo 1

P = 25, E = 150, B = 1
Masking
Bidirectional (LSTM 64 TanH)
Dropout 0.3
Time Distr. (Dense 30 ReLU)
Time Distr. (Dense 12 Softmax)

(b) Modelo 2

P = 25, E = 150, B = 1
Masking
Bidirectional (LSTM 128 TanH)
Dropout 0.3
Time Distr. (Dense 12 Softmax)

(c) Modelo 3

P = 25, E = 150, B = 1
Masking
LSTM 200 TanH
Dropout 0.5
Batch Normalization
Time Distr. (Dense 50 ReLU)
Time Distr. (Dense 30 ReLU)
Time Distr. (Dense 12 Softmax)

(d) Modelo 4

P = 25, E = 150, B = 1
Masking
LSTM 64 TanH
Dropout 0.3
Time Distr. (Dense 12 Softmax)

(e) Modelo 5

P = ∞, E = 150, B = 1
Masking
Bidirectional (LSTM 64 TanH)
Dropout 0.3
Time Distr. (Dense 12 Softmax)

(f) Modelo 6

Tabela 2 – Arquiteturas dos modelos.

5.2 COMPARAÇÃO ENTRE MODELOS

Na Figura 24 apresentamos os gráficos de cada modelo, mostrando a evolução da acurácia do conjunto de treino, acurácia do conjunto de teste, perda do conjunto de treino e perda do conjunto de teste ao longo das épocas de treino.

Todos os modelos possuem resultados similares. Vale ressaltar que, o Modelo 1 e o Modelo 6 são idênticos exceto pelo falo de que o Modelo 6 foi treinado por 150 épocas enquanto o Modelo 1 sofreu *early stopping* na época 33. Como veremos mais adiante, o Modelo 1 possui melhor *score* se comparado ao Modelo 6. Isso é devido ao *overfitting* que começa a ocorrer no Modelo 6 após cerca de 40 épocas. *Overfitting* ocorre quando o modelo torna-se muito bom em prever dados que foram incluídos durante seu treinamento, mas não é tão bom ao classificar dados para os quais não foi treinado.

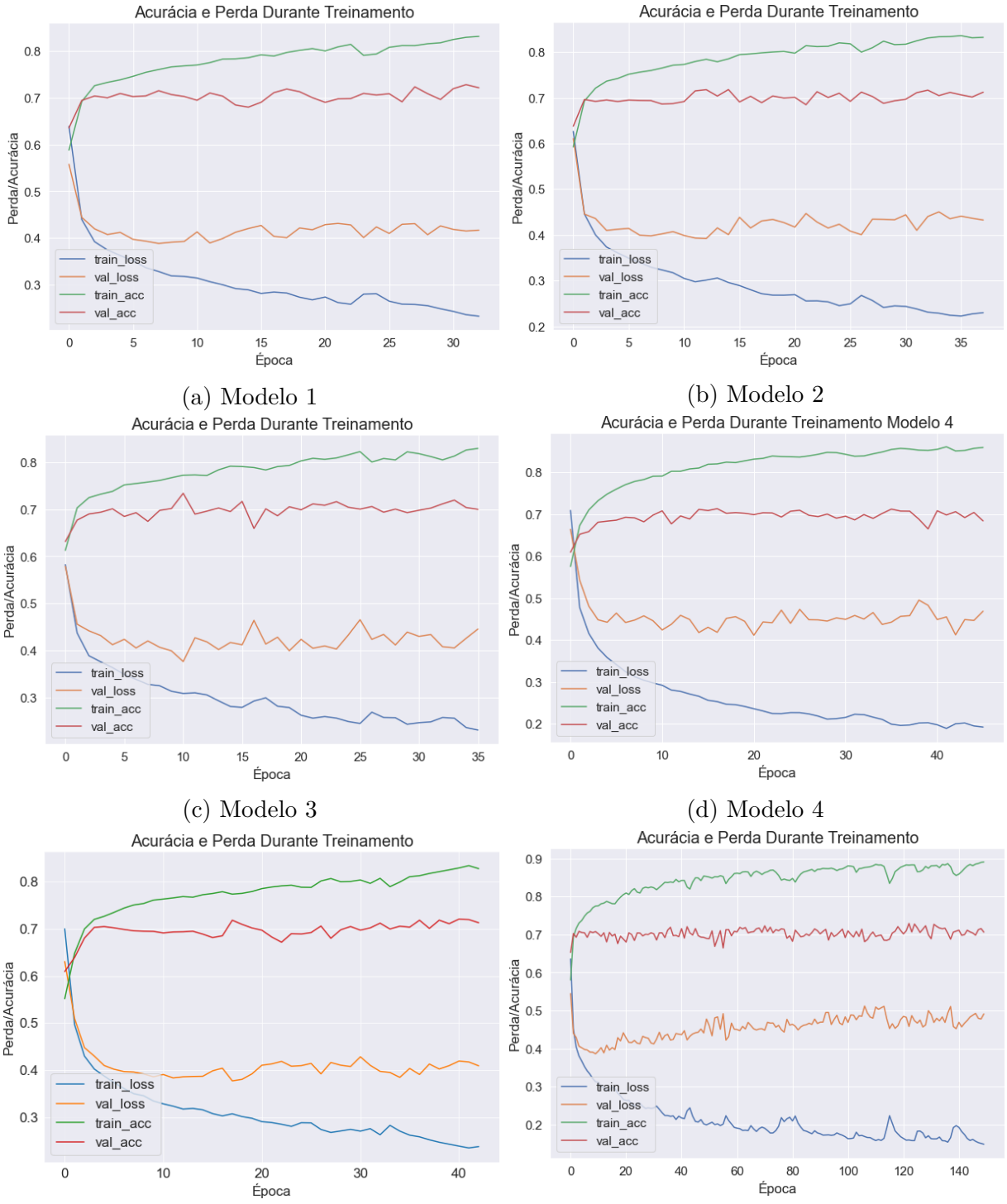
Na Figura 25 apresentamos a matriz de confusão de cada modelo avaliado sobre o conjunto de teste. O mapeamento do identificador e litologia das matrizes de confusão pode ser visto na Tabela 3.

Notamos que a classe mais prevista pelo modelo é a classe xisto, o que faz sentido, dado que tal classe representa 59.51% das classes do conjunto de teste. Entretanto, observamos que todos os modelos acabam erroneamente prevendo muitas amostras da classe xisto para amostras que na verdade pertencem a classe arenito/xisto. Isso ocorre pois a classe xisto é bem similar a classe arenito/xisto. Por conta da similaridade entre tais classes, esse tipo de erro é pouco penalizado pela métrica *score*, como pode ser visto na Figura 22. Observamos também que a classe embasamento não está presente no conjunto de teste, isso se deve a sua pouquíssima quantidade de exemplos no nosso conjunto de dados. Os modelos corretamente não preveem a classe embasamento para nenhuma das amostras. Também vemos que praticamente o único modelo que prevê a classe dolomita é o Modelo 6. Isso pode ter ocorrido pois a classe dolomita é percentualmente aproximadamente duas vezes mais presente no conjunto de treino do que no conjunto de teste (0.1669% das amostras no conjunto de treino e 0.077318% no conjunto de teste), e sabemos que o Modelo 6 é o único modelo que sofreu *overffiting*.

Na Tabela 4, temos a quantidade de acertos, quantidade de amostras do conjunto de teste e a porcentagem de acerto por classe do Modelo 1. Com tal tabela podemos ver mais claramente que o modelo tem dificuldade em identificar dolomitas (0.0%), arenito/xisto (22.96%), giz (39.52%) e carvão (40.76%). Por outro lado, o modelo possui mais de 80% de acerto ao identificar embasamento (100%), halita (98.35%), anidrita (97.56%) e xisto (82.22%).

Por fim apresentamos, na Tabela 5, a qualidade de cada modelo utilizando o *score* e a acurácia sobre o conjunto de teste e validação.

O Modelo 1 possui a maior acurácia (72.19%) e *score* (-0.7006) no conjunto de teste, porém possui o segundo pior resultado no conjunto de validação. Por outro lado, o Modelo 6 possui a maior acurácia (77.24%) e *score* (-0.5667) no conjunto de validação,



(a) Modelo 1

(b) Modelo 2

(c) Modelo 3

(d) Modelo 4

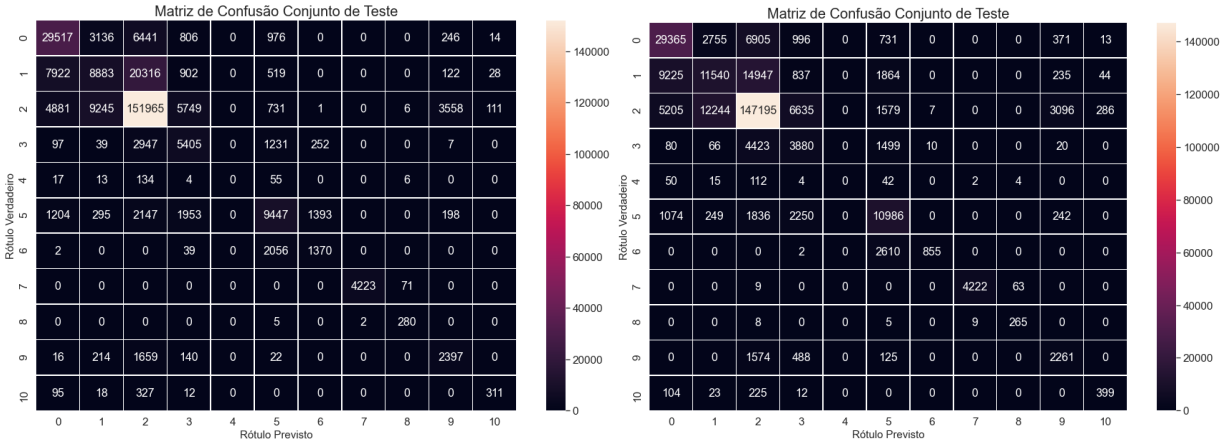
(e) Modelo 5

(f) Modelo 6

Figura 24 – Comparação de acurácia e perda, durante treinamento, entre os modelos.

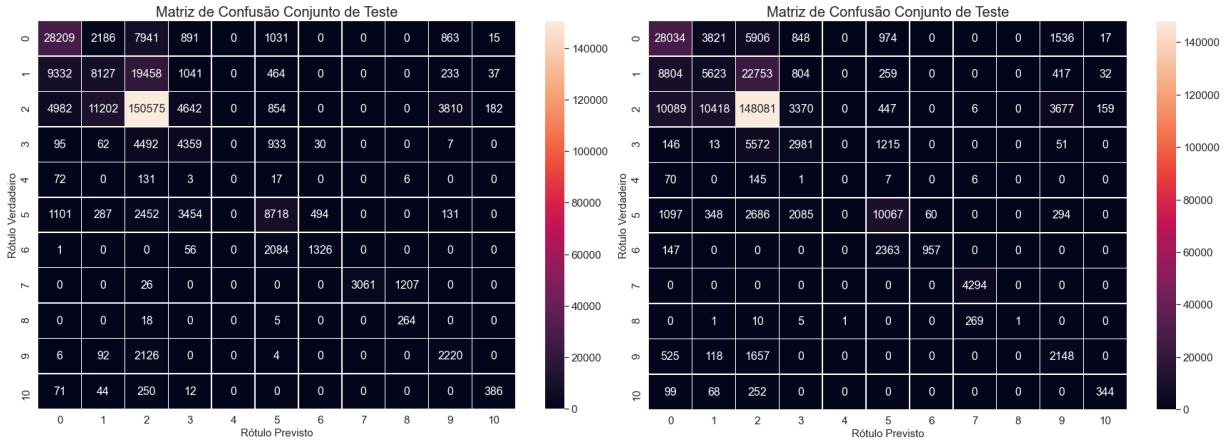
Fonte – autor

e fica em segundo no conjunto de teste. Essa disparidade entre a qualidade no conjunto de teste e no conjunto de validação também ocorreu durante a competição, como pode ser visto na Tabela 6. Dado que não teríamos acesso ao conjunto de validação durante a competição, não podemos selecionar o modelo que obteve melhor desempenho no conjunto



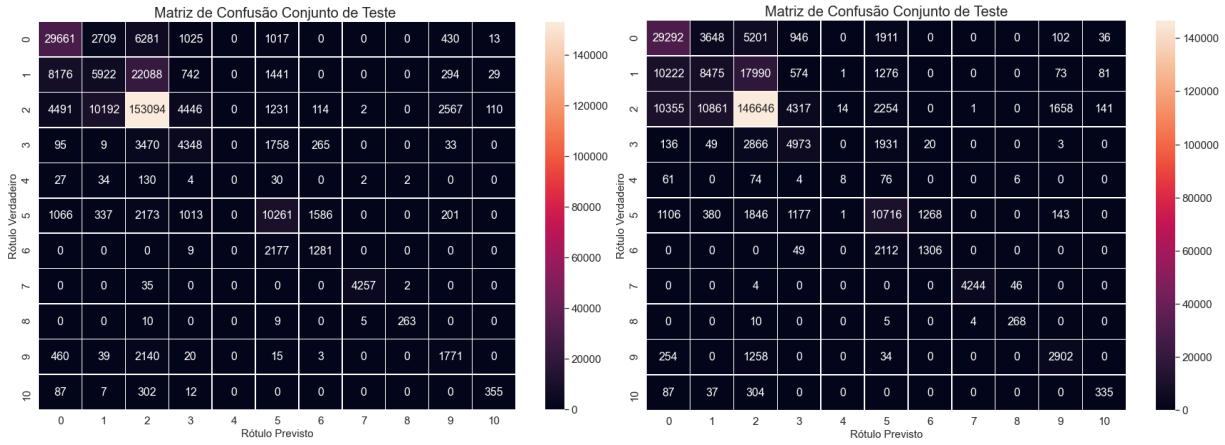
(a) Modelo 1

(b) Modelo 2



(c) Modelo 3

(d) Modelo 4



(e) Modelo 5

(f) Modelo 6

Figura 25 – Comparação de matrizes de confusão, sobre o conjunto de teste, entre os modelos.

Fonte – autor

de validação. Dado que o Modelo 1 obteve os melhores resultados no conjunto de teste, optamos por selecioná-lo como sendo o melhor modelo dentre os modelos testados.

Identificador	Litologia
0	Arenito
1	Arenito/Xisto
2	Xisto
3	Marga
4	Dolomita
5	Calcário
6	Giz
7	Halita
8	Anidrita
9	Tufo
10	Carvão
11	Embasamento

Tabela 3 – Mapeamento entre identificador e litologia das matrizes de convolução.

Fonte – autor

Classe	Acertos	Quantidade	Porcentagem de acerto
Arenito	29517	41136	71.75%
Arenito/Xisto	8883	38692	22.96%
Xisto	151965	176247	86.22%
Marga	5405	9978	54.17%
Dolomita	0	229	0.0%
Calcário	9447	16637	56.78%
Giz	1370	3467	39.52%
Halita	4223	4294	98.35%
Anidrita	280	287	97.56%
Tufo	2397	4448	53.89%
Carvão	311	763	40.76%
Embasamento	0	0	100%

Tabela 4 – Acertos, quantidade de amostras no conjunto de teste e porcentagem de acerto do Modelo 1.

Fonte – autor

Modelo	Acurácia (Teste)	Acurácia (Validação)	Score (Teste)	Score (Validação)
1	**72,19%**	72,93%	**-0.7006**	-0.6862
2	71,23%	75,69%	-0.7328	-0.5954
3	69,97%	73,16%	-0.7633	-0.6663
4	68,38%	75,33%	-0.8120	-0.6133
5	70,61%	70,09%	-0.7606	-0.7526
6	71,31%	**77,24%**	-0.7262	**-0.5667**

Tabela 5 – Comparação de acurácia e *score*, sobre o conjunto de teste e validação, entre os modelos.

Fonte – autor

Time	Score teste	Posição teste	Score validação	Posição validação
Olawale Ibrahim	-0.5118	24	-0.4690	1
GIR	-0.5037	11	-0.4792	2
ICA	-0.4943	6	-0.4954	3
H3G	-0.509	17	-0.5045	4
ISPL	-0.4885	2	-0.5084	5

Tabela 6 – Comparação de acurácia e posição, sobre o conjunto de teste (da competição) e validação, entre os 5 primeiros colocados da competição.

Fonte – autor

5.3 RESULTADOS

A partir da experimentação, escolhemos o Modelo 1 como o melhor dos modelos e treinamos ele novamente, mas dessa vez utilizando tanto o conjunto de treino quanto o de teste. Além disso, configuramos seu número de épocas para 40 e removemos o *early stopping*. Em seguida, mostramos seu gráfico de acurácia e perda, sua matriz de confusão, avaliamos sua qualidade usando o *score* no conjunto de validação, e comparamos o seu *score* com os 5 primeiros colocados da competição.

A Figura 26 apresenta a acurácia e perda do melhor modelo durante o treinamento. Tal gráfico é similar aos dos demais modelos vistos anteriormente. Podemos observar que a perda do treinamento cai com o passar das épocas, o que é esperado. Possivelmente, dado que o modelo tivesse mais épocas para ser treinado, a perda de treino provavelmente continuaria caindo. Entretanto, como observamos no Modelo 6 da Figura 24, o modelo possivelmente começaria a sofrer *overffiting* após aproximadamente 40 épocas.

Na Figura 27 mostramos a matriz de confusão do melhor modelo sobre o conjunto de validação. Tal matriz de confusão apresenta características similares às matrizes de confusão apresentadas durante a experimentação. A classe mais prevista é a classe xisto, dado sua maior porcentagem (58.68%) em relação as demais amostras do conjunto de validação. Continuamos cometendo erros para amostras que são similares entre si, como prever a classe xisto enquanto a classe verdadeira é arenito/xisto. Além disso, continuamos acertando ao não prever nenhuma amostra como pertencente a classe embasamento, visto que no conjunto de validação também não temos nenhuma amostra dessa classe.

Na Tabela 7, temos a quantidade de acertos, quantidade de amostras do conjunto de validação e a porcentagem de acerto por classe do melhor modelo. Observamos que o melhor modelo tem dificuldade em identificar dolomitas (20.56%), arenito/xisto (21.91%), marga (24.7%), tufo (39.96%) e anidrita (40.54%). Em contrapartida, o modelo possui mais de 80% de acerto ao identificar embasamento (100%), xisto (93.09%) e halita (90.43%).

A Tabela 8 mostra os *scores*, modelo e posição, utilizando os dados de validação, dos 5 primeiros colocados na competição, assim como o *score*, modelo e posição do modelo selecionado criado no presente trabalho.

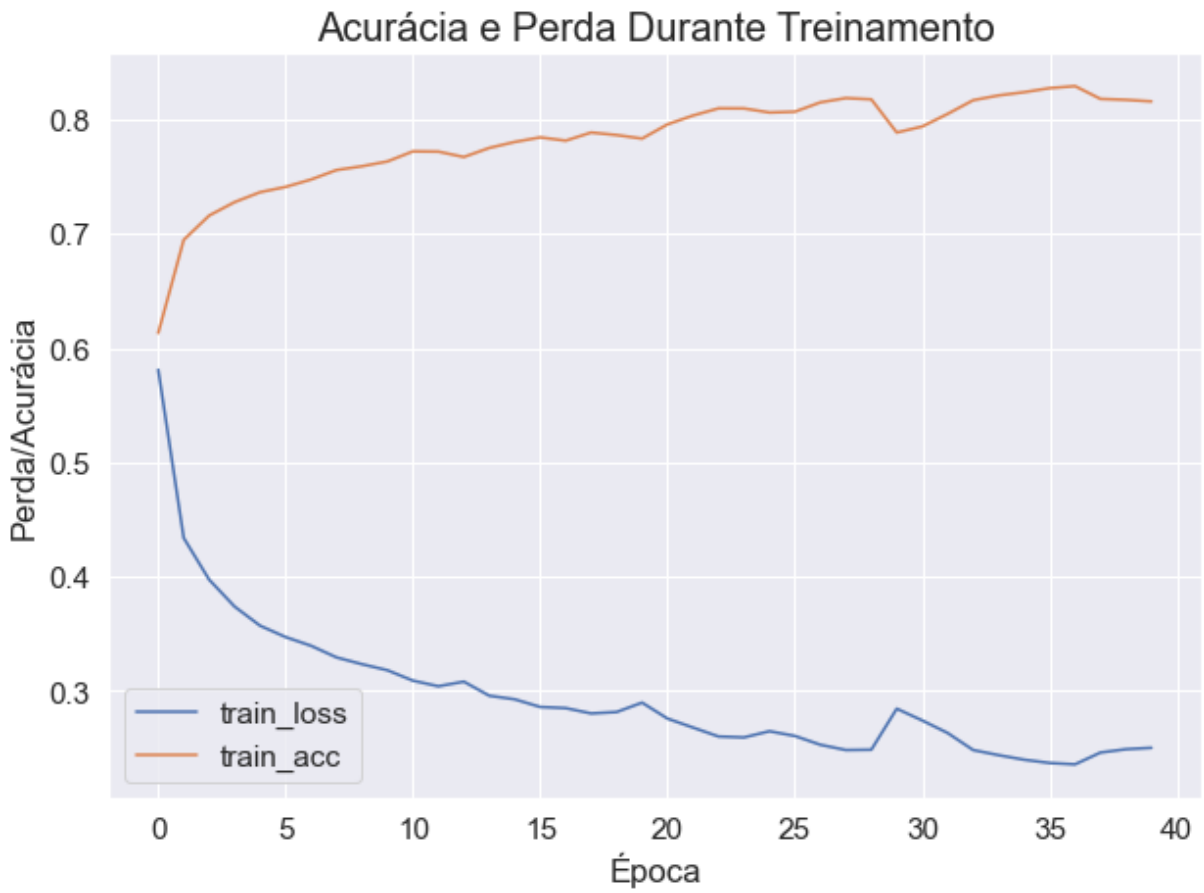


Figura 26 – Acurácia e perda do melhor modelo durante treinamento.

Fonte – autor

O modelo proposto não apresentou resultados melhores do que os primeiros colocados da competição sobre o conjunto de validação, seu *score* final foi de -0.5450 e sua acurácia final foi de 78,13%. A posição do modelo não pôde ser estabelecida pois o resultado final da competição é apresentado apenas até o décimo terceiro colocado, que possui seu *score* em -0.5441. O resultado atingido pelo modelo proposto não comprova a ineficiência da utilização de LSTM sobre o conjunto de dados descrito no presente trabalho. Mais experimentos poderiam ser realizados considerando outros aspectos tanto da arquitetura do modelo quanto do pré-processamento dos dados para que tivéssemos mais confiança em descartar a utilização de LSTM para a classificação de litofácies.

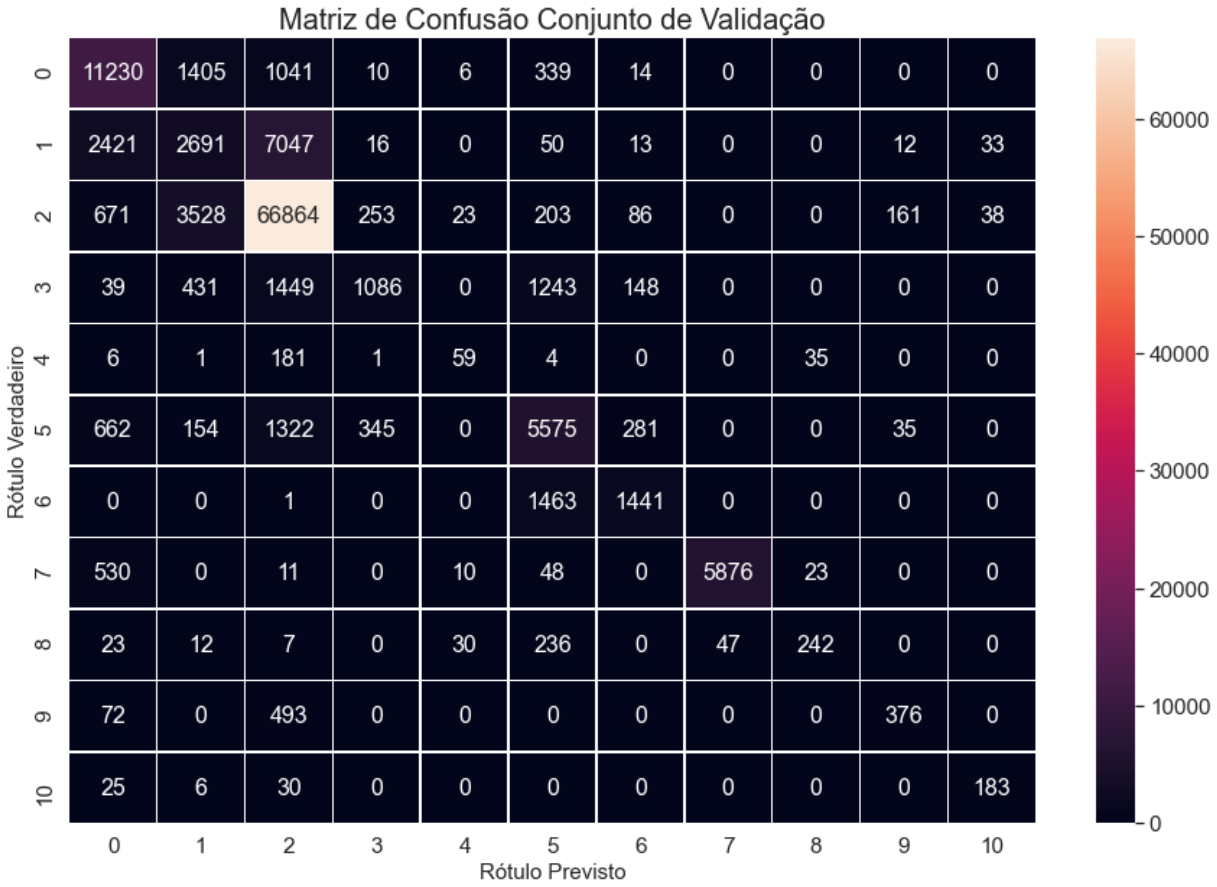


Figura 27 – Matriz de convolução do melhor modelo sobre o conjunto de validação.

Fonte – autor

Classe	Acertos	Quantidade	Porcentagem de acerto
Arenito	11230	14045	79.96%
Arenito/Xisto	2691	12283	21.91%
Xisto	66864	71827	93.09%
Marga	1086	4396	24.7%
Dolomita	59	287	20.56%
Calcário	5575	8374	66.58%
Giz	1441	2905	49.6%
Halita	5876	6498	90.43%
Anidrita	242	597	40.54%
Tufo	376	941	39.96%
Carvão	183	244	75.0%
Embasamento	0	0	100%

Tabela 7 – Acertos, quantidade de amostras no conjunto de validação e porcentagem de acerto do melhor modelo.

Fonte – autor

Time	Modelo	Score	Posição
Olawale Ibrahim	XGBoost	-0.4690	1
GIR	XGBoost	-0.4792	2
ICA	Random Forest	-0.4954	3
H3G	XGBoost	-0.5045	4
ISPL	XGBoost	-0.5084	5
Modelo Selecionado	LSTM	-0.5450	?

Tabela 8 – Comparação de acurácia e *score*, sobre o conjunto de validação, entre os modelos.

Fonte – autor

6 CONCLUSÕES

Neste trabalho apresentamos os conceitos básicos sobre fácies sedimentares, litofácies e uma breve descrição dos atributos utilizados pela Force 2020 Machine Learning Competition para a classificação de litofácies. Além disso, tivemos uma introdução aos conceitos de IA, ML, DL, RNN e LSTM. Em seguida, mostramos e discutimos os resultados de quatro artigos que aplicaram técnicas para a classificação de litofácies através de modelos de ML. Apresentamos também o conjunto de dados, o pré-processamento realizado sobre os dados, e a métrica de avaliação *score*. Demonstramos alguns dos experimentos realizados utilizando LSTMs, realizamos a comparação entre eles, selecionamos o melhor dentre os modelos apresentados, e comparamos sua qualidade em relação aos modelos dos 5 primeiros colocados na competição.

O objetivo geral do presente trabalho foi atingido. Utilizamos uma rede LSTM, que é capaz de levar em consideração a sequência temporal de formação das diversas camadas de rochas, no problema da classificação automática de litofácies. Apesar do modelo proposto não ter superado os primeiros colocados na competição, seu desempenho não ficou muito distante dos mesmos. Todos os modelos dos 5 primeiros colocados na competição são constituídos por conjuntos de árvores de decisão.

Árvores de decisão não necessitam de imputação de dados. O time Olawale Ibrahim, primeiro colocado, fez o preenchimento dos valores ausentes apenas com um único valor constante de valor -999 e ainda conseguiu bons resultados. Por outro lado, o time ICA, terceiro colocado, imputou os valores utilizando a mediana para atributos contínuos e a moda para atributos discretos. Talvez a imputação utilizada no presente trabalho não tenha sido a melhor opção de imputação.

Não conseguimos alterar a função de perda para que o modelo levasse em consideração a matriz de penalidade durante o treinamento da rede neural. Modelos baseados em árvores de decisão não possuem tal parâmetro, portanto não precisaram levar isso em consideração. Possivelmente, com uma função de perda funcionando, o resultado do modelo poderia ter melhorado.

Os modelos propostos pelos times ICA e H3G consideraram o desbalanceamento entre classes durante o treinamento de seus modelos. Visto que temos um grande desbalanceamento entre classes, possivelmente a utilização de pesos para classes poderia ter aumentado a qualidade do modelo proposto.

O modelo também poderia possivelmente ter sido melhorado com a utilização de um conjunto diferente de atributos, atribuição de pesos às camadas específicas e experimentação com outras arquiteturas de rede neural.

6.1 TRABALHOS FUTUROS

Existem algumas possibilidades para melhorar o desempenho da LSTM. Para a realização de trabalhos futuros seria relevante considerar o desbalanceamento entre classes. Para tal poderia-se criar dados sintéticos para as classes sub-representadas, ou reduzir a quantidade de dados das classes mais volumosas, ou até um misto entre essas duas estratégias. Ainda há a possibilidade de considerar o desbalanceamento entre classes diretamente na arquitetura da rede, passando um peso diferente para cada uma das classes.

Outra etapa que poderia ser adicionada seria a utilização do atributo FORCE_2020_LITHOFACIES_CONFIDENCE que indica a medida qualitativa de confiança da interpretação da litologia descrita. Tal atributo poderia ser utilizado durante o treinamento para filtrar dados e/ou atribuir pesos às amostras.

Outras estratégias para imputação de dados também poderiam ser exploradas. Opções variam desde a utilização de conhecimentos geológicos e utilização de outros modelos de ML, até o preenchimento com a média ou a moda dos dados.

A própria competição cita a possibilidade de explorar a redução do peso das amostras imediatamente próximas aos limites das litofácies, pois a escolha exata em cm dos limites das litofácies é frequentemente subjetiva e um pouco imprecisa.

Dentre as arquiteturas experimentadas, o modelo proposto possuía a arquitetura mais simples. Poderia-se ainda alterar a arquitetura do modelo, diminuir ou aumentar a quantidade de neurônios, explorar diferentes combinações de camadas, funções de ativação, número de épocas e demais argumentos.

Por fim, durante o treinamento do modelo proposto, utilizamos como função de perda a *categorical crossentropy*. Utilizar tal função não é o ideal para o treinamento do modelo, visto que há algumas classes que são similares entre si e já temos esse conhecimento de similaridade indicado na matriz de penalidade da competição. Dito isso, criar uma função de perda customizada que leve em consideração a matriz de penalidade possivelmente auxiliaria no treinamento do modelo.

REFERÊNCIAS

2020, Force. **Force 2020 Machine Learning competition**. Disponível em: https://github.com/bolgebrygg/Force-2020-Machine-Learning-competition. Acessado em: 04/03/2021.

BOGGS, S. **Principles of Sedimentology and Stratigraphy**. [*S.l.*]: Prentice Hall, 2001. ISBN 9780130996961.

BRITANNICA. **Sedimentary Facies**. Disponível em: https://www.britannica.com/science/sedimentary-facies. Acessado em: 06/03/2021).

COLLEGE, Wenatchee Valley. **Depositional Environments**. Disponível em: https://commons.wvc.edu/rdawes/g101ocl/basics/depoenvirons.html#:~:text=A%5C%20depositional%5C%20environment%5C%20is%5C%20a,are%5C%20sometimes%5C%20called%5C%20sedimentary%5C%20environments. Acessado em: 06/03/2021).

DELL'AVERSANA, Paolo. Comparison of different Machine Learning algorithms for lithofacies classification from well logs. **Bollettino di Geofisica Teorica ed Applicata**, v. 60, n. 1, 2019.

DRAMSCH, Jesper Sören. Chapter One - 70 years of machine learning in geoscience in review. *In:* MOSELEY, Ben; KRISCHER, Lion (Ed.). **Machine Learning in Geosciences**. [*S.l.*]: Elsevier, 2020. v. 61. (Advances in Geophysics). P. 1–55. DOI: https://doi.org/10.1016/bs.agph.2020.08.002. Disponível em: http://www.sciencedirect.com/science/article/pii/S0065268720300054.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. [*S.l.*]: MIT Press, 2016. http://www.deeplearningbook.org.

IMAMVERDIYEV, Yadigar; SUKHOSTAT, Lyudmila. Lithological facies classification using deep convolutional neural network. **Journal of Petroleum Science and Engineering**, Elsevier, v. 174, p. 216–228, 2019.

INTERIOR BUREAU OF RECLAMATION, U.S. Department of the. **Engineering Geology Field Manual**. [*S.l.*: *s.n.*], 1998. https://www.usbr.gov/tsc/techreferences/mands/geologyfieldmanual.html.

JAIKLA, Chayawan *et al.* FaciesNet: Machine Learning Applications for Facies Classification in Well Logs. *In:* SECOND Workshop on Machine Learning and the Physical Sciences at the 33rd Conference on Neural Information Processing Systems (NeurIPS). [*S.l.*: *s.n.*], 2019. P. 10–12.

MANDAL, Partha Pratim; REZAEE, Reza. Facies classification with different machine learning algorithm–An efficient artificial intelligence technique for improved classification. **ASEG Extended Abstracts**, Taylor & Francis, v. 2019, n. 1, p. 1–6, 2019.

MITCHELL, Tom Michael. **Machine Learning**. 1. ed. [*S.l.*]: McGraw-Hill Education, 1997. ISBN 0070428077, 9780070428072.

MOHRI, M.; ROSTAMIZADEH, A.; TALWALKAR, A. **Foundations of Machine Learning**. [*S.l.*]: MIT Press, 2012. (Adaptive Computation and Machine Learning series). ISBN 9780262018258.

MOORE, John A. Geographic variation of adaptive characters in Rana pipiens Schreber. **Evolution**, JSTOR, p. 1–24, 1949.

PETROWIKI. **Spectral gamma ray logs**. Disponível em: `https://petrowiki.spe.org/Spectral_gamma_ray_logs`. Acessado em: 21/03/2021.

RUSSELL, S.J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3. ed. Upper Saddle River, New Jersey 07458: Prentice Hall, 2010. ISBN 0136042597, 9780136042594.

SAMUEL, A. L. Some Studies in Machine Learning Using the Game of Checkers. **IBM Journal of Research and Development**, v. 3, n. 3, p. 210–229, 1959. DOI: `10.1147/rd.33.0210`.

SCHLUMBERGER. **The oil and gas industry's reference work**. Disponível em: `https://www.glossary.oilfield.slb.com/en/`. Acessado em: 21/03/2021.

SUGIYAMA, S. **Human Behavior and Another Kind in Consciousness: Emerging Research and Opportunities: Emerging Research and Opportunities**. [*S.l.*]: IGI Global, 2019. (Advances in Human and Social Aspects of Technology (2328-1316)). ISBN 9781522582182. Disponível em: `https://books.google.com.br/books?id=9CqQDwAAQBAJ`.

WEI, Zhili *et al.* Characterizing rock facies using machine learning algorithm based on a convolutional neural network and data padding strategy. **Pure and Applied Geophysics**, Springer, v. 176, n. 8, p. 3593–3605, 2019.

WIKI, Computer Science. **Max-pooling / Pooling**. Disponível em: `https://computersciencewiki.org/index.php/Max-pooling_/_Pooling`. Acessado em: 13/06/2021.

YU, Y. *et al.* A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. **Neural Computation**, v. 31, n. 7, p. 1235–1270, 2019. DOI: `10.1162/neco_a_01199`.

YU, Yong *et al.* A review of recurrent neural networks: LSTM cells and network architectures. **Neural computation**, MIT Press, v. 31, n. 7, p. 1235–1270, 2019.

APÊNDICE A – CÓDIGO FONTE

Neste apêndice será apresentado o código fonte referente ao presente trabalho.

```
# Import libraries

import pickle, random, math, copy, json, os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn import metrics

from keras.models import Sequential, load_model
from keras.layers import LSTM, Dense, Dropout, TimeDistributed, BatchNormalization,
↪  Bidirectional, Masking
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.optimizers import adam_v2
from keras import backend as K

# Set seeds to produce reproducible results

# This process of setting seeds was not done during TCC's experimentation phase. The
↪  only seeds set during experimentation were the random.seed and iterative imputer
↪  seed, thus, the split process and imputation were the same for all models, but the
↪  training process differs. Nevertheless, most of the models used during TCC were
↪  saved and can be found at https://github.com/MatheusSchaly/TCC.

seed_value = 42
os.environ['PYTHONHASHSEED'] = str(seed_value)
random.seed(seed_value)
np.random.seed(seed_value)
tf.random.set_seed(seed_value)
session_conf = tf.compat.v1.ConfigProto(intra_op_parallelism_threads=1,
↪  inter_op_parallelism_threads=1)
sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(),
↪  config=session_conf)
tf.compat.v1.keras.backend.set_session(sess)

# Load data

# Download the data:
```

```
38
39  # train.csv:
    ↪ https://drive.google.com/file/d/13Z7pZuiBqsAlPz_om5AQuRKVoF5sK132/view?usp=sharing
40
41  # hidden_validation.csv:
    ↪ https://drive.google.com/file/d/1bTfkCGerc2rkWZzEqZlyJ54ig7ZNCpLO/view?usp=sharing
42
43  # penalty_matrix.npy:
    ↪ https://drive.google.com/file/d/1aCurcgoGMOexI-MknRZdvhgH8UqBLIgV/view?usp=sharing
44
45  # Load training data
46  df = pd.read_csv('train.csv', sep=';')
47
48  # Load hidden validation data
49  df_val = pd.read_csv('hidden_validation.csv', sep=';')
50
51  # Load penalty matrix
52  A = np.load('penalty_matrix.npy', allow_pickle=True)
53
54  # Print 5 samples of training data
55  df.sample(5)
56
57  # Feature selection
58
59  # Based on the five winning codes of the competition, NaN values and feature selection
    ↪ methods
60  df.drop(['FORCE_2020_LITHOFACIES_CONFIDENCE', 'FORMATION', 'SGR', 'DTS',
61           'ROPA', 'DCAL', 'MUDWEIGHT', 'RMIC', 'ROP', 'RXO'], axis=1, inplace=True)
62
63  # Sort by WELL and then by DEPTH_MD
64  df.sort_values(['WELL', 'DEPTH_MD'], inplace=True)
65
66  # Print df shape (rows, columns)
67  print(df.shape)
68
69  # One-hot-encoding
70
71  # One-hot-encode the columns passed in the list
72  dummies = pd.get_dummies(df[['GROUP']], drop_first=True)
73  df = pd.concat([df.drop(['GROUP'], axis=1), dummies], axis=1)
74
75  # Print head of training data
76  df.head(5)
77
78  # Split the data
79
80  # Mapping of labels for easier understanding
81  lithology_numbers = {30000: 0, # Sandstone
82                       65030: 1, # Sandstone/Shale
```

```
                    65000: 2, # Shale
                    80000: 3, # Marl
                    74000: 4, # Dolomite
                    70000: 5, # Limestone
                    70032: 6, # Chalk
                    88000: 7, # Halite
                    86000: 8, # Anhydrite
                    99000: 9, # Tuff
                    90000: 10, # Coal
                    93000: 11} # Basement

# The data is devided based on wells, 3/4 of wells go to training and 1/4 goes to
↪  testing
n_wells_test = math.ceil(df['WELL'].nunique() / 4)
n_wells_train = df['WELL'].nunique() - n_wells_test
wells_train = set(random.sample(df['WELL'].unique().tolist(), n_wells_train))
wells_test = set(df['WELL'].unique().tolist()) - wells_train
print(f'n_wells_train: {n_wells_train}, n_wells_test: {n_wells_test}')

# Split the data into train and test
X_train_init = df[df['WELL'].isin(wells_train)]
X_test_init = df[df['WELL'].isin(wells_test)]

# Separate the classes from the features
y_train_init = X_train_init['FORCE_2020_LITHOFACIES_LITHOLOGY']
y_test_init = X_test_init['FORCE_2020_LITHOFACIES_LITHOLOGY']

# Get sequence of wells, in order, for future use
X_train_well = X_train_init['WELL'].tolist()
X_test_well = X_test_init['WELL'].tolist()

# Remove classes and well feature from X
X_train_init = X_train_init.drop(['FORCE_2020_LITHOFACIES_LITHOLOGY', 'WELL'], axis=1)
X_test_init = X_test_init.drop(['FORCE_2020_LITHOFACIES_LITHOLOGY', 'WELL'], axis=1)

# Apply classes mapping
y_train_init = y_train_init.map(lithology_numbers)
y_test_init = y_test_init.map(lithology_numbers)

# Print shapes
print(f'\nX_train_init.shape: {X_train_init.shape}\n\
y_train_init.shape: {y_train_init.shape}\n\
X_test_init.shape: {X_test_init.shape}\n\
y_test_init.shape: {y_test_init.shape}')

# Count classes, be aware that there is only one well that contains class 11
↪  (basement)
print(f'\ny_train_init.value_count():\n{y_train_init.value_counts()}')
print(f'\ny_test_init.value_count():\n{y_test_init.value_counts()}')
```

```
130
131 # Print the number of classes in train and test. If the numbers printed here are
    ↪ different, you have a label beloging to only training or only testing sets
132 print(f'\nlen(y_train_init.value_counts()): {len(y_train_init.value_counts())}')
133 print(f'len(y_test_init.value_counts()): {len(y_test_init.value_counts())}')
134
135 # Scaling
136
137 # Scale data using StandardScaler
138 scaler = StandardScaler()
139 X_train = scaler.fit_transform(X_train_init)
140 # Use the same scaler used in train into test and validation (further ahead)
141 X_test = scaler.transform(X_test_init)
142
143 # Transform data back to dataframe
144 X_train = pd.DataFrame(X_train, columns=X_train_init.columns)
145 X_test = pd.DataFrame(X_test, columns=X_test_init.columns)
146
147 # Impute missing data
148
149 # Impute missing data using IterativeImputer, it takes roughly 11 min to run this
150 imputer = IterativeImputer(random_state=seed_value)
151 X_train = imputer.fit_transform(X_train)
152 # Use the same scaler used in train into test and validation (further ahead)
153 X_test = imputer.transform(X_test)
154
155 # Transform data back to dataframe
156 X_train = pd.DataFrame(X_train, columns=X_train_init.columns)
157 X_test = pd.DataFrame(X_test, columns=X_test_init.columns)
158
159 # Print head of training data
160 df.head(5)
161
162 # Check if there are any missing value left (it should have none)
163 X_train.isna().sum().sort_values()
164
165 # LSTM model
166
167 # Format data into RNN format
168
169 # Create RNN format matrix
170 def create_rnn_matrix(df):
171     matrices = []
172     # Get well with most samples and pad the other well accordingly
173     n_max_samples = df['WELL'].value_counts()[0]
174     for well in df['WELL'].unique():
175         # Pad, at the end of each well, n_max_samples minus number of current well
            ↪ samples, so that all wells have same number of samples
176         unique_well_df = df.loc[df['WELL'] == well]
```

```
177         unique_well_np = unique_well_df.drop(['WELL'], axis=1).to_numpy()
178         n_pad_rows = n_max_samples - len(unique_well_np)
179         unique_well_np_padded = np.pad(unique_well_np, [(0, n_pad_rows), (0, 0)],
            ↪  mode='constant', constant_values=0)
180         matrices.append(unique_well_np_padded)
181     # Return np.array matrices having shapes (WELL, N_SAMPLES, N_FEATURES)
182     return np.array(matrices)
183 
184 # Return well and label columns
185 Xy_train = X_train.copy()
186 Xy_train['WELL'] = X_train_well
187 Xy_test = X_test.copy()
188 Xy_test['WELL'] = X_test_well
189 
190 # One-hot-encode the label column
191 y_train = pd.get_dummies(y_train_init)
192 y_test = pd.get_dummies(y_test_init)
193 
194 # As the test set doesn't have the label 11 in it, we will include it
195 y_test[11] = 0
196 
197 # Reinsert well to enable padding logic
198 y_train['WELL'] = X_train_well
199 y_test['WELL'] = X_test_well
200 
201 # Convert the data to Recurrent Neural Netowrk (RNN) format
202 X_train = create_rnn_matrix(Xy_train)
203 X_test = create_rnn_matrix(Xy_test)
204 y_train = create_rnn_matrix(y_train)
205 y_test = create_rnn_matrix(y_test)
206 
207 # Print shapes
208 print(f'\nX_train.shape: {X_train.shape}\n\
209 X_test.shape: {X_test.shape}\n\
210 y_train.shape: {y_train.shape}\n\
211 y_test.shape: {y_test.shape}')
212 
213 # If there is an error here, it's because during one-hot-encoding you created a label
    ↪  that exists in train but does not exist in test or vice-versa.
214 # This happens because either train has a sample (in the one-hot-encoded column) value
    ↪  that is not contained in test (or vice-versa)
215 X_train_init_cols = X_train_init.columns
216 X_test_init_cols = X_test_init.columns
217 for i in range(len(X_test_init.columns)):
218     if X_train_init_cols[i] != X_test_init_cols[i]:
219         print('We have a problem, train and test column have a different column
            ↪  order')
220 if len(X_train_init_cols) != len(X_test_init_cols):
221     print('We have a problem, either train has more columns than test or vice-versa')
```

```
222
223 # Save data input to the LSTM in files
224
225 np.save('X_train.npy', X_train)
226 np.save('X_test.npy', X_test)
227 np.save('y_train.npy', y_train)
228 np.save('y_test.npy', y_test)
229 # X_train = np.load('X_train.npy')
230 # X_test = np.load('X_test.npy')
231 # y_train = np.load('y_train.npy')
232 # y_test = np.load('y_test.npy')
233
234 # Creating and feeding LSTM architecture
235
236 # Create callbacks
237 callbacks = [EarlyStopping(monitor='val_loss', patience=25),
238              ModelCheckpoint('../model_checkpoint/model.h5',
239              save_best_only=True,
240              save_weights_only=False,
241              monitor='val_loss',
242              mode='min')]
243
244 # Create architecture
245 model = Sequential()
246 model.add(Masking(mask_value = 0.))
247 model.add(Bidirectional(LSTM(64, return_sequences=True, activation = "tanh")))
248 model.add(Dropout(0.3))
249 model.add(TimeDistributed(Dense(12, activation='softmax')))
250 adam = adam_v2.Adam(learning_rate=0.002)
251 model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
252
253 # Fit network architecture
254 history = model.fit(X_train, y_train, epochs=150, batch_size=1, shuffle=False,
255                     validation_data=(X_test, y_test), verbose=2,
                    ↪  callbacks=[callbacks])
256
257 # Print model summary
258 print(model.summary())
259
260 # Plot training process
261
262 # Plot history loss and accuracy
263 sns.set_style('darkgrid')
264 plt.figure(figsize=(10, 7))
265 plt.rc('font', size=13)
266 plt.plot(history.history['loss'], label='train_loss')
267 plt.plot(history.history['val_loss'], label='val_loss')
268 plt.plot(history.history['accuracy'], label='train_acc')
269 plt.plot(history.history['val_accuracy'], label='val_acc')
```

```
270 plt.title('Acurácia e Perda Durante Treinamento', fontsize = 20)
271 plt.xlabel("Época")
272 plt.ylabel("Perda/Acurácia")
273 plt.legend(loc='lower left')
274 plt.show()
275
276 # Save model and loss, val_loss, accuracy and val_accuracy
277
278 # Save model
279 model.save('models/LSTM_model_1.h5')
280 # model = load_model('models/LSTM_model_1.h5')
281
282 # Save history
283 json.dump(history.history, open("models/LSTM_model_1_history.json", 'w'))
284 # data = json.load(open("models/LSTM_model_1_history.json"))
285
286 # Making predictions
287
288 # Define competition's score function
289 def score(y_true, y_pred):
290     S = 0.0
291     y_true = y_true.astype(int)
292     y_pred = y_pred.astype(int)
293     for i in range(0, y_true.shape[0]):
294         S -= A[y_true[i], y_pred[i]]
295     return S/y_true.shape[0]
296
297 # As we will be predicting labels for all samples, including the padded ones, we will
    ↪  have to disconsider the rows that were padded.
298 # The rows to keep will have 1 as value and the rows to remove will have 0 as value
299 rows_to_keep = []
300 n_max_samples = Xy_test['WELL'].value_counts()[0]
301 for well in Xy_test['WELL'].unique():
302     n_samples = len(df.loc[df['WELL'] == well])
303     n_pad_rows = n_max_samples - n_samples
304     for i in range(n_samples):
305         rows_to_keep.append(1)
306     for i in range(n_pad_rows):
307         rows_to_keep.append(0)
308
309 # Save rows_to_keep to a file
310 np.save('rows_to_keep.npy', rows_to_keep)
311 # rows_to_keep = np.load('rows_to_keep.npy')
312
313 # Print rows to keep, it should have same length as X_test.shape[0] * X_test.shape[1]
314 print(f'len(rows_to_keep): {len(rows_to_keep)}')
315 print(f'X_test.shape[0] * X_test.shape[1]: {X_test.shape[0] * X_test.shape[1]}')
316
317 # Make predictions
```

```
318 yhat = np.argmax(model.predict(X_test), axis=-1)
319 y_pred_temp = yhat.ravel()
320
321 # Print predictions shape, it should have same length as len(rows_to_keep)
322 print(f'\ny_pred_temp.shape: {y_pred_temp.shape}')
323 print(f'len(rows_to_keep): {len(rows_to_keep)}')
324
325 # Remove padded rows from predictions
326 y_pred = []
327 for i in range(len(rows_to_keep)):
328     if rows_to_keep[i] == 1:
329         y_pred.append(y_pred_temp[i])
330
331 # Print length of predictions, it should have same length as y_test_init.shape
332 print(f'\nlen(y_pred): {len(y_pred)}')
333 print(f'y_test_init.shape: {y_test_init.shape}')
334
335 # Get y_true
336 y_true = y_test_init.to_list()
337
338 # Convert to series
339 y_pred = pd.Series(y_pred)
340 y_true = pd.Series(y_true)
341
342 # Save y_pred and y_true to a file
343 np.save('y_pred.npy', y_pred)
344 np.save('y_true.npy', y_true)
345 # y_pred = np.load('y_pred.npy')
346 # y_true = np.load('y_true.npy')
347
348 # Calculate score and accuracy
349 print('Score:', score(y_true, y_pred))
350 print('Accuracy:', metrics.accuracy_score(y_true, y_pred))
351
352 # Plot confusin matrix
353 cm = metrics.confusion_matrix(y_true, y_pred)
354 df_cm = pd.DataFrame(cm, range(cm.shape[0]), range(cm.shape[0]))
355 plt.figure(figsize=(15, 10))
356 sns.set(font_scale=1.4) # for label size
357 sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='d', linewidths=0.5) # font
    ↪ size
358
359 plt.title('Matriz de Confusão Conjunto de Teste', fontsize = 20)
360 plt.xlabel('Rótulo Previsto', fontsize = 15)
361 plt.ylabel('Rótulo Verdadeiro', fontsize = 15)
362
363 plt.show()
364
365 # Validation data
```

```
366
367 # Go through almost the same steps as the ones used on training
368
369 # Print 5 samples of validation data
370 df_val.sample(5)
371
372 # Feature selection
373
374 # Based on the five winning codes of the competition, NaN values and feature selection
    ↪ methods
375 df_val.drop(['FORCE_2020_LITHOFACIES_CONFIDENCE', 'FORMATION', 'SGR', 'DTS',
376              'ROPA', 'DCAL', 'MUDWEIGHT', 'RMIC', 'ROP', 'RXO'], axis=1, inplace=True)
377
378 # Sort by WELL and then by DEPTH_M
379 df_val = df_val.sort_values(['WELL', 'DEPTH_MD'])
380
381 # Print df shape (rows, columns)
382 print(df_val.shape)
383
384 # One-hot-encoding
385
386 # One-hot-encode the columns passed in the list
387 dummies = pd.get_dummies(df_val[['GROUP']], drop_first=True)
388 df_val = pd.concat([df_val.drop(['GROUP'], axis=1), dummies], axis=1)
389
390 # Print head of validation data
391 df_val.head(5)
392
393 # As validation set doesn't have two groups, we will add them as zero
394 df_val['GROUP_BOKNFJORD GP.'] = 0
395 df_val['GROUP_TYNE GP.'] = 0
396
397 # Reorder coluns according to training
398 df_val = df_val[df.columns]
399
400 # If there is an error here, it's because during one-hot-encoding you created a label
    ↪ that exists in train but does not exist in validation or vice-versa.
401 # This happens because either train has a sample (in the one-hot-encoded column) value
    ↪ that is not contained in validation (or vice-versa)
402 df_cols = df.columns
403 df_val_cols = df_val.columns
404 for i in range(len(df_val.columns)):
405     if df_cols[i] != df_val_cols[i]:
406         print('We have a problem, train and validation column have a different column
            ↪ order')
407 if len(df_cols) != len(df_val_cols):
408     print('We have a problem, either train has more columns than validation or
        ↪ vice-versa')
409
```

```
410 # Check if are any columns missing in validation dataset, if set is not empty, we may
    ↪  have same number of columns, but they have different names, and meanings, which is
    ↪  a problem
411 print(f'set(df_val.columns).difference(set(df.columns)):
    ↪ {set(df_val.columns).difference(set(df.columns))}')
412
413 # "Split" the data
414
415 # Separate the classes from the features
416 y_init_val = df_val['FORCE_2020_LITHOFACIES_LITHOLOGY']
417
418 # Do the classes mapping
419 y_init_val = y_init_val.map(lithology_numbers)
420
421 # Get sequence of wells, in order, for future use
422 X_well_val = df_val['WELL'].tolist()
423
424 # Remove classes and well feature from X
425 X_val_init = df_val.drop(['FORCE_2020_LITHOFACIES_LITHOLOGY', 'WELL'], axis=1)
426
427 # Print shapes
428 print(f'\nX_val_init.shape: {X_val_init.shape},\n\
429 y_init_val.shape: {y_init_val.shape}')
430
431 # Count classes, be aware that validation does not contain class 11 (basement)
432 print(f'\ny_init_val.value_counts(): {y_init_val.value_counts()}')
433
434 # Print the number of classes
435 print(f'{len(y_init_val.value_counts())}')
436
437 # Scaling
438
439 # Scale data using same scaler as used by the training dataset
440 X_val = scaler.transform(X_val_init)
441 X_val = pd.DataFrame(X_val, columns=X_val_init.columns)
442
443 # Impute missing data
444
445 # Impute missing data using IterativeImputer, it takes roughly 11 min to run this
446 X_val = imputer.transform(X_val)
447
448 # Transform data back to dataframe
449 X_val = pd.DataFrame(X_val, columns=X_val_init.columns)
450
451 # Print head of validation data
452 df_val.head(5)
453
454 # Format data into RNN format
455
```

```
456 # Return well and label columns
457 Xy_val = X_val.copy()
458 Xy_val['WELL'] = X_well_val
459
460 # One-hot-encode the label column
461 y_val = pd.get_dummies(y_init_val)
462
463 # As the validation set doesn't have the label 11 in it, we will include it
464 y_val[11] = 0
465
466 # Reinsert well to enable padding logic
467 y_val['WELL'] = X_well_val
468
469 # Convert the data to Recurrent Neural Netowrk (RNN) format
470 X_val = create_rnn_matrix(Xy_val)
471 y_val = create_rnn_matrix(y_val)
472
473 # Print shapes
474 print(f'\nX_val.shape: {X_val.shape}\n\
475 y_val.shape: {y_val.shape}')
476
477 # Making predictions
478
479 # As we will be predicting labels for all samples, including the padded ones, we will
    ↪  have to disconsider the rows that were padded.
480 # The rows to keep will have 1 as value and the rows to remove will have 0 as value
481 rows_to_keep_val = []
482 n_max_samples_val = Xy_val['WELL'].value_counts()[0]
483 for well in Xy_val['WELL'].unique():
484     n_samples_val = len(df_val.loc[df_val['WELL'] == well])
485     n_pad_rows_val = n_max_samples_val - n_samples_val
486     for i in range(n_samples_val):
487         rows_to_keep_val.append(1)
488     for i in range(n_pad_rows_val):
489         rows_to_keep_val.append(0)
490 len(rows_to_keep_val)
491
492 # Print rows to keep, it should have same length as X_val.shape[0] * X_val.shape[1]
493 print(f'len(rows_to_keep_val): {len(rows_to_keep_val)}')
494 print(f'X_val.shape[0] * X_val.shape[1]: {X_val.shape[0] * X_val.shape[1]}')
495
496 # Make predictions
497 yhat_val = np.argmax(model.predict(X_val), axis=-1)
498 y_pred_temp_val = yhat_val.ravel()
499
500 # Print predictions shape, it should have same length as len(rows_to_keep_val)
501 print(f'\ny_pred_temp_val.shape: {y_pred_temp_val.shape}')
502 print(f'len(rows_to_keep_val): {len(rows_to_keep_val)}')
503
```

```
# Remove padded rows from predictions
y_pred_val = []
for i in range(len(rows_to_keep_val)):
    if rows_to_keep_val[i] == 1:
        y_pred_val.append(y_pred_temp_val[i])
len(y_pred_val)

# Print length of predictions, it should have same length as y_init_val.shape
print(f'\nlen(y_pred_val): {len(y_pred_val)}')
print(f'y_init_val.shape: {y_init_val.shape}')

# Get y_true
y_true_val = y_init_val.to_list()

# Convert to series
y_pred_val = pd.Series(y_pred_val)
y_true_val = pd.Series(y_true_val)

# Save y_pred_val and y_true_val to a file
np.save('y_pred_val.npy', y_pred_val)
np.save('y_true_val.npy', y_true_val)
# y_pred_val = np.load('y_pred_val.npy')
# y_true_val = np.load('y_true_val.npy')

# Calculate score
print('\nScore:', score(y_true_val, y_pred_val))
print('Accuracy:', metrics.accuracy_score(y_true_val, y_pred_val))

# Plot confusin matrix
cm = metrics.confusion_matrix(y_true_val, y_pred_val)
df_cm = pd.DataFrame(cm, range(cm.shape[0]), range(cm.shape[0]))
plt.figure(figsize=(15, 10))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='d', linewidths=0.5)

plt.title('Matriz de Confusão Conjunto de Validação', fontsize = 20)
plt.xlabel('Rótulo Previsto', fontsize = 15)
plt.ylabel('Rótulo Verdadeiro', fontsize = 15)

plt.show()
```

APÊNDICE B – ARTIGO

Neste apêndice será apresentado o artigo, referente ao presente trabalho, no formato SBC.

Aplicação de Aprendizado de Máquina na Classificação de Litofácies

Matheus Henrique Schaly[1]

[1]Departamento de Informática e Estatística - Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

matheus.schaly@grad.ufsc.br

Abstract. *Lithofacies classification is a task performed by geologists that consists of analyzing a series of electrical and physicochemical records obtained through sensors that run along the wall of a drilled well and, from these, identify which lithological units (lithofacies) characterize the environment of formation and the compositional aspects of the rocks. This work proposes an effective model of ML, including data manipulation, for the classification of lithofacies in geological wells. We believe that taking into account the sequence of sedimentary patterns can help in the classification disambiguation process. To achieve that, a modified version of a recurrent neural network (RNN) could be used.*

Resumo. *Classificação de litofácies é uma tarefa realizada por geólogos que consiste em analisar uma série de registros elétricos e físico-químicos obtidos através de sensores que percorrem a parede de um poço perfurado e, a partir das leituras destes, identificar que unidades litológicas (litofácies) caracterizam o ambiente de formação e os aspectos composicionais das rochas. O objetivo do trabalho é propor um modelo eficaz de aprendizado de máquina (do inglês ML), incluindo a parte de manipulação dos dados, para a classificação de litofácies em poços geológicos. Acreditamos que levar em consideração a sequência de padrões sedimentares possa ajudar no processo de desambiguação da classificação. Para isso poderia ser utilizado uma versão modificada de uma rede neural recorrente (do inglês RNN).*

1. Introdução

Há diversas definições existentes para o termo fácies. Definimos fácies como qualquer parte restrita não comparável de uma unidade estratigráfica projetada que exibe caráter significativamente diferente de outras partes da unidade [Moore 1949]. Biofácies são fácies identificadas por características paleontológicas (conteúdo fóssil) sem levar em conta o caráter litológico. Litofácies são fácies identificadas com base em características litológicas [Boggs 2001]. Usaremos litofácies como base de dados no presente trabalho.

A classificação de litofácies é atribuir uma classe de rocha a uma amostra específica com base nas características medidas. A fonte ideal para classificação de litofácies são amostras de núcleo de rochas extraídas de poços. No entanto, devido aos custos associados, nem sempre as amostras de núcleo podem ser obtidas. Além disso, o método convencional é um processo tedioso e demorado, pois consiste em classificar litofácies manualmente por intérpretes humanos. Portanto, um método para classificar fácies a partir de medidas indiretas (por exemplo, gerar perfis utilizando sensores presos

a cabo de aço) é necessário. Várias abordagens distintas para a questão da classificação de fácies utilizando dados de poços já foram propostas [Mandal and Rezaee 2019]. Neste trabalho será investigado um conjunto de 118 perfis de poços que possui 27 atributos e 12 classes.

Nos últimos anos, aprendizado de máquina (do inglês Machine Learning (ML)) se tornou uma ferramenta interdisciplinar cada vez mais importante, que avançou vários campos da ciência, como biologia, química, medicina e farmacologia. Especificamente, o método de rede neural profunda (do inglês DNN) encontrou ampla aplicação. Enquanto a geociência foi mais lenta na adoção, a bibliometria mostra adoção do aprendizado profundo (do inglês DL) em todos os aspectos da geociência [Dramsch 2020].

Aprendizado de máquina está profundamente enraizado na estatística aplicada, criando modelos computacionais que usam inferência e reconhecimento de padrões em vez de conjuntos explícitos de regras [Dramsch 2020]. Aprendizado de máquina é o campo de estudo que fornece aos computadores a capacidade de aprender sem serem explicitamente programados [Samuel 1959]. Aprendizagem supervisionada consiste na tarefa de um algoritmo de ML em aprender uma função que mapeia uma entrada para uma saída com base em exemplos de pares de entrada e saída [Russell and Norvig 2010]. Uma função é inferida a partir de dados de treinamento rotulados que consistem em um conjunto de exemplos de treinamento [Mohri et al. 2012].

O DL é uma forma de ML que permite que os computadores aprendam com a experiência e entendam o mundo em termos de uma hierarquia de conceitos. A hierarquia de conceitos permite que o computador aprenda conceitos complicados construindo-os a partir de outros mais simples [Goodfellow et al. 2016].

Recentemente, as técnicas de DL foram desenvolvidas e amplamente adotadas para extrair informações de vários tipos de dados. Considerando as diferentes características dos dados de entrada, existem vários tipos de arquiteturas para DL, como a rede neural recorrente (do inglês Recurrent Neural Network (RNN)), rede neural convolucional (do inglês CNN), e DNN. Diferentemente da DNN, a RNN pode lidar com dados que possuem informações temporais. Portanto, em áreas de pesquisa que contêm dados sequências, como texto, áudio e vídeo, RNNs são dominantes. Contudo, RNNs são incapazes de aprender as informações relevantes dos dados de entrada quando o intervalo de entrada é grande. A diferença entre uma RNN e uma LSTM está na função de portão. Portanto, ao introduzirmos funções de portão na estrutura da célula de uma RNN passamos a ter uma LSTM. Com a LSTM podemos lidar bem com o problema das dependências de longo prazo. Desde a introdução da RNN quase todos os resultados interessantes baseados em RNNs foram alcançados pela LSTM. A LSTM se tornou o foco do DL [Yu et al. 2019].

O problema de classificação automática de litofácies deve ser explorado a fim de diminuir os custos envolvidos na classificação manual de litofácies. Existem competições envolvendo a classificação automática acurada de perfis de poços por meio de algoritmos de ML. No presente trabalho avaliaremos a competição, já encerrada, chamada Force 2020 Machine Learning Competition [1] [2020], na qual o modelo de ML vencedor da

[1]Link do site da competição: https://xeek.ai/challenges/force-well-logs/overview, Link do GitHub da competição: https://github.com/bolgebrygg/Force-2020-Machine-Learning-

competição utilizou o algoritmo XGBoost. Nossa solução ao problema de classificação de litofácies utilizará um algoritmo de ML supervisionado. Acreditamos que a sequência de padrões sedimentares possa ajudar no processo de classificação. Portanto, podemos, mais especificamente, criar uma nova topologia de LSTM que venha a considerar este aspecto sequencial do nosso banco de dados. Além disso, criaremos um *pipeline* para a manipulação dos dados para organizar e melhorar os dados de entrada ao modelo.

2. Fundamentação Teórica

Aqui serão apresentados os conceitos principais para a realização deste trabalho. Começaremos apresentando a definição de fácies sedimentares, litofácies, e uma breve descrição dos atributos utilizados para a classificação de litofácies. Em seguida, será feito uma breve introdução aos conceitos de IA, ML, DL, RNN e LSTM.

2.1. Litofácies

Existem várias maneiras de descrever ou designar fácies sedimentares. Ao observar as características físicas (ou litológicas) primárias, é possível reconhecer litofácies. Os atributos biológicos (ou mais corretamente, paleontológicos) - os fósseis - definem biofácies. Ambos são o resultado direto da história deposicional da bacia. Ao atribuir modos de origem a diferentes fácies (ou seja, interpretando as litofácies ou biofácies), pode-se visualizar um sistema genético de fácies [Britannica].

2.2. Recurrent Neural Network

As RNNs têm sido amplamente adotadas em áreas de pesquisa relacionadas com dados sequenciais, como texto, áudio e vídeo. Nas RNNs, as camadas recorrentes ou camadas ocultas consistem em células recorrentes cujos estados são afetados tanto pelos estados passados quanto pela entrada atual a partir de conexões de *feedback*. Normalmente RNNs são redes que consistem em células recorrentes padrão, como células sigma (equação 1) e células tanh (equação 2). A Figura 1 mostra um esquema da célula sigma recorrente padrão. A expressão matemática da célula sigma recorrente padrão é definida pela equação 3, onde x_t, h_t, and y_t denotam a entrada, a informação recorrente e a saída da célula no tempo t, respectivamente; w_h e w_x são os pesos; e b é o viés [Yu et al. 2019].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2}$$

$$\begin{aligned} h_t &= \sigma(w_h h_{t-1} + w_x x_t + b) \\ y_t &= h_t \end{aligned} \tag{3}$$

competition/tree/master/lithology_competition

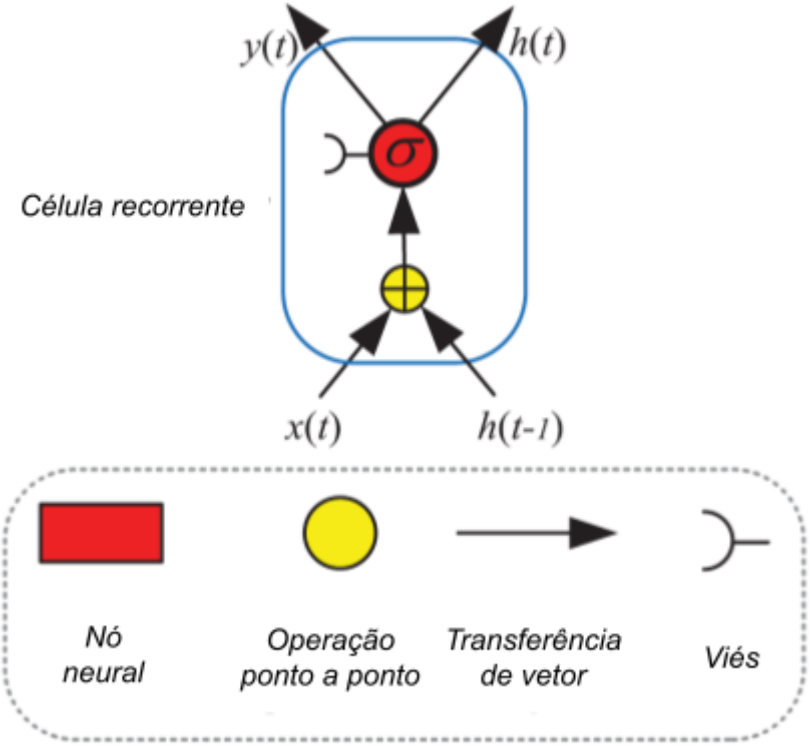


Figure 1. Esquema da célula sigma recorrente padrão.

As células recorrentes padrão obtiveram algum sucesso em alguns problemas. No entanto, RNNs que consistem em células sigma ou células tanh são incapazes de aprender as informações relevantes dos dados de entrada quando o intervalo de entrada é grande: à medida que o intervalo entre as entradas relacionadas aumenta, é difícil aprender as informações de conexão [Yu et al. 2019].

Para lidar com o problema das "dependências de longo prazo", Hochreiter e Schmidhuber (1997) propuseram a célula LSTM. Desda sua introdução, quase todos os resultados empolgantes baseados em RNNs foram alcançados pela LSTM. A LSTM se tornou o foco do DL. A capacidade de memorização da célula recorrente padrão foi aumentada ao introduzir um "portão" na célula. Desde este trabalho pioneiro, as LSTMs foram modificadas e popularizadas por muitos pesquisadores. As variações incluem LSTM sem um portão de esquecimento, LSTM com um portão de esquecimento e LSTM com uma conexão de olho mágico [Yu et al. 2019]. Em seguida, apresentamos o modelo LSTM com um portão de esquecimento.

A Figura 2 apresenta as conexões internas de uma LSTM com portão de esquecimento. Com base nas conexões mostradas na Figura 2, a célula LSTM pode ser expressa matematicamente pela equação 4, onde c_t denota o estado da LSTM, w_i, $w_{\tilde{c}}$, w_o, w_f são os pesos do portão de entrada, da célula de memória, do portão de saída e do portão de esquecimento respectivamente, e o operador "$\cdot$" denota a multiplicação ponto a ponto de dois vetores. Ao atualizar o estado da célula, o portão de entrada pode decidir quais novas informações podem ser armazenadas no estado da célula, o portão de saída decide quais informações podem ser enviadas com base no estado da célula e o portão de esquecimento pode decidir quais informações serão descartadas do estado da célula. Quando o valor do

portão de esquecimento, f_t, é 1, ele mantém essa informação, por outro lado, um valor de 0 significa que o portão se livra de todas as informações [Yu et al. 2019].

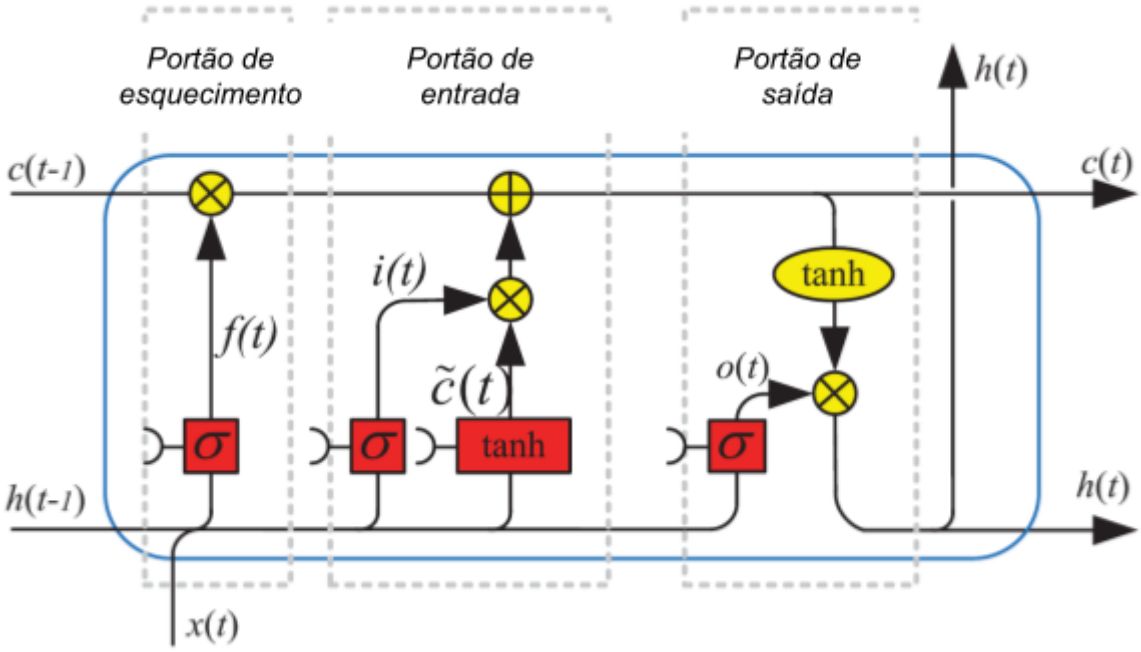


Figure 2. Arquitetura da LSTM com um portão de esquecimento..

$$
\begin{aligned}
f_t &= \sigma(w_{fh}h_{t-1} + w_{fx}x_t + b_f) \\
i_t &= \sigma(w_{ih}h_{t-1} + w_{ix}x_t + b_i) \\
\tilde{c}_t &= tanh(w_{\tilde{c}h}h_{t-1} + w_{\tilde{c}x}x_t + b_{\tilde{c}}) \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \\
o_t &= \sigma(w_{oh}h_{t-1} + w_{ox}x_t + b_o) \\
h_t &= o_t \cdot tanh(c_t)
\end{aligned}
\tag{4}
$$

3. Trabalho relacionado

No trabalho *FaciesNet: Machine Learning Applications for Facies Classification in Well Logs* apresentado por [Jaikla et al. 2019] é desenvolvido um modelo de classificação de fácies usando redes neurais recorrentes bidirecionais (do inglês BRNN) que incorporam sequências de fácies na previsão. Além de BRNNs, experimentou-se outra arquitetura adicionando camadas de decodificação e codificação de redes neurais convolucionais profundas (do inglês DCNN) para extrair informações latentes antes de alimentá-las nas camadas de BRNNs.

Para a arquitetura de BRNN os experimentos incluíram o treinamento do conjunto de dados em modelos com 1, 2 e 3 camadas de BRNNs com 16, 32, 64 e 128 estados ocultos de unidades recorrentes bloqueadas (do inglês GRU).

Já para a arquitetura com DCNNs e BRNN, a arquitetura que tem a maior acurácia e acurácia equilibrada no conjunto de teste consiste em 5 camadas de codificação e

decodificação DCNNs seguidas por 2 camadas de BRNNs com 128 estados ocultos usando a *dice loss function*. Tal arquitetura foi chamada de FaciesNet (Figura 3).

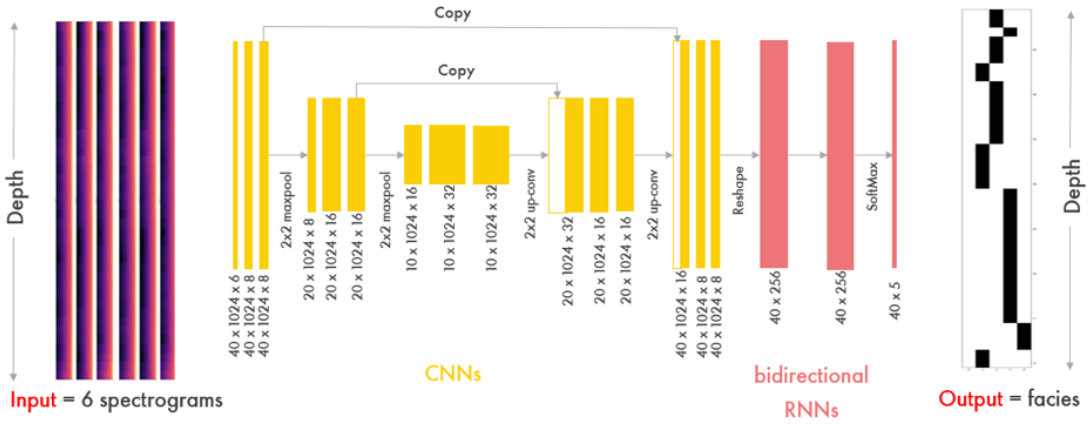


Figure 3. Arquitetura FaciesNet.

O conjunto de dados possui 4 poços contendo um total de 170 amostras. O conjunto de dados contém 5 tipos de litofácies: arenito cimentado, heterolítico, lamito, arenito limpo, e arenito sujo. Os 6 atributos do conjunto de dados são: GR, VSH, DEN, DTC, DTS, e NEU.

Foi realizado uma análise comparandos os modelos de Naïve Bayes, árvore de decisão, floresta aleatória, BRNN e FaciesNet, com base em acurácia e acurácia equilibrada (Figura 4). Além disso, uma análise comparativa entre Naïve Bayes e FaciesNet, com base em precisão, *recall* e *F1-score* são mostradas na Figura 5.

Model	Accuracy	Balanced accuracy
Naive Bayes	83.45%	56.97%
Decision Tree	83.69%	51.55%
Random Forest	84.88%	51.21%
BRNNs	64.11%	24.43%
FaciesNet	74.85%	40.01%

Figure 4. Acurácia e acurácia balanceada da rede.

4. Desenvolvimento

Neste capítulo, explicamos como os conceitos teóricos apresentados na seção de fundamentação teórica são colocados em prática na elaboração do método proposto[2]. Primeiramente demonstramos o ambiente utilizado na implementação, tais como linguagem e bibliotecas. Em seguida, apresentamos o conjunto de dados utilizados como entrada para o modelo proposto. Por fim, mostramos as métricas utilizadas para a avaliação dos modelos produzidos.

[2]O código pode ser encontrado em https://github.com/MatheusSchaly/TCC

FaciesNet	Precision	Recall	F1 score
Cemented sandstone	0.8125	0.2718	0.4063
Heterolithic	0.1895	0.2000	0.1956
Mudstone	0.6209	0.5125	0.5621
Sandstone	0.8485	0.9133	0.8797
Dirty sandstone	0.1320	0.1029	0.1157
Naive Bayes	Precision	Recall	F1 score
Cemented sandstone	0.8913	0.8542	0.8723
Heterolithic	0	0	0
Mudstone	0.5745	1	0.7298
Sandstone	0.9315	0.9401	0.9358
Dirty sandstone	0	0	0

Figure 5. Comparação de precisão, *recall*, *F1-score*, da FaciesNet com Naïve Bayes.

4.1. Ambiente

A implementação foi desenvolvida em Python, utilizando como IDE o Jupyter Notebook. As principais bibliotecas utilizadas foram *numpy* e *pandas* para a manipulação e visualização dos dados; *sklearn* para o pré-processamento dos dados; *keras* para a construção da rede neural; e *matplotlib* para a construção de gráficos.

4.2. Conjunto de Dados

Neste trabalho usaremos o conjunto de dados fornecido pela competição Force 2020 Machine Learning Competition. Os dados são provenientes da costa da Noruega. O conjunto de dados é separado em conjunto de treino, teste, e validação. O conjunto de treino foi liberado no início da competição. O conjunto de teste era utilizado para computar o placar dos participantes da competição durante a competição. Para tal, os participantes submetiam suas predições sobre o conjunto de teste, para a competição, e a competição retornava a pontuação do participante. Por fim, o conjunto de validação foi liberado apenas após o final da competição, para que a competição pudesse determinar o vencedor.

O conjunto de dados consiste em um total de 1.429.694 exemplos de litofácies, sendo que 1.170.511 são pertencentes ao conjunto de treino, 136.786 ao conjunto de teste, e 122.397 ao conjunto de validação. Há um total de 118 perfis de poços, 98 para o conjunto de treino, 10 para o conjunto de teste, e 10 para o conjunto de validação. O conjunto de dados de treino e de validação possuem 27 atributos de registro de poços, além de um atributo de confiança de interpretação, e outro atributo contendo a classe das litofácies, totalizando 29 atributos. Visto que o conjunto de teste não possui a confiança da interpretação nem a classe das litofácies, ele possui apenas 27 atributos. Portanto, dado que o conjunto de teste não possui as classes para as litofácies e que a competição já se encerrou, tal conjunto de dados não será utilizado no presente trabalho.

O conjunto de dados possui vários atributos com inúmeros valores ausentes. Por exemplo, o atributo SGR, que é o atributo que possui a maior quantidade de valores ausentes, possui 1.101.158 valores ausentes no conjunto de treino e 122.397 valores

ausentes no conjunto de validação. Representando uma ausência de 94.07% de dados no conjunto de treino e 89.48% de dados no conjunto de validação. Considerando os 27 atributos do conjunto de dados de treino e de validação, temos um total de 31.603.797 e 3.304.719 valores respectivamente. Desse total, 10.245.502 são valores ausentes no conjunto de treino e 995.942 são valores ausentes no conjunto de validação. Portanto, 32,42% dos valores são valores ausentes no conjunto de treino e 30,14% dos valores são valores ausentes no conjunto de validação. Os atributos ausentes do conjunto de treino podem ser vistos abaixo.

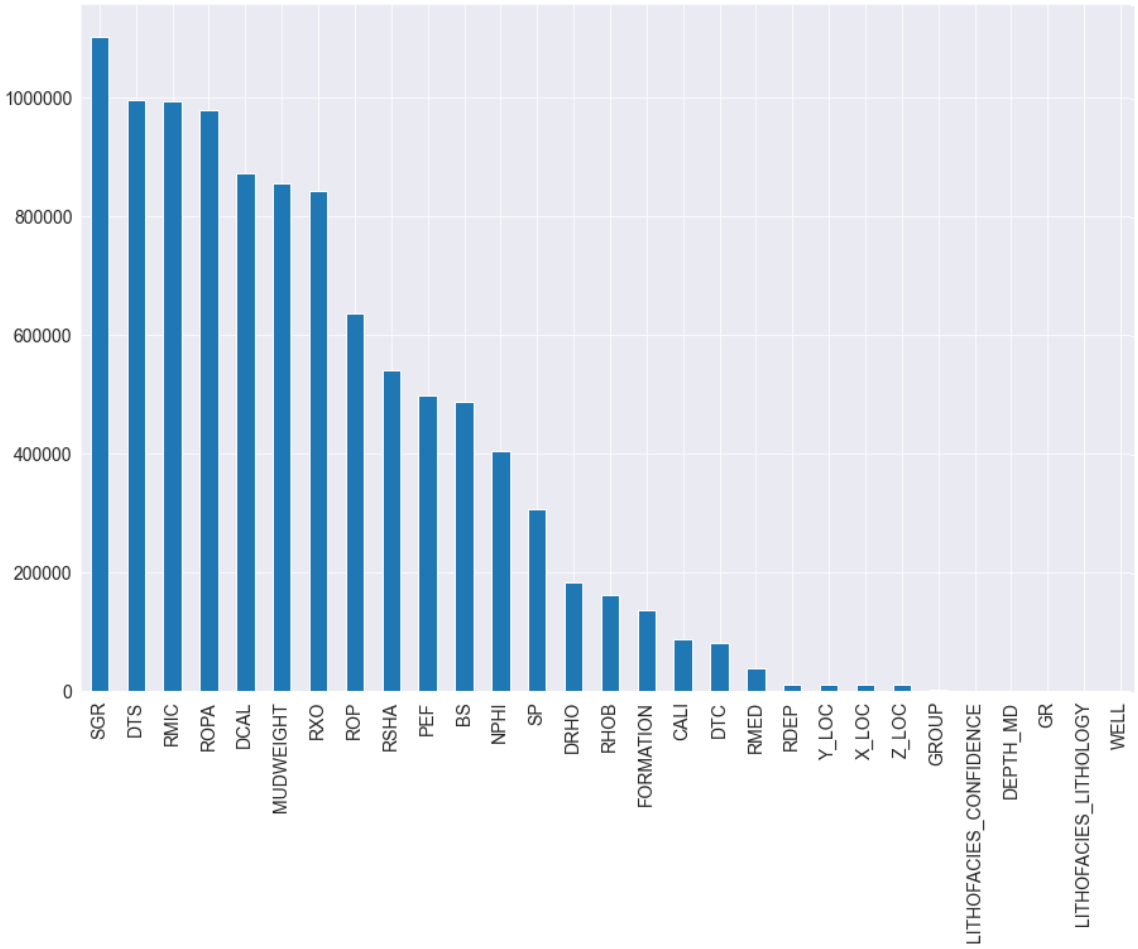


Figure 6. Valores ausentes no conjunto de treino.

Temos também um grande desbalanceamento de classes (Figura 8). Considerando o conjunto de treino, temos que a classe xisto representa a 61.5802% das classes. No outro extremo, temos que a classe embasamento representando apenas 0.0088% das classes.

4.3. Pré-Processamento dos Dados

As etapas do pré-processamento dos dados (Figura 8) foram:

1. Carregamento dos dados: fazemos o carregamento dos dados de treinamento da competição em um pandas *dataframe*.
2. Seleção dos atributos: aplicamos diferentes métodos de seleção de atributos para reduzir o efeito do *curse of dimensionality*, consequentemente diminuindo o risco

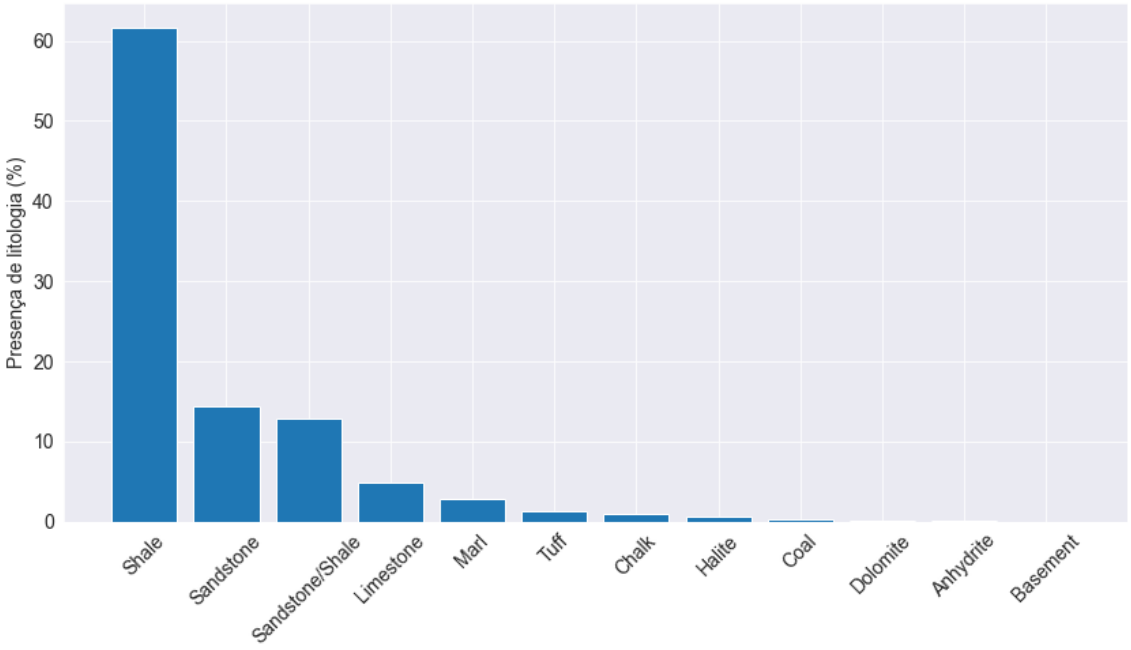


Figure 7. Desbalanceamento de classes no conjunto de treino.

de *overfitting*, aumentando a velocidade do método e sua acurácia. Utilizamos os seguintes métodos de seleção de atributos:

- Métodos de seleção de atributos limite de variância, teste qui-quadrado, limiar de correlação, *backward elimination*, eliminação de recurso recursivo, e LassoCV.
- Quantidade de valores faltantes nos atributos.
- Comparação com a seleção dos atributos realizados pelos top 5 participantes da competição. Os atributos descartados foram: SGR, DTS, ROPA, DCAL, RMIC, ROP, RXO, FORCE_2020_LITHOFACIES_CONFIDENCE, FORMATION e MUDWEIGHT. Com isso ficamos com 19 atributos.

3. *One hot encoding*: dado que o modelo espera atributos numéricos, usamos o *one hot encoding* para converter as colunas categóricas em colunas numéricas. O processo de *one hot encoding* consiste em criar um atributo próprio para cada um dos valores distintos do atributo categórico e excluir o atributo categórico. Estas novas colunas terão, em suas linhas, um único valor 1 no atributo correspondente ao atributo categórico anterior e o restante dos valores é 0. Aplicamos o *one hot encoding* no atributo GROUP, o qual possuía 13 valores distintos, com isso passamos de 19 atributos para 31 atributos.
4. Divisão dos dados: dos 98 poços disponíveis para treino, realizamos uma seleção aleatória de 73 desses poços para treino e 25 para teste. A divisão dos dados fica nas seguintes 4 variáveis *X_train*, *X_test*, *y_train* e *y_test*. Onde *X_train* e *X_test* são os atributos do conjunto de treino e de teste respectivamente, e *y_train* e *y_test* são as classes do conjunto de treino e de teste respectivamente. Tínhamos 31 atributos e passamos para 29 atributos ao remover os atributos FORCE_2020_LITHOFACIES_LITHOLOGY e WELL.

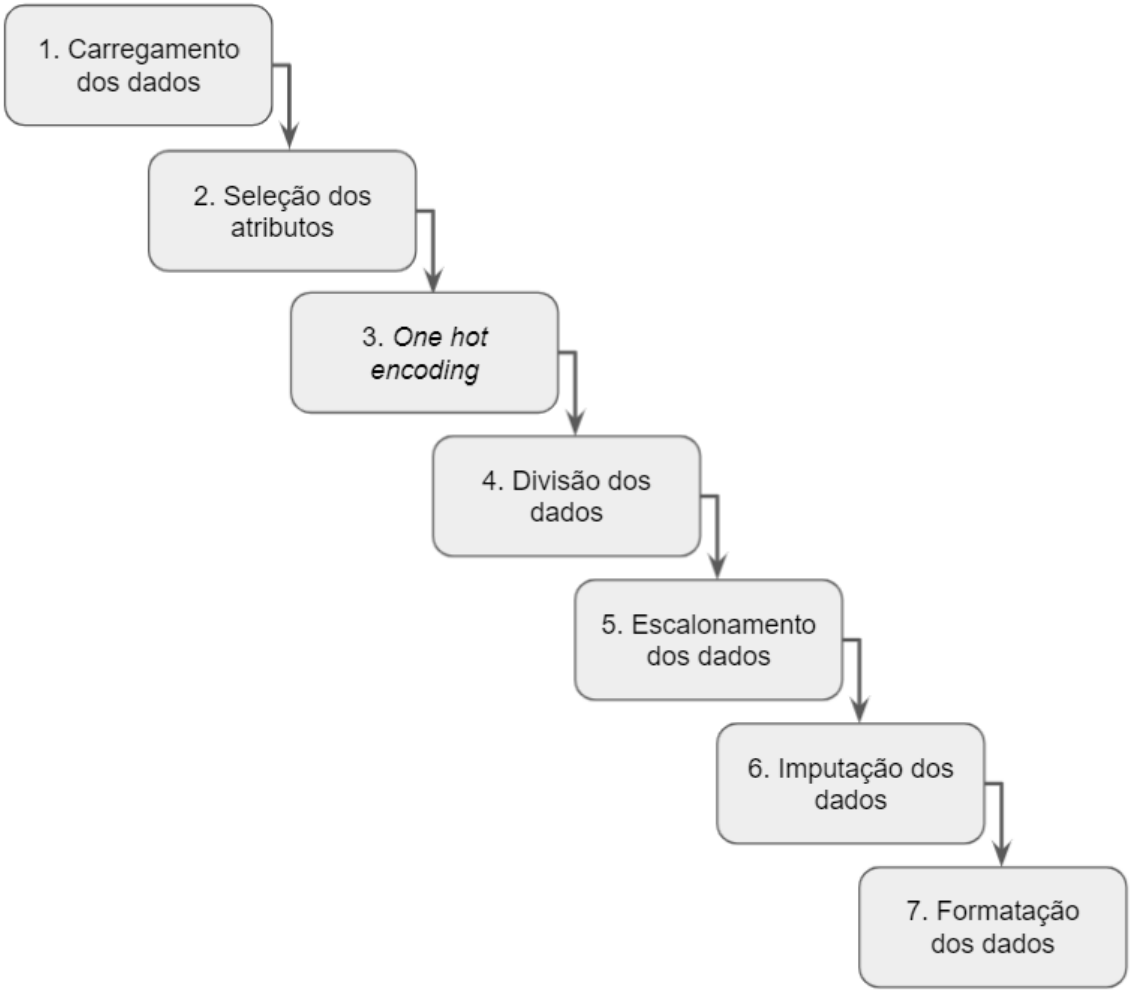


Figure 8. Etapas de pré-processamento dos dados.

5. Escalonamento dos dados: Para acelerar a velocidade do modelo, assim como torná-lo mais robusto, usamos o *standard scaler* nos valores dos atributos, fazendo com que seus intervalos de valores variem entre aproximadamente -3 até aproximadamente +3.
6. Imputação (atribuição/preenchimento) dos dados: Como temos vários atributos com dados ausentes, utilizamos o modelo *Bayesian Ridge* para imputar os valores ausentes de um atributo baseado nos valores dos demais atributos.
7. Formatação dos dados: o formato de entrada necessário para nossa LSTM consiste em uma matriz 3D. As variáveis *X_train* e *X_test* possuem as dimensões: poços, exemplos de camadas do poço, e atributos. Já as variáveis *y_train* e *y_test* possuem as dimensões: poços, exemplos de camadas de poços, e categorias no formato *one hot encoded*. Contudo, para que a rede aceite a entrada, todos os poços devem ter a mesma quantidade de camadas. Para que possamos ter a mesma quantidade de camadas em cada poço, pegamos o poço com a maior quantidade de camadas e fizemos com que todos os poços tenham a mesma quantidade de camada através do *padding* com zeros ao final de cada um dos poços. Tal processo de *padding* foi feito tanto para os conjuntos de treino *X_train* e *y_train*, assim como para os

conjuntos de teste *X_test* e *y_test*. Com isso, as dimensões finais dos conjuntos de dados são:

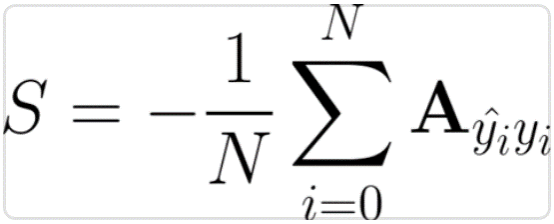
- X_train: (73, 25131, 29), onde 73 é o número de poços, 25131 é a quantidade de camadas do poço com maior número de camadas, X é o número de atributos utilizados.
- X_test: (25, 23879, 29), onde 25 é o número de poços, 23879 é a quantidade de camadas do poço com maior número de camadas, X é o número de atributos utilizados.
- y_train: (73, 25131, 12), onde 73 é o número de poços, 25131 é a quantidade de camadas do poço com maior número de camadas, e 12 é o número de classes no formato *one hot encoded*.
- y_test: (25, 23879, 12), onde 73 é o número de poços, 23879 é a quantidade de camadas do poço com maior número de camadas, e 12 é o número de classes no formato *one hot encoded*.

4.4. Métrica de Avaliação

A competição possui sua própria métrica de avaliação, a qual chamaremos de *score*. Em vez de penalizar cada previsão errada das litofácies, a competição decidiu usar uma matriz de penalidade customizada (Figura 9) derivada da entrada média de uma amostra representativa de geocientistas. Isso permite que previsões petrofisicamente irracionais sejam avaliadas por um grau de ”erro” (Figura 10). A Figura 10 apresenta o *score*, onde A é a matriz de penalidade abaixo, N é o número de amostras, $\hat{y}_i$ é o rótulo litológico verdadeiro e y_i é o rótulo litológico previsto. Esta métrica de avaliação não é utilizada durante o treinamento da rede, mas apenas no modelo final para mensurar sua qualidade em comparação com os outros modelos da competição.

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 9. Matriz de penalidade.



$$S = -\frac{1}{N}\sum_{i=0}^{N} \mathbf{A}_{\hat{y}_i y_i}$$

Figure 10. Métrica de avaliação.

5. Experimentos e Resultados

A estratégia para a realização dos experimentos foi comparar diferentes arquiteturas de LSTM avaliando o *score* sobre o conjunto de teste (*X_test*, *y_test*). Em seguida, selecionamos o melhor modelo, treinamos ele novamente só que agora usando tanto o conjunto de treino quanto o de teste, e avaliamos sua qualidade usando o *score*. Os passos descritos na seção de pré-processamento dos dados foram realizados em todos os experimentos.

5.1. Modelos

Apresentamos aqui as características dos principais modelos treinados durante os experimentos. Uma arquitetura de rede neural pode possuir diversas camadas. Cada camada possui suas próprias características e parâmetros, as camadas utilizadas nas arquiteturas foram:

- *Biderectional*: Conectam duas camadas ocultas de direções opostas à mesma saída. Com isso a camada de saída pode obter informações dos estados do passado (para trás) e futuros (para frente) simultaneamente.
- LSTM: trata as camadas de forma sequencial. Dado que a LSTM está configurada para retornar sequências, a sua saída possui um formato 3D.
- *Dropout*: define aleatoriamente as unidade de entrada para 0 durante o processo de treinamento, possivelmente ajudando a prevenir o *overfitting* do modelo. Sua taxa de entradas para serem descartadas é passado como argumento.
- *Batch normalization*: aplica uma transformação que mantém a média de saída próxima a 0 e o desvio padrão da saída próximo a 1, possivelmente contribuindo para a velocidade do processo de treinamento e com a regularização, e reduzindo o erro de generalização.
- *Dense*: camada padrão de modelos de rede neural, ela serve tanto para possivelmente contribuir com a acurácia da rede quanto para mudar as dimensões dos vetores. Foi utilizado tanto a função de ativação ReLU quanto a *softmax*. A função de ativação *softmax* é escolhida para fazer com que o vetor de saída possua valores entre 0 e 1 e que sua soma seja igual a 1. Com isso podemos escolher o índice de maior valor do vetor o qual corresponde a classe da amostra da camada litológica em questão.
- *Time distributed*: trata o formato de saída 3D da camada LSTM para que a camada *dense* possa utilizá-la.
- *Masking*: para cada passo de tempo fornecido na entrada da rede, se todos os valores naquele intervalo de tempo forem iguais ao parâmetro *mask_value*, o passo de

tempo será mascarado (pulado) em todas as camadas abaixo. Isto é, dado que utilizamos *padding* nos nossos dados de entrada, a camada *masking* irá desconsiderar o *padding* durante o treinamento da rede.

Além disso, também temos parâmetros para configurar no momento da compilação e treinamento do modelo, tais como:

- Otimizadores: algoritmos que otimizam a conversão do modelo. O otimizador utilizado nos modelos foi o Adam.
- Taxa de aprendizado: indica a velocidade de atualização dos pesos durante o treinamento da rede. Foi configurado para 0.002 em todos os modelos.
- Função de perda: utilizamos a *categorical crossentropy*, que é a função utilizada para problemas de classificação de multi classes, fazendo com que cada amostra de camada seja pertencente a apenas uma classe.
- Métrica de avaliação: utilizamos a métrica acurácia, que também é utilizada para tratar problemas de classificação. Tal métrica leva em consideração os valores de verdadeiro positivo, verdadeiro negativo, falso positivo e falso negativo.
- Dados de entrada: utilizamos os conjuntos de dados de treino *X_train* e *y_train* para realizar o treinamento do modelo.
- Dados de validação: utilizamos os conjuntos de dados de teste *X_test* e *y_test* para a validação do modelo.
- Épocas: número de épocas para treinar o modelo. Uma época é uma iteração sobre todos os dados de entrada fornecidos. Utilizamos 150 épocas em todos os modelos.
- Embaralhamento: se deve-se embaralhar os dados de treinamento antes de cada época. Visto que a sequência dos dados de treinamento é relevante, tal booleano foi configurado para falso.
- Tamanho do lote: Número de amostras por atualização do gradiente da rede.

Por fim, temos o parâmetro *callback*, que é passado como argumento para o treinamento do modelo, e que possui as seguintes características:

- *Early stopping*: encerra o treinamento quando uma métrica monitorada parar de melhorar durante um certo número de épocas. A métrica monitorada é a perda do conjunto de validação. O número de épocas para esperar antes de encerrar o treinamento do modelo é chamada de paciência.

A Tabela 7 abaixo apresenta as arquiteturas dos modelos treinados, onde P significa paciência, E significa épocas e B significa tamanho do lote. A função de ativação utilizada na LSTM foi a tangente hiperbólica (TanH).

Após alguns testes de modelos prévios, chagamos ao Modelo 1. O Modelo 1 possui a arquitetura mais simples dentre os demais modelos apresentados, visto que ele possui apenas 64 neurônios na LSTM e apenas uma camada *dense*. No Modelo 2 incluímos uma camada *dense* com 30 neurônios. No modelo 3 aumentamos a quantidade de neurônios para 128. O Modelo 4 não possui uma camada *bidirectional*, possui uma camada de *dropout*, *batch normalization*, e 3 camadas *dense*. O Modelo 5 é idêntico ao Modelo 1, exceto pelo fato de que o Modelo 5 não possui a camada *bidirectional*. Após tais experimentos, o Modelo 1 foi o que apresentou melhores resultados, mas ainda havia a possibilidade de tal modelo melhorar seus resultados caso ele fosse treinado por mais

épocas. Portanto, no Modelo 6, também criamos um modelo idêntico ao Modelo 1, exceto que, no Modelo 6, não definimos um *early stopping*, consequentemente o Modelo 6 foi executado por 150 épocas.

P = 25, E = 150, B = 1
Masking
Bidirectional (LSTM 64 TanH)
Dropout 0.3
Time Distr. (Dense 12 Softmax)

Table 1. Modelo 1

P = 25, E = 150, B = 1
Masking
Bidirectional (LSTM 64 TanH)
Dropout 0.3
Time Distr. (Dense 30 ReLU)
Time Distr. (Dense 12 Softmax)

Table 2. Modelo 2

P = 25, E = 150, B = 1
Masking
Bidirectional (LSTM 128 TanH)
Dropout 0.3
Time Distr. (Dense 12 Softmax)

Table 3. Modelo 3

P = 25, E = 150, B = 1
Masking
LSTM 200 TanH
Dropout 0.5
Batch Normalization
Time Distr. (Dense 50 ReLU)
Time Distr. (Dense 30 ReLU)
Time Distr. (Dense 12 Softmax)

Table 4. Modelo 4

P = 25, E = 150, B = 1
Masking
LSTM 64 TanH
Dropout 0.3
Time Distr. (Dense 12 Softmax)

Table 5. Modelo 5

P = ∞, E = 150, B = 1
Masking
Bidirectional (LSTM 64 TanH)
Dropout 0.3
Time Distr. (Dense 12 Softmax)

Table 6. Modelo 6

Table 7. Arquiteturas dos modelos.

5.2. Comparação entre modelos

Na Figura 17 apresentamos os gráficos de cada modelo, mostrando a evolução da acurácia do conjunto de treino, acurácia do conjunto de teste, perda do conjunto de treino e perda do conjunto de teste ao longo das épocas de treino.

Todos os modelos possuem resultados similares. Vale ressaltar que, o Modelo 1 e o Modelo 6 são idênticos exceto pelo falo de que o Modelo 6 foi treinado por 150 épocas enquanto o Modelo 1 sofreu *early stopping* na época 33. Como veremos mais adiante, o Modelo 1 possui melhor *score* se comparado ao Modelo 6. Isso é devido ao *overfitting* que começa a ocorrer no Modelo 6 após cerca de 40 épocas. *Overfitting* ocorre quando o modelo torna-se muito bom em prever dados que foram incluídos durante seu treinamento, mas não é tão bom ao classificar dados para os quais não foi treinado.

Na Figura 24 apresentamos a matriz de confusão de cada modelo avaliado sobre o conjunto de teste. O mapeamento do identificador e litologia das matrizes de confusão pode ser visto na Tabela 8.

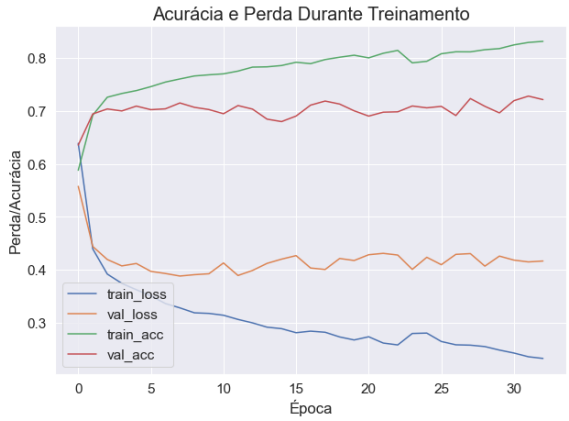


Figure 11. Modelo 1

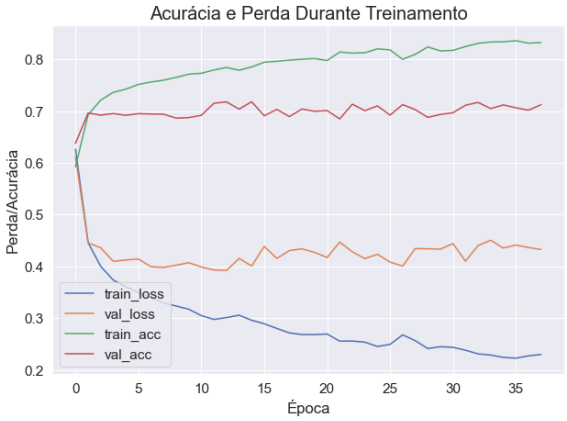


Figure 12. Modelo 2

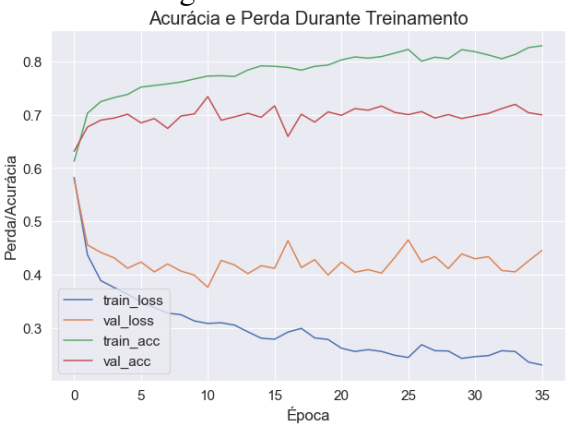


Figure 13. Modelo 3

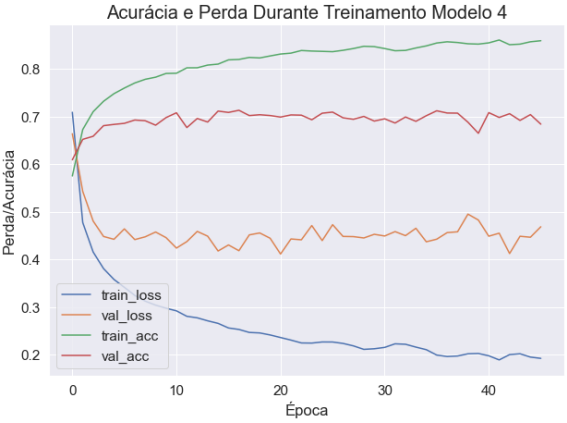


Figure 14. Modelo 4

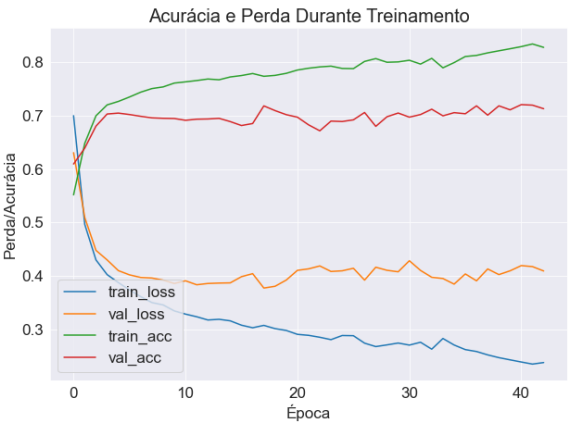


Figure 15. Modelo 5

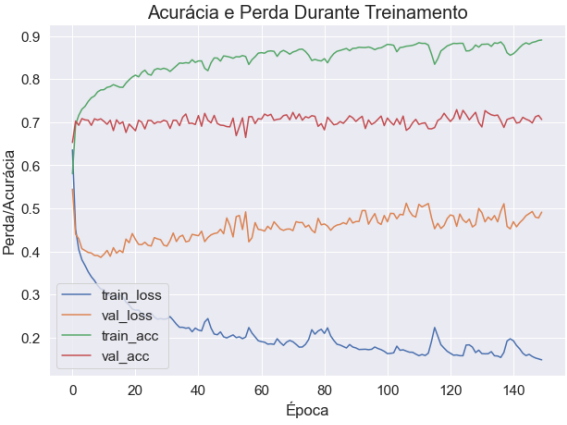


Figure 16. Modelo 6

Figure 17. Comparação de acurácia e perda, durante treinamento, entre os modelos.

Notamos que a classe mais prevista pelo modelo é a classe xisto, o que faz sentido, dado que tal classe representa 59.51% das classes do conjunto de teste. Entretanto, observamos que todos os modelos acabam erroneamente prevendo muitas amostras da classe xisto para amostras que na verdade pertencem a classe arenito/xisto. Isso ocorre pois a classe xisto é bem similar a classe arenito/xisto. Por conta da similaridade entre tais

classes, esse tipo de erro é pouco penalizado pela métrica *score*, como pode ser visto na Figura 9. Observamos também que a classe embasamento não está presente no conjunto de teste, isso se deve a sua pouquíssima quantidade de exemplos no nosso conjunto de dados. Os modelos corretamente não preveem a classe embasamento para nenhuma das amostras. Também vemos que praticamente o único modelo que prevê a classe dolomita é o Modelo 6. Isso pode ter ocorrido pois a classe dolomita é percentualmente aproximadamente duas vezes mais presente no conjunto de treino do que no conjunto de teste (0.1669% das amostras no conjunto de treino e 0.077318% no conjunto de teste), e sabemos que o Modelo 6 é o único modelo que sofreu *overffiting*.

Na Tabela 9, temos a quantidade de acertos, quantidade de amostras do conjunto de teste e a porcentagem de acerto por classe do Modelo 1. Com tal tabela podemos ver mais claramente que o modelo tem dificuldade em identificar dolomitas (0.0%), arenito/xisto (22.96%), giz (39.52%) e carvão (40.76%). Por outro lado, o modelo possui mais de 80% de acerto ao identificar embasamento (100%), halita (98.35%), anidrita (97.56%) e xisto (82.22%).

Identificador	Litologia
0	Arenito
1	Arenito/Xisto
2	Xisto
3	Marga
4	Dolomita
5	Calcário
6	Giz
7	Halita
8	Anidrita
9	Tufo
10	Carvão
11	Embasamento

Table 8. Mapeamento entre identificador e litologia das matrizes de convolução.

Por fim apresentamos, na Tabela 10, a qualidade de cada modelo utilizando o *score* e a acurácia sobre o conjunto de teste e validação.

O Modelo 1 possui a maior acurácia (72.19%) e *score* (-0.7006) no conjunto de teste, porém possui o segundo pior resultado no conjunto de validação. Por outro lado, o Modelo 6 possui a maior acurácia (77.24%) e *score* (-0.5667) no conjunto de validação, e fica em segundo no conjunto de teste. Essa disparidade entre a qualidade no conjunto de teste e no conjunto de validação também ocorreu durante a competição, como pode ser visto na Tabela 11. Dado que não teríamos acesso ao conjunto de validação durante a competição, não podemos selecionar o modelo que obteve melhor desempenho no conjunto de validação. Dado que o Modelo 1 obteve os melhores resultados no conjunto de teste, optamos por selecioná-lo como sendo o melhor modelo dentre os modelos testados.

5.3. Resultados

A partir da experimentação, escolhemos o Modelo 1 como o melhor dos modelos e treinamos ele novamente, mas dessa vez utilizando tanto o conjunto de treino quanto

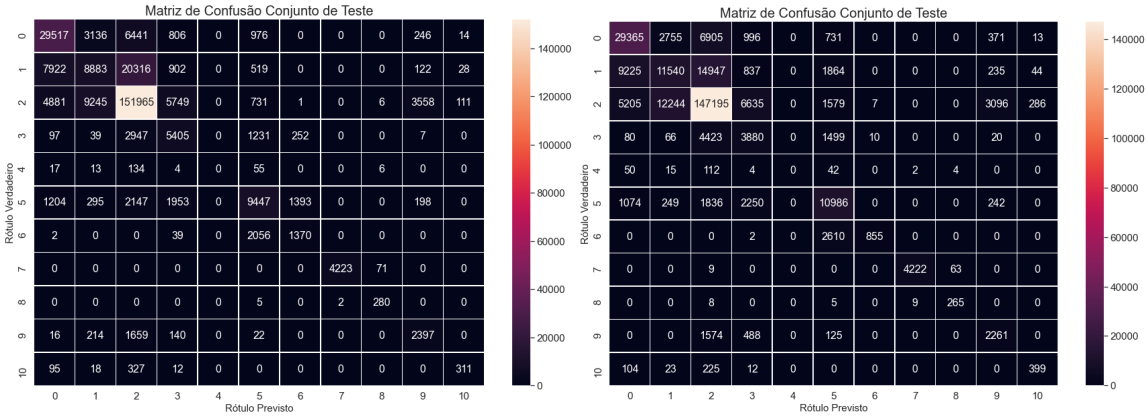


Figure 18. Modelo 1

Figure 19. Modelo 2

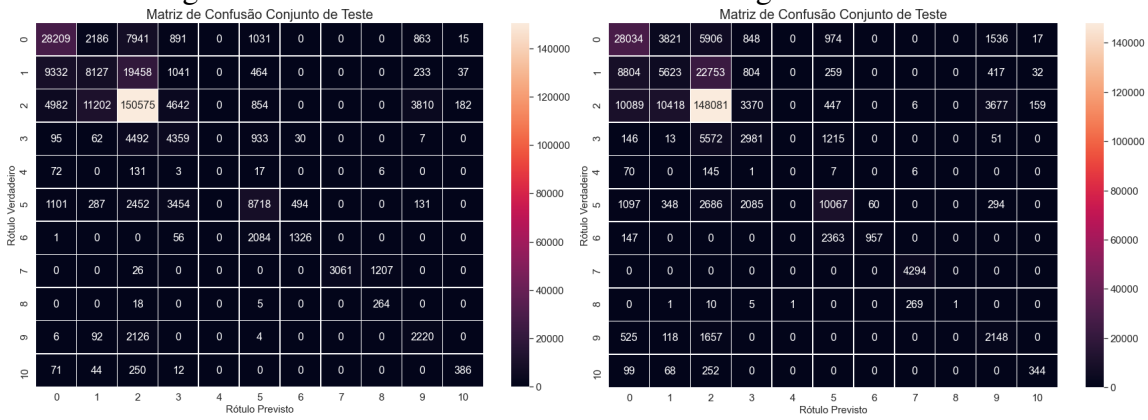


Figure 20. Modelo 3

Figure 21. Modelo 4

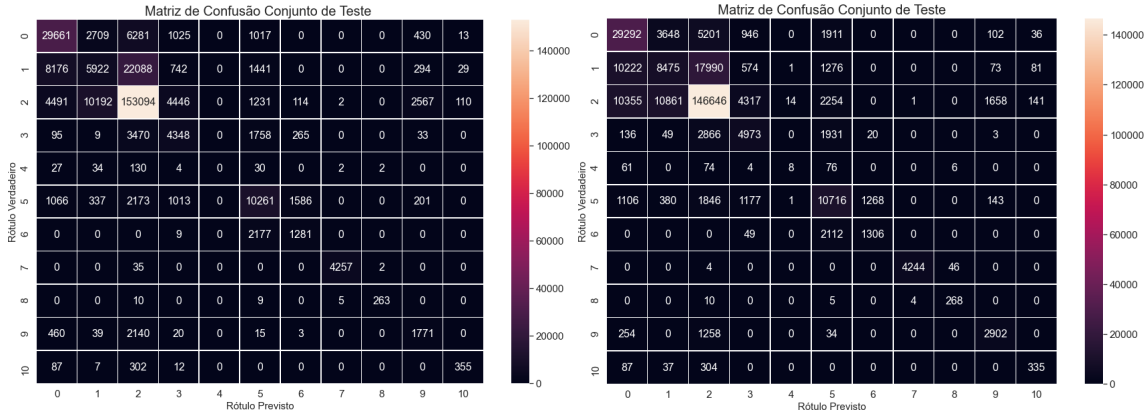


Figure 22. Modelo 5

Figure 23. Modelo 6

Figure 24. Comparação de matrizes de confusão, sobre o conjunto de teste, entre os modelos.

o de teste. Além disso, configuramos seu número de épocas para 40 e removemos o *early stopping*. Em seguida, mostramos seu gráfico de acurácia e perda, sua matriz de confusão, avaliamos sua qualidade usando o *score* no conjunto de validação, e comparamos o seu *score* com os 5 primeiros colocados da competição.

Classe	Acertos	Quantidade	Porcentagem de acerto
Arenito	29517	41136	71.75%
Arenito/Xisto	8883	38692	22.96%
Xisto	151965	176247	86.22%
Marga	5405	9978	54.17%
Dolomita	0	229	0.0%
Calcário	9447	16637	56.78%
Giz	1370	3467	39.52%
Halita	4223	4294	98.35%
Anidrita	280	287	97.56%
Tufo	2397	4448	53.89%
Carvão	311	763	40.76%
Embasamento	0	0	100%

Table 9. Acertos, quantidade de amostras no conjunto de teste e porcentagem de acerto do Modelo 1.

Modelo	Acurácia (Teste)	Acurácia (Validação)	Score (Teste)	Score (Validação)
1	**72,19%**	72,93%	**-0.7006**	-0.6862
2	71,23%	75,69%	-0.7328	-0.5954
3	69,97%	73,16%	-0.7633	-0.6663
4	68,38%	75,33%	-0.8120	-0.6133
5	70,61%	70,09%	-0.7606	-0.7526
6	71,31%	**77,24%**	-0.7262	**-0.5667**

Table 10. Comparação de acurácia e *score*, sobre o conjunto de teste e validação, entre os modelos.

A Figura 25 apresenta a acurácia e perda do melhor modelo durante o treinamento. Tal gráfico é similar aos dos demais modelos vistos anteriormente. Podemos observar que a perda do treinamento cai com o passar das épocas, o que é esperado. Possivelmente, dado que o modelo tivesse mais épocas para ser treinado, a perda de treino provavelmente continuaria caindo. Entretanto, como observamos no Modelo 6 da Figura 17, o modelo possivelmente começaria a sofrer *overffiting* após aproximadamente 40 épocas.

Na Figura 26 mostramos a matriz de confusão do melhor modelo sobre o conjunto de validação. Tal matriz de confusão apresenta características similares às matrizes de confusão apresentadas durante a experimentação. A classe mais prevista é a classe xisto, dado sua maior porcentagem (58.68%) em relação as demais amostras do conjunto de validação. Continuamos cometendo erros para amostras que são similares entre si, como prever a classe xisto enquanto a classe verdadeira é arenito/xisto. Além disso, continuamos acertando ao não prever nenhuma amostra como pertencente a classe embasamento, visto que no conjunto de validação também não temos nenhuma amostra dessa classe.

Na Tabela 12, temos a quantidade de acertos, quantidade de amostras do conjunto de validação e a porcentagem de acerto por classe do melhor modelo. Observamos que o melhor modelo tem dificuldade em identificar dolomitas (20.56%), arenito/xisto (21.91%), marga (24.7%), tufo (39.96%) e anidrita (40.54%). Em contrapartida, o mod-

Time	Score teste	Posição teste	Score validação	Posição validação
Olawale Ibrahim	-0.5118	24	-0.4690	1
GIR	-0.5037	11	-0.4792	2
ICA	-0.4943	6	-0.4954	3
H3G	-0.509	17	-0.5045	4
ISPL	-0.4885	2	-0.5084	5

Table 11. Comparação de acurácia e posição, sobre o conjunto de teste (da competição) e validação, entre os 5 primeiros colocados da competição.

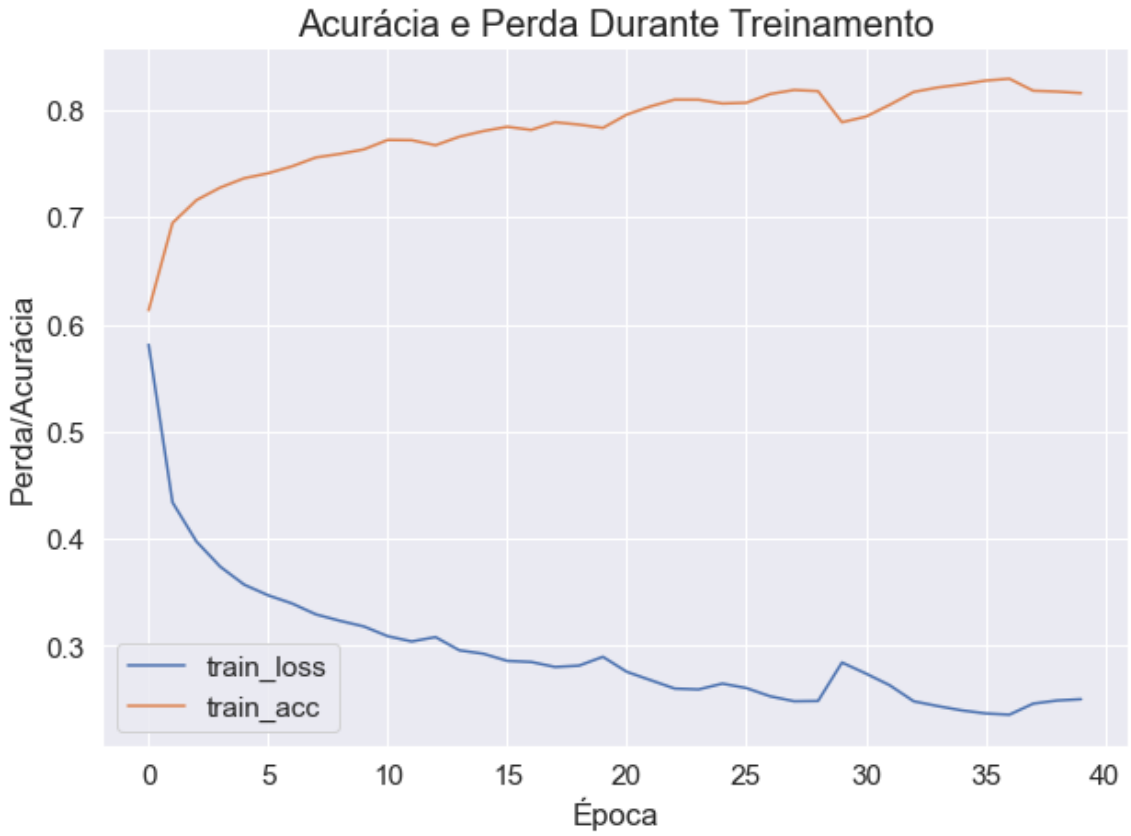


Figure 25. Acurácia e perda do melhor modelo durante treinamento.

elo possui mais de 80% de acerto ao identificar embasamento (100%), xisto (93.09%) e halita (90.43%).

A Tabela 13 mostra os *scores*, modelo e posição, utilizando os dados de validação, dos 5 primeiros colocados na competição, assim como o *score*, modelo e posição do modelo selecionado criado no presente trabalho.

O modelo proposto não apresentou resultados melhores do que os primeiros colocados da competição sobre o conjunto de validação, seu *score* final foi de -0.5450 e sua acurácia final foi de 78,13%. A posição do modelo não pôde ser estabelecida pois o resultado final da competição é apresentado apenas até o décimo terceiro colocado, que possui seu *score* em -0.5441. O resultado atingido pelo modelo proposto não comprova a ineficiência da utilização de LSTM sobre o conjunto de dados descrito no presente tra-

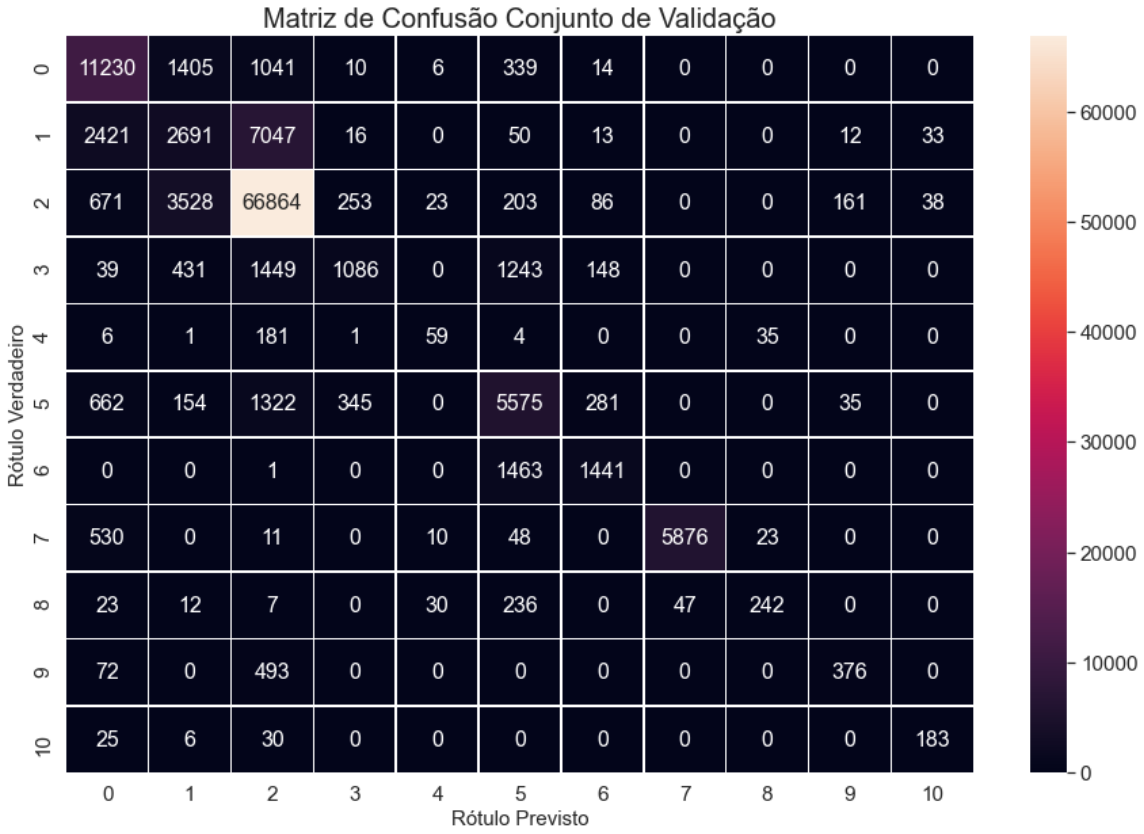


Figure 26. Matriz de convolução do melhor modelo sobre o conjunto de validação.

balho. Mais experimentos poderiam ser realizados considerando outros aspectos tanto da arquitetura do modelo quanto do pré-processamento dos dados para que tivéssemos mais confiança em descartar a utilização de LSTM para a classificação de litofácies.

6. Conclusões e Trabalhos Futuros

Neste trabalho apresentamos os conceitos básicos sobre fácies sedimentares, litofácies e uma breve descrição dos atributos utilizados pela Force 2020 Machine Learning Competition para a classificação de litofácies. Além disso, tivemos uma introdução aos conceitos de IA, ML, DL, RNN e LSTM. Em seguida, mostramos e discutimos os resultados de quatro artigos que aplicaram técnicas para a classificação de litofácies através de modelos de ML. Apresentamos também o conjunto de dados, o pré-processamento realizado sobre os dados, e a métrica de avaliação *score*. Demonstramos alguns dos experimentos realizados utilizando LSTMs, realizamos a comparação entre eles, selecionamos o melhor dentre os modelos apresentados, e comparamos sua qualidade em relação aos modelos dos 5 primeiros colocados na competição.

O objetivo geral do presente trabalho foi atingido. Utilizamos uma rede LSTM, que é capaz de levar em consideração a sequência temporal de formação das diversas camadas de rochas, no problema da classificação automática de litofácies. Apesar do modelo proposto não ter superado os primeiros colocados na competição, seu desempenho não ficou muito distante dos mesmos. Todos os modelos dos 5 primeiros colocados na competição são constituídos por conjuntos de árvores de decisão.

Classe	Acertos	Quantidade	Porcentagem de acerto
Arenito	11230	14045	79.96%
Arenito/Xisto	2691	12283	21.91%
Xisto	66864	71827	93.09%
Marga	1086	4396	24.7%
Dolomita	59	287	20.56%
Calcário	5575	8374	66.58%
Giz	1441	2905	49.6%
Halita	5876	6498	90.43%
Anidrita	242	597	40.54%
Tufo	376	941	39.96%
Carvão	183	244	75.0%
Embasamento	0	0	100%

Table 12. Acertos, quantidade de amostras no conjunto de validação e porcentagem de acerto do melhor modelo.

Time	Modelo	Score	Posição
Olawale Ibrahim	XGBoost	-0.4690	1
GIR	XGBoost	-0.4792	2
ICA	Random Forest	-0.4954	3
H3G	XGBoost	-0.5045	4
ISPL	XGBoost	-0.5084	5
Modelo Selecionado	LSTM	-0.5450	?

Table 13. Comparação de acurácia e *score*, sobre o conjunto de validação, entre os modelos.

Árvores de decisão não necessitam de imputação de dados. O time Olawale Ibrahim, primeiro colocado, fez o preenchimento dos valores ausentes apenas com um único valor constante de valor -999 e ainda conseguiu bons resultados. Por outro lado, o time ICA, terceiro colocado, imputou os valores utilizando a mediana para atributos contínuos e a moda para atributos discretos. Talvez a imputação utilizada no presente trabalho não tenha sido a melhor opção de imputação.

Não conseguimos alterar a função de perda para que o modelo levasse em consideração a matriz de penalidade durante o treinamento da rede neural. Modelos baseados em árvores de decisão não possuem tal parâmetro, portanto não precisaram levar isso em consideração. Possivelmente, com uma função de perda funcionando, o resultado do modelo poderia ter melhorado.

Os modelos propostos pelos times ICA e H3G consideraram o desbalanceamento entre classes durante o treinamento de seus modelos. Visto que temos um grande desbalanceamento entre classes, possivelmente a utilização de pesos para classes poderia ter aumentado a qualidade do modelo proposto.

O modelo também poderia possivelmente ter sido melhorado com a utilização de um conjunto diferente de atributos, atribuição de pesos às camadas específicas e experimentação com outras arquiteturas de rede neural.

Existem algumas possibilidades para melhorar o desempenho da LSTM. Para a realização de trabalhos futuros seria relevante considerar o desbalanceamento entre classes. Para tal poderia-se criar dados sintéticos para as classes sub-representadas, ou reduzir a quantidade de dados das classes mais volumosas, ou até um misto entre essas duas estratégias. Ainda há a possibilidade de considerar o desbalanceamento entre classes diretamente na arquitetura da rede, passando um peso diferente para cada uma das classes.

Outra etapa que poderia ser adicionada seria a utilização do atributo FORCE_2020_LITHOFACIES_CONFIDENCE que indica a medida qualitativa de confiança da interpretação da litologia descrita. Tal atributo poderia ser utilizado durante o treinamento para filtrar dados e/ou atribuir pesos às amostras.

Outras estratégias para imputação de dados também poderiam ser exploradas. Opções variam desde a utilização de conhecimentos geológicos e utilização de outros modelos de ML, até o preenchimento com a média ou a moda dos dados.

A própria competição cita a possibilidade de explorar a redução do peso das amostras imediatamente próximas aos limites das litofácies, pois a escolha exata em cm dos limites das litofácies é frequentemente subjetiva e um pouco imprecisa.

Dentre as arquiteturas experimentadas, o modelo proposto possuía a arquitetura mais simples. Poderia-se ainda alterar a arquitetura do modelo, diminuir ou aumentar a quantidade de neurônios, explorar diferentes combinações de camadas, funções de ativação, número de épocas e demais argumentos.

Por fim, durante o treinamento do modelo proposto, utilizamos como função de perda a *categorical crossentropy*. Utilizar tal função não é o ideal para o treinamento do modelo, visto que há algumas classes que são similares entre si e já temos esse conhecimento de similaridade indicado na matriz de penalidade da competição. Dito isso, criar uma função de perda customizada que leve em consideração a matriz de penalidade possivelmente auxiliaria no treinamento do modelo.

References

2020, F. Force 2020 machine learning competition.

Boggs, S. (2001). *Principles of Sedimentology and Stratigraphy*. Prentice Hall.

Britannica. Sedimentary facies.

Dramsch, J. S. (2020). Chapter one - 70 years of machine learning in geoscience in review. In Moseley, B. and Krischer, L., editors, *Machine Learning in Geosciences*, volume 61 of *Advances in Geophysics*, pages 1 – 55. Elsevier.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Jaikla, C., Devarakota, P., Auchter, N., Sidahmed, M., and Espejo, I. (2019). Faciesnet: Machine learning applications for facies classification in well logs. In *Second Workshop on Machine Learning and the Physical Sciences at the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, pages 10–12.

Mandal, P. P. and Rezaee, R. (2019). Facies classification with different machine learning algorithm–an efficient artificial intelligence technique for improved classification. *ASEG Extended Abstracts*, 2019(1):1–6.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press.

Moore, J. A. (1949). Geographic variation of adaptive characters in rana pipiens schreber. *Evolution*, pages 1–24.

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229.

Yu, Y., Si, X., Hu, C., and Zhang, J. (2019). A review of recurrent neural networks: Lstm cells and network architectures. *Neural Computation*, 31(7):1235–1270.

Yu, Y., Si, X., Hu, C., and Zhang, J. (2019). A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270.