VITA

Release 1.0.0

Fernanda Guidoti Stramantino

Jul 16, 2021

CONTENTS

O projeto VITA foi desenvolvido para treinar um classificador binário incorporando o conceito de stacking em sua solução.

VITA foi implementada como uma aplicação flask em Python e utiliza em sua comunicação API Rest via JSON.

A documentação de todo o código-fonte é encontrada a seguir:

CHAPTER

ONE

CONTEÚDO

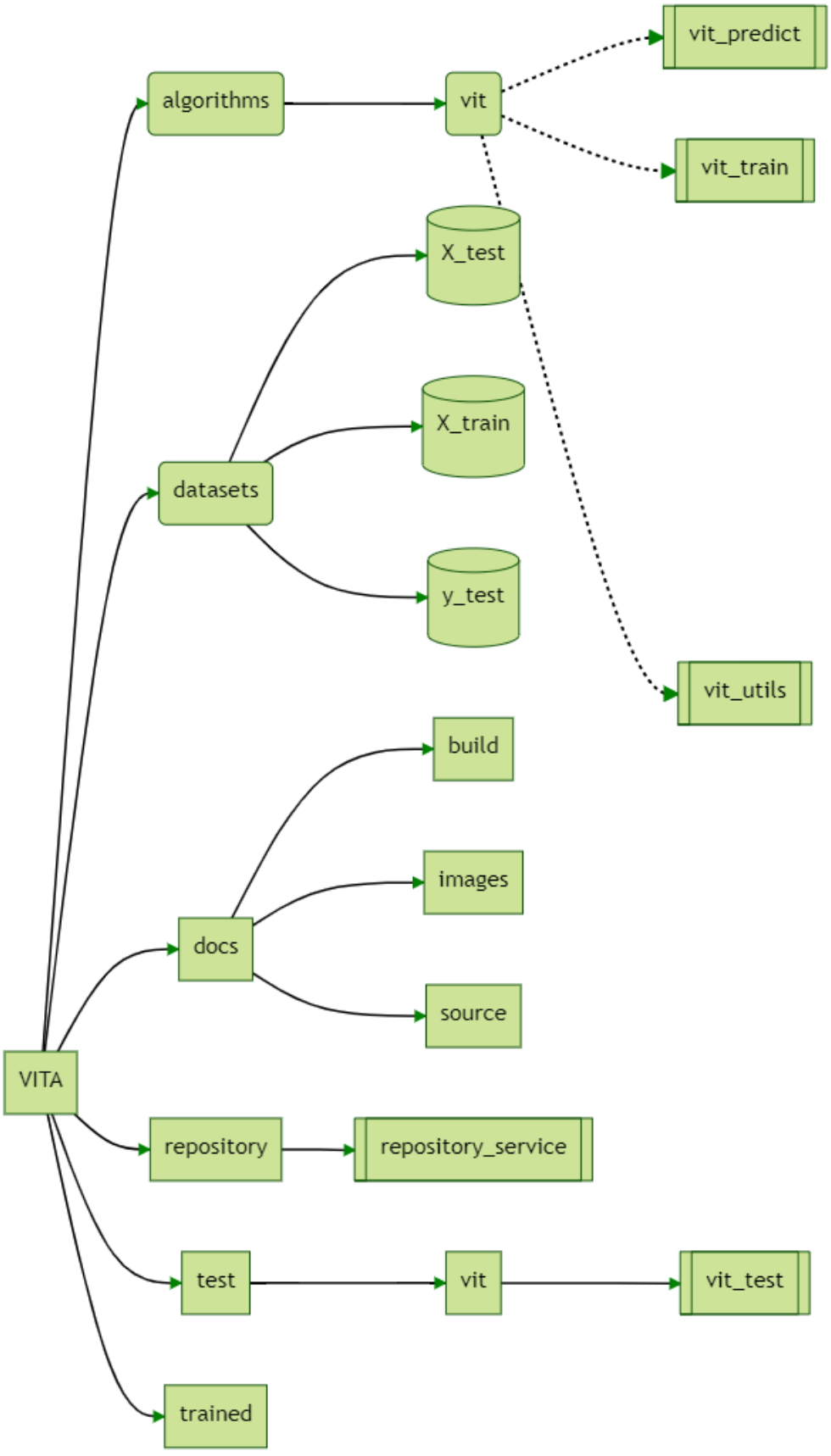
1.1 INTRODUÇÃO

A Plataforma VITA (Virtual Intelligence Training Application) foi desenvolvida para auxiliar cientistas de dados para análise de dados e integração de algoritmos inteligentes.

VITA tem uma arquitetura base que permite intuitivamente o desenvolvimento modular da aplicação. É fácil de implementar e por utilizar container Docker é simples a implantação.

Foi desenvolvido um algoritmo para treinar um classificador binário incorporando o conceito de stacking em sua solução.

A Plataforma VITA possui a seguinte estrutura geral de pastas:


vit_predict
algorithms
vit
vit_train
X_test
X_train
datasets
y_test
vit_utils
build
images
docs
source
VITA
repository
repository_service
test
vit
vit_test
trained

A pasta *algorithms* temos o algoritmo desenvolvido e seus módulos.

A pasta *datasets* contém os arquivos utilizados no treinamento e avaliação dos modelos.

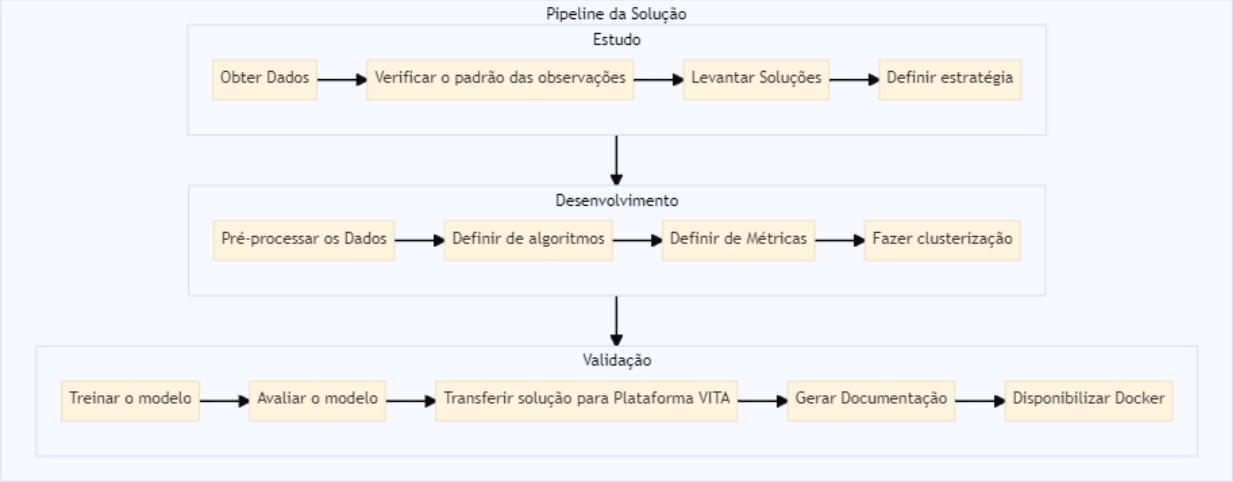
A pasta *docs* contém a documentação de todos os códigos desenvolvidos utilizando a biblioteca Sphinx 4.0.1.

A pasta *repository* contém códigos auxiliares criados para manipulação de dados.

A pasta *test* contém o arquivo de teste unitário de software.

A pasta *trained* contém os modelos treinados.

O Fluxograma Geral da Solução do Desafio:



Por limitação de Tempo, o módulo VIT não foi implementado na Plataforma VITA. Porém seu notebook está disponibilizado na pasta docs como VIT20.ipynb

1.2 Referências

H. Sayadi, N. Patel, S. M. P.D., A. Sasan, S. Rafatirad and H. Homayoun, "Ensemble Learning for Effective Run-Time Hardware-Based Malware Detection: A Comprehensive Analysis and Classification," 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), 2018, pp. 1-6, doi: 10.1109/DAC.2018.8465828.

Haralabopoulos, Giannis, Ioannis Anagnostopoulos, and Derek McAuley. 2020. "Ensemble Deep Learning for Multi-label Binary Classification of User-Generated Content" Algorithms 13, no. 4: 83. https://doi.org/10.3390/a13040083

Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. 2019. Democratizing Data Science through Interactive Curation of ML Pipelines. In Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19). Association for Computing Machinery, New York, NY, USA, 1171–1188. DOI:https://doi.org/10.1145/3299869.3319863

https://docs.docker.com/compose/

1.3 Notebook

import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns %matplotlib inline

from sklearn.preprocessing import MinMaxScaler from sklearn import preprocessing from sklearn.cluster import KMeans

from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier from sklearn.linear_model import LogisticRegression from sklearn.preprocessing import StandardScaler, MinMaxScaler from sklearn.model_selection import KFold from sklearn.metrics import log_loss, accuracy_score from sklearn.pipeline import make_pipeline

 # def main(df): #carrega os datasets

train = pd.read_csv("X_train.csv") test = pd.read_csv("X_test.csv") ytest = pd.read_csv("y_test.csv", names=["target"])

#normaliza entre 0 e 1 x = train.values #returns a numpy array min_max_scaler = preprocessing.MinMaxScaler() x_scaled = min_max_scaler.fit_transform(x) train = pd.DataFrame(x_scaled)

#verifica valores vazios/nulos e exclui # train.isnull() train.dropna(inplace=True)

#faz clusterização kmeans = KMeans(n_clusters=2) kmeans.fit(train)

target = **kmeans.labels_**

df = pd.DataFrame(**kmeans.labels_**) df.columns = ['target']

X = train y = df

kf = KFold(n_splits=2, random_state=0, shuffle=True)

second_level = np.zeros((X.shape[0], 4))

for tr, ts in kf.split(X,y): Xtr, Xval = X.iloc[tr], X.iloc[ts] ytr, yval = y.iloc[tr], y.iloc[ts]

rf = RandomForestClassifier(n_estimators=100, n_jobs=6, random_state=10) rf.fit(Xtr, ytr) prf = rf.predict_proba(Xval)[:,1] **prf_** = (prf > 0.5).astype(int)

print("RF Accuracy: {} - Log Loss: {}".format(accuracy_score(yval, **prf_**), log_loss(yval, prf)))

et = ExtraTreesClassifier(n_estimators=100, n_jobs=6, random_state=10) et.fit(Xtr, ytr) pet = et.predict_proba(Xval)[:,1] **pet_** = (pet > 0.5).astype(int)

print("ET Accuracy: {} - Log Loss: {}".format(accuracy_score(yval, **pet_**), log_loss(yval, pet)))

lr1 = make_pipeline(StandardScaler(), LogisticRegression()) lr1.fit(Xtr, ytr) plr1 = lr1.predict_proba(Xval)[:,1] **plr1_** = (plr1 > 0.5).astype(int)

print("LR StdScaler Accuracy: {} - Log Loss: {}".format(accuracy_score(yval, **plr1_**), log_loss(yval, plr1)))

lr2 = make_pipeline(MinMaxScaler(), LogisticRegression()) lr2.fit(Xtr, ytr) plr2 = lr2.predict_proba(Xval)[:,1] **plr2_** = (plr2 > 0.5).astype(int)

print("LR MinMax Accuracy: {} - Log Loss: {}".format(accuracy_score(yval, **plr2_**), log_loss(yval, plr2)))

second_level[ts, 0] = prf second_level[ts, 1] = pet second_level[ts, 2] = plr1 second_level[ts, 3] = plr2

print()

fatores de diversidade

for tr, ts in kf.split(X,y):

Xtr, Xval = second_level[tr], second_level[ts] ytr, yval = y.iloc[tr], y.iloc[ts]

lr_stack = LogisticRegression(C=1.) lr_stack.fit(Xtr, ytr) plr_stack = lr_stack.predict_proba(Xval)[:,1] **plr_stack_** = (plr_stack > 0.5).astype(int)

print("Stack Accuracy: {} Log loss: {}".format(accuracy_score(yval, plr_stack_), log_loss(yval, plr_stack))) print()

pd.DataFrame(np.corrcoef(second_level.T))

1.4 API - EXEMPLOS DE UTILIZAÇÃO

Este documento tem por objetivo documentar o código da API implementada e como deve ser realizada a sua utilização.

1.4.1 VIT - Algoritmo de Classificador Binário

Esse módulo possui duas possíveis chamadas: *localhost:5000/train_vit* e *localhost:5000/predict_vit*.

1 - Método GET:

A requisição abaixo efetua o treinamento do modelo

Requisição: localhost:5000/train_vit

Exemplo de Saída: (JSON)

```
{
    "1.3562050423284459 64e+00",
    "-1.5091684165331695 77e+00"
}
```

2 - Método POST:

A requisição abaixo efetua a predição do modelo

Requisição: localhost:5000/predict_vit

Exemplo de Entrada: (JSON)

```
{
    "Exemplo"
}
```

Exemplo de Saída: (JSON)

```
{
    "1.3562050423284459 64e+00",
    "-1.5091684165331695 77e+00"
}
```

1.5 ALGORITMOS

Este documento tem por objetivo documentar o código da API implementada e como deve ser realizada a sua utilização.

Porém existe apenas os exemplos, já que não foi possível integrar pelo tempo limitante.

1.5.1 VIT - Algoritmo de Classificação Binária

vit

algorithms.vit.vit_train.**preprocessing**(*df*)
pré processamento dos dados

Parameters **df** – dataframe

Returns dataframe 'limpo'

algorithms.vit.vit_train.**train**(*df*)
utiliza os dados para treinamento e teste do modelo e retorna o modelo treinado :param df: dataframe com os dados :return modelo treinado

algorithms.vit.vit_utils.**remove_null_columns**(*df*)
remove columns null

Parameters **df** – dataframe to process

Returns dataframe without null columns

algorithms.vit.vit_utils.**scaled**(*df*)
normaliza os dados entre 0 e 1

Parameters **df** – dataframe para processar

Returns dados entre 0 e 1

CHAPTER

TWO

ÍNDICES E TABELAS

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX