YULAB tecnologia

bemaker

Workshop Machine Learning utilizando Python

Guia didático

Imaginar, desenvolver, construir, aproximar, integrar, amadurecer e evoluir! Se você teve uma grande ideia, provavelmente passou por este ciclo, é a melhor forma de criar um sucesso. Com essas palavras em mente, construímos a Bemaker! Um makerspace que nasceu da vontade de impulsionar o movimento *Do It Yourself* (Faça você mesmo) para qualquer esfera da sociedade. Nossa missão é disponibilizar um espaço com ferramentas para que você construa seus sonhos! Venha fazer parte deste movimento!

Autores:

- Adelino Pinheiro Silva (adelino@yulab.com.br)
- Fabiano Calado (fabiano@yulab.com.br)
- Lucas Calado Alves Pereira (kalado@yulab.com.br)
- Thiago Amaral (thiago@yulab.com.br)

Contextualização

Este é um guia didático com objetivo de introduzir as técnicas de aprendizado de máquina e aplicá-las na prática. Este material é uma adaptação do guia de Helge Bjorland - An Interactive Data Science Tutorial.

A ferramenta utilizada será o Spyder (que pode ser baixado no link) que roda em ambiente Python. As demais ferramentas são os pacotes da biblioteca SciPy.

O código completo para este workshp pode ser obtido neste link.

1.1 Ambiente Spyder

O Spyder é uma interface de desenvolvimento integrada (IDE) para programação em Phyton que agrega algumas funcionalidades como a interpretação interativa.

A tela principal do Spyder pode ser visualizada na Figura 1.1.

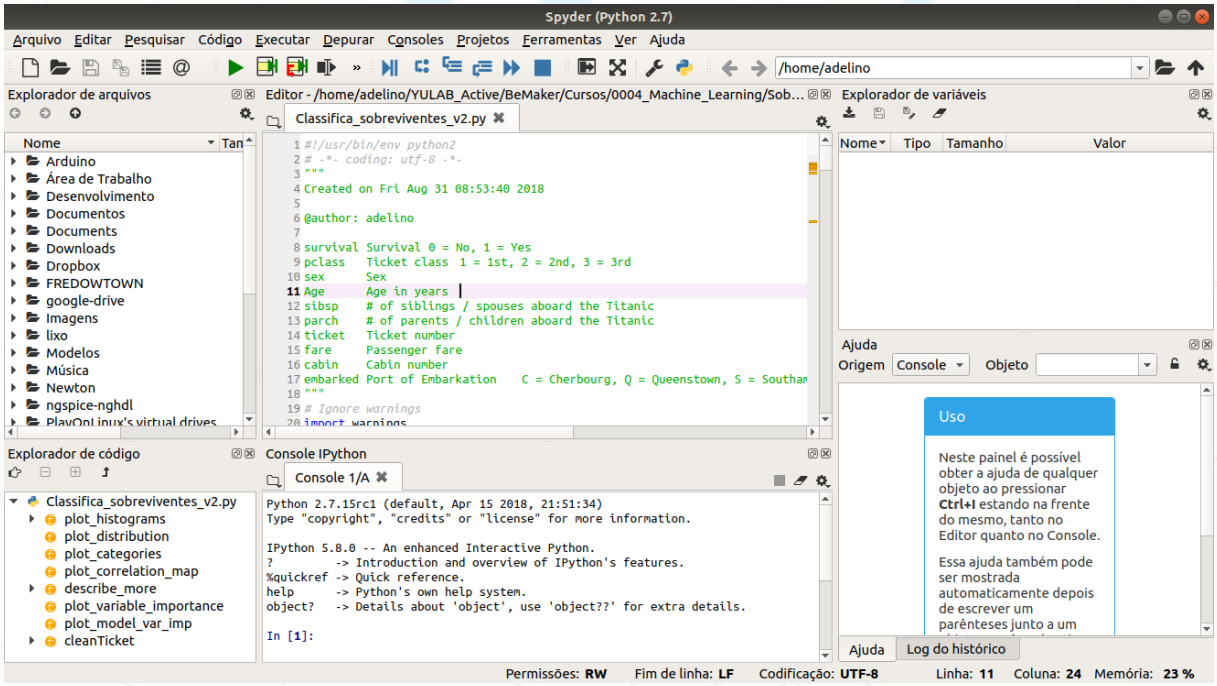

Figura 1.1: Tela de trabalho do Spyder como *layout* do Matlab.

Na tela do Spyder tem-se várias funcionalidades. As mais utilizadas são: a janela para o código, que é interpretado de forma interativa pelo Iphyton; a lista de variáveis e o console (ou *prompt*) de comando.

Uma dica importante, para que os gráficos gerados apareçam em uma nova janela, e não no console siga o seguinte menu e altere a variável `Graphics backend` para `Automatic`.

Tools > preferences > IPython console > Graphics > Graphics backend > Backend: Automatic

Em seguida reinicie o *kernel* do Iphyton com `Ctrl+.`.

Colocando a mão na massa

Nesta etapa vamos colocar a mão na massa e programar no ambiente Spyder. O objetivo principal é utilizar as principais ferramentas de aprendizado de máquina sem se aprofundar na teoria.

2.1 Compreensão e Visualização dos Dados

2.1.1 Importando as bibliotecas

O primeiro passo é preparar o ambiente para manipular os dados. O bloco de código abaixo carregas as principais bibliotecas que serão utilizadas no desenvolvimento.

```
# Comando para ignorar mensagens de warnings
import warnings
warnings.filterwarnings('ignore')

# Bibliotecas para lidar com tabelas e matrizes
import numpy as np
import pandas as pd

# Algoritmos de modelagem
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier

# Algoritmos auxiliares para modelagem de dados
from sklearn.preprocessing import Imputer , Normalizer , scale
from sklearn.cross_validation import train_test_split , StratifiedKFold
from sklearn.feature_selection import RFECV

# Bibliotecas para visualizacao de dados
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import seaborn as sns
```

```
# Fechar todas janelas abertas
plt.close("all")

# Configuracoes de visualizacao
mpl.style.use( 'ggplot' )
sns.set_style( 'white' )
pylab.rcParams[ 'figure.figsize' ] = 8 , 6
```

2.1.2 Definindo funções auxiliares

A linguagem Python também permite fazer o encapsulamento de blocos de códigos. O tutorial original não explica o funcionamento das funções.

Basicamente são funções para visualização de dados.

```


def plot_histograms( df , variables , n_rows , n_cols ):
    fig = plt.figure( figsize = ( 16 , 12 ) )
    for i, var_name in enumerate( variables ):
        ax=fig.add_subplot( n_rows , n_cols , i+1 )
        df[ var_name ].hist( bins=10 , ax=ax )
        ax.set_title( 'Skew: ' + str( round( float( df[ var_name ].skew() ) , ) ) ) #
        ax.set_xticklabels( [] , visible=False )
        ax.set_yticklabels( [] , visible=False )
    fig.tight_layout()  # Improves appearance a bit.
    plt.show()

def plot_distribution( df , var , target , **kwargs ):
    row = kwargs.get( 'row' , None )
    col = kwargs.get( 'col' , None )
    facet = sns.FacetGrid( df , hue=target , aspect=4 , row = row , col = col )
    facet.map( sns.kdeplot , var , shade= True )
    facet.set( xlim=( 0 , df[ var ].max() ) )
    facet.add_legend()

def plot_categories( df , cat , target , **kwargs ):
    row = kwargs.get( 'row' , None )
    col = kwargs.get( 'col' , None )
    facet = sns.FacetGrid( df , row = row , col = col )
    facet.map( sns.barplot , cat , target )
    facet.add_legend()

def plot_correlation_map( df ):
    corr = titanic.corr()
    _ , ax = plt.subplots( figsize =( 12 , 10 ) )
    cmap = sns.diverging_palette( 220 , 10 , as_cmap = True )
    _ = sns.heatmap(
        corr,
        cmap = cmap,
        square=True,
        cbar_kws={ 'shrink' : .9 },
        ax=ax,
        annot = True,
```

```
        annot_kws = { 'fontsize' : 12 }
    )

def describe_more( df ):
    var = [] ; l = [] ; t = []
    for x in df:
        var.append( x )
        l.append( len( pd.value_counts( df[ x ] ) ) )
        t.append( df[ x ].dtypes )
    levels = pd.DataFrame( { 'Variable' : var , 'Levels' : l , 'Datatype' : t } )
    levels.sort_values( by = 'Levels' , inplace = True )
    return levels

def plot_variable_importance( X , y ):
    tree = DecisionTreeClassifier( random_state = 99 )
    tree.fit( X , y )
    plot_model_var_imp( tree , X , y )

def plot_model_var_imp( model , X , y ):
    imp = pd.DataFrame(
        model.feature_importances_  ,
        columns = [ 'Importance' ] ,
        index = X.columns
    )
    imp = imp.sort_values( [ 'Importance' ] , ascending = True )
    imp[ : 10 ].plot( kind = 'barh' )
    print (model.score( X , y ))
```

2.1.3 Carregando dados

Está é uma etapa importante para qualquer tarefa de aprendizado de máquina. Os dados são toda informação disponível para superar os desafios.

Basicamente existem dois conjuntos de dados, os de trinamento e os de teste.

O conjunto de treinamento apresenta dos dados do problema, ou seja, as informações de entrada, e o resultado esperado. Este conjunto permite validar como o aprendizado de máquina esta calibrado.

Com os dados de teste verifica-se o poder de generalização do aprendizado de máquina.

O bloco de código a seguir é utilizado para carregar e fazer uma visualização prévia nos dados.

```
#get titanic & test csv files as a DataFrame
train = pd.read_csv("train.csv")
test    = pd.read_csv("test.csv")

full = train.append( test , ignore_index = True )
titanic = full[ :891 ]

del train , test

```

```
10 print ('Datasets:' , 'full:' , full.shape , 'titanic:' , titanic.shape)
```

Para se familiarizar com os dados é possível acessar algumas linhas da tabela com o comando `titanic.head()`.

Basicamente os dados apresentam as seguintes informações:

1. **survival**: Indica de o passageiro foi um sobrevivente ou não, se o valor for 0 é um não sobrevivente e se o valor for 1 é um sobrevivente.
2. **pclass**: é a classe que a passagem dá acesso. Neste caso tem-se passagens de 1^a, 2^a e 3^a classe.
3. **Name**: Nome do passageiro.
4. **sex**: sexo do passageiro, e uma variável categórica, sendo `male` ou `female`.
5. **Age**: idade em anos.
6. **sibsp**: número de irmãos ou cônjuges abordo do Titanic.
7. **parch**: número de pais ou filhos abordo do Titanic.
8. **ticket**: número da passagem.
9. **faer**: valor da tarifa paga pelo passageiro.
10. **cabin**: Número da cabine.
11. **embarked**: Variável categória que indica o porto em que o passageiro embarcou, sendo `C` para Cherbourg, `Q` para Queenstown e `S` para Southampton.

Uma variável numérica é aquela que apresenta valores inteiros ou números reais, enquanto a variável categórica assume apenas um número limitado, e geralmente fixo, de valores possíveis, como por exemplo o tipo sanguíneo ou o sexo.

2.1.4 Visualização de variáveis numéricas

A sequência de código a seguir permite verificar como as variáveis numéricas se distribuem.

```
1 plot_correlation_map( titanic )
2 plt.savefig('V3_Correlacao.png')
3
4 # Plot distributions of Age of passangers who survived or did not survive
5 plot_distribution( titanic , var = 'Age' , target = 'Survived' , row = 'Sex' )
6
7 # Excersise 1
8 # Plot survival rate by Embarked
9 plot_categories( titanic , cat = 'Embarked' , target = 'Survived' )
```

O desafio desta etapa é verificar as variáveis correlacionadas e entender como elas podem explicar o problema.

Exercício 1: Investigando as taxas de sobrevivência

Nesta etapas vamos o primeiro exercício, gerar gráficos que indique os índices de sobrevivência de acordo com as variáveis do problema.

Este é um exemplo de figura que pode ser gerada.

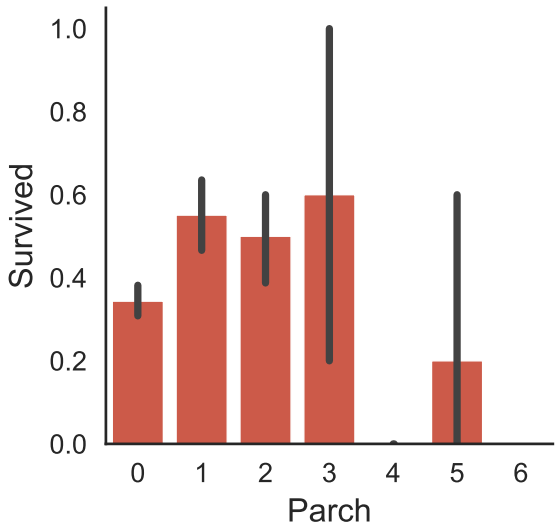

Figura 2.1: Taxa de sobrevivência de acordo com o numero de pais e filhos a bordo.

2.1.5 Visualização de variáveis categóricas

As variáveis `Embarked`, `Pclass` e `Sex` são tratadas nos dados como variáveis categóricas. Como os algoritmos recebem variáveis numéricas, faz-se necessário criar variáveis numéricas relacionadas às variáveis categóricas.

Desta forma, cada variável poderá assumir os valores 0 ou 1 se forem inseridas na em determinada categoria. Por exemplo, para o sexo, cria-se uma variável chamada `male` e o "valor" será codificado como uma variável binária (0 ou 1).

```
1  # Transform Sex into binary values 0 and 1
2  sex = pd.Series( np.where( full.Sex == 'male' , 1 , 0 ) , name = 'Sex' )
3
4  # Create a new variable for every unique value of Embarked
5  embarked = pd.get_dummies( full.Embarked , prefix='Embarked' )
6  embarked.head()
7
8  # Create a new variable for every unique value of Embarked
9  pclass = pd.get_dummies( full.Pclass , prefix='Pclass' )
10 pclass.head()
```

2.2 Preparação dos Dados

2.2.1 Variáveis ausentes

A maioria dos algoritmos de aprendizado de máquina exige que todas as variáveis tenham valores para usá-lo no treinamento do modelo. O método mais simples é preencher os valores ausentes com a média da variável em todas as observações no conjunto de treinamento.

```
# Create dataset
imputed = pd.DataFrame()
# Fill missing values of Age with the average of Age (mean)
imputed['Age'] = full.Age.fillna( full.Age.mean() )
# Fill missing values of Fare with the average of Fare (mean)
imputed['Fare'] = full.Fare.fillna( full.Fare.mean() )
imputed.head()
```

2.3 Engenharia de Características/Recursos - Criação de novas variáveis

2.3.1 Títulos dos nomes dos passageiros

Títulos refletem status social. No caso do Titanic. estes títulos podem prever probabilidade de sobrevivência?

Para verificar esta intuição podemos criar as variáveis referentes aos títulos.

```
title = pd.DataFrame()
# we extract the title from each name
title['Title'] = full['Name'].map( lambda name: name.split(',')[1].split('.')[

# a map of more aggregated titles
Title_Dictionary = {
                    "Capt":       "Officer",
                    "Col":        "Officer",
                    "Major":      "Officer",
                    "Jonkheer":   "Royalty",
                    "Don":        "Royalty",
                    "Sir" :       "Royalty",
                    "Dr":         "Officer",
                    "Rev":        "Officer",
                    "the Countess":"Royalty",
                    "Dona":       "Royalty",
                    "Mme":        "Mrs",
                    "Mlle":       "Miss",
                    "Ms":         "Mrs",
                    "Mr" :        "Mr",
```

```
                    "Mrs" :        "Mrs",
                    "Miss" :       "Miss",
                    "Master" :     "Master",
                    "Lady" :       "Royalty"
                    }

# we map each title
title[ 'Title' ] = title.Title.map( Title_Dictionary )
title = pd.get_dummies( title.Title )
#title = pd.concat( [ title , titles_dummies ] , axis = 1 )
title.head()
```

2.3.2 Informação na categoria das cabines

```
cabin = pd.DataFrame()

# replacing missing cabins with U (for Uknown)
cabin[ 'Cabin' ] = full.Cabin.fillna( 'U' )

# mapping each Cabin value with the cabin letter
cabin[ 'Cabin' ] = cabin[ 'Cabin' ].map( lambda c : c[0] )

# dummy encoding ...
cabin = pd.get_dummies( cabin['Cabin'] , prefix = 'Cabin' )

cabin.head()
```

2.3.3 O que é a classe do *ticket*?

```
# a function that extracts each prefix of the ticket, returns 'XXX' if no prefix (i.e
def cleanTicket( ticket ):
    ticket = ticket.replace( '.' , '' )
    ticket = ticket.replace( '/' , '' )
    ticket = ticket.split()
    ticket = map( lambda t : t.strip() , ticket )
    ticket = list(filter( lambda t : not t.isdigit() , ticket ))
    if len( ticket ) > 0:
        return ticket[0]
    else:
        return 'XXX'

ticket = pd.DataFrame()

# Extracting dummy variables from tickets:
ticket[ 'Ticket' ] = full[ 'Ticket' ].map( cleanTicket )
ticket = pd.get_dummies( ticket[ 'Ticket' ] , prefix = 'Ticket' )

ticket.shape
ticket.head()
```

2.3.4 Tamanho e tipo de família

```
family = pd.DataFrame()
```

```
2
3  # introducing a new feature : the size of families (including the passenger)
4  family[ 'FamilySize' ] = full[ 'Parch' ] + full[ 'SibSp' ] + 1
5
6  # introducing other features based on the family size
7  family[ 'Family_Single' ] = family[ 'FamilySize' ].map( lambda s : 1 if s == 1 else 0
8  family[ 'Family_Small' ]  = family[ 'FamilySize' ].map( lambda s : 1 if 2 <= s <= 4 el
9  family[ 'Family_Large' ]  = family[ 'FamilySize' ].map( lambda s : 1 if 5 <= s else 0
10
11 family.head()
```

2.4 Criando da base de dados do modelo final

Nesta etapa criam-se os conjuntos de trinamento e de teste para ter um conjunto de avaliação do modelo.

O conjunto de dados também é dividido por colunas em uma matriz (X) contendo os dados de entrada e um vetor (y) contendo o alvo (ou rótulos, sobreviveu ou não).

Selecione quais variáveis deseja incluir no conjunto de dados na lista abaixo:

- imputed;
- embarked;
- pclass;
- sex;
- family;
- cabin;
- ticket;
- title?

Inclua as variáveis que você gostaria de usar na função abaixo separadas por vírgula e, em seguida, execute o bloco de código abaixo.

```
1  # Select which features/variables to include in the dataset from the list below:
2  # imputed , embarked , pclass , sex , family , cabin , ticket
3
4  full_X = pd.concat( [ imputed , embarked , cabin , sex ] , axis=1 )
5  full_X.head()
6
7  # Create all datasets that are necessary to train, validate and test models
8  train_valid_X = full_X[ 0:891 ]
9  train_valid_y = titanic.Survived
10 test_X = full_X[ 891: ]
11 train_X , valid_X , train_y , valid_y = train_test_split( train_valid_X , train_valid_
12 print (full_X.shape , train_X.shape , valid_X.shape , train_y.shape , valid_y.shape ,
13 plot_variable_importance(train_X, train_y)
```

2.5 Modelagem

A modelagem do aprendizado de máquina é basicamente a tipologia ou a técnica de classificação das variáveis de entrada para obter a variável de saída. Neste caso, se o passageiro sobreviveu ou não.

2.5.1 Seleção do Modelo, Treinamento e Avaliação

Utilize a o modelo que julgar mais adequado. Uma observação, no quadro de código a seguir retire o comentário do modelo que deseja entre as linas 1 e 7. Caso utilize o modelo da linha 1 (*Randon Florest*) retire o comentário da linha 15.

```
1  #model = RandomForestClassifier(n_estimators=100)
2  #model = SVC()
3  #model = GradientBoostingClassifier()
4  #model = KNeighborsClassifier(n_neighbors = 3)
5  #model = GaussianNB()
6  model = MLPClassifier(learning_rate = 'adaptive', hidden_layer_sizes = 200, max_iter =
7  #model = LogisticRegression()
8
9  model.fit( train_X , train_y )
10 print (model.score( train_X , train_y ) , model.score( valid_X , valid_y ))
11 #plot_model_var_imp(model, train_X, train_y)
12
13 #rfecv = RFECV(estimator = model, step = 1, cv = StratifiedKFold(2), scoring = 'accura
14 rfecv = RFECV( estimator = model , step = 1 , cv = StratifiedKFold( train_y , 2 ) , sc
15 #rfecv.fit( train_X , train_y )
16
17 test_Y = model.predict( test_X )
18 test_Y = np.asarray(test_Y, dtype=int)
19 passenger_id = full[891:].PassengerId
20 test = pd.DataFrame( { 'PassengerId': passenger_id , 'Survived': test_Y } )
21 test.shape
22 test.head()
23 test.to_csv( 'titanic_pred.csv' , index = False )
```

Esta é a etapa final da do processo. Vamos avaliar os resultados de acordo com a métrica do problema.

2.5.2 Métrica

A métrica para avaliar a pontuação é porcentagem de passageiros previstos corretamente. Este parâmetro é conhecido como precisão ou acurácia.

A precisão mede o quão bem a classificação binária identifica ou exclui corretamente uma condição. Ou seja, a precisão é a proporção de resultados verdadeiros (verdadeiros positivos e negativos) entre o número total de casos examinados.

2.5.3 Teste e submissão dos resultados

Para submeter o resultado no site do kaggle acesse o link e faça login.

Na tela do desafio do Titanic clique em `Submit Predictions`

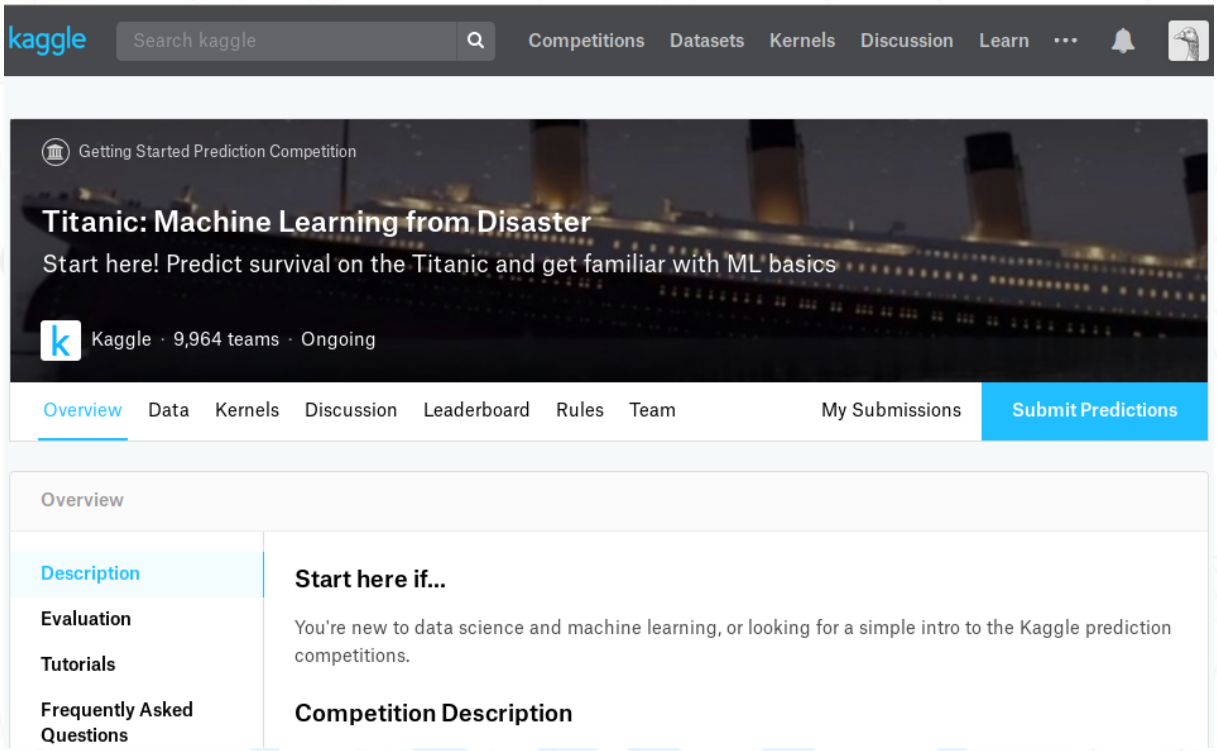

Figura 2.2: Tela de submissão dos resultados no Kaggle.

Considerações Finais

Conheça a BEMAKER nas redes sociais

www.facebook.com/bemakerbrasil

www.instagram.com/bemakerbr

www.linkedin.com/company/bemakerbr/

Bemaker - Eventos
Avenida Ivaí, 228, Dom Bosco
Belo Horizonte, MG

Yulab Tecnologia / Bemaker
Avenida Santa Matilde, 535, Dom Cabral
Belo Horizonte, MG

Este material está disponível no GitLab.