UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS APUCARANA

GUSTAVO MATOS LÁZARO

RELATÓRIO TITANIC E UMA SOLUÇÃO DE ML

07/10/2020

1. Introdução

Este relatório visa explicar como foi realizado um modelo de previsão de sobreviventes à catástrofe do Titanic, assim como também tem o objetivo de mostrar o passo a passo para ter um modelo final.

O naufrágio do Titanic é um acontecimento conhecidos por todos, algo que vai ficar para sempre na história, por ser um navio considerado “inafundável”, que veio a afundar ao colidir com um iceberg durante a sua primeira viajem, matando assim 1502 pessoas das 2224 a bordo do navio. O desafio proposto nesse relatório visa, com base em estatísticas e probabilidade, construir um modelo preditivo utilizando Python que possa informar se uma pessoa sobreviveria ou não ao desastre de 1912.

2. Desenvolvimento

Bem, antes de mais nada precisa-se obter informações para que se possa trabalhar em cima delas. Intuitivamente, o desafio já te entrega arquivos .csv contendo tudo que se precisa para começar.

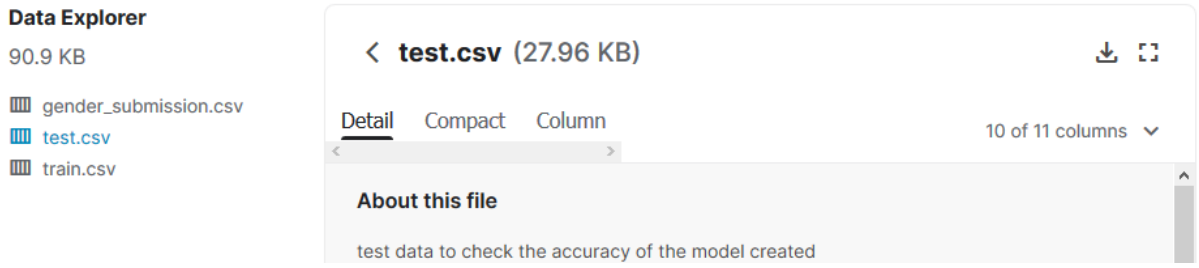

Figura 1 – Dados a serem utilizados entregues pelo kaggle.com

Ao carregar os arquivos no código, pode-se ver que cada arquivo desses contém uma série de informações, de pessoas que estavam no Titanic, separadas em colunas de atributos. Sendo eles:

- PassengerId: ID do passageiro
- Survived: 0 = Morreu; 1 = Sobreviveu
- Pclass: Classe de embarque escolhida
- Name: Nome
- Sex: Sexo
- Age: Idade
- SipSp: Quantidade de irmãos e cônjuges a bordo
- Parch: Quantidade de pais e filhos a bordo
- Ticket: Código da passagem
- Fare: Preço pago para embarcar
- Cabin: Cabine de aposentos
- Embarked: Porto de embarque (C = Cherboung; Q = Queenstown; S = Southampton)

```
titanic_treino.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

Imagem 2 - Retorno da instrução *.head(10)* que mostra as 10 primeiras linhas do arquivo *titanic_teste.*

Depois de se aprofundar um pouco mais no arquivo de teste para ter uma ideia dos valores possíveis de um atributo (célula 353 a 359), foi interessante ver quantos entre as 891 colunas de informações sobreviveram ou não:

```
In [360]: #Quantos sobreviveram ao total

          titanic_treino['Survived'].value_counts()

Out[360]: 0    549
          1    342
          Name: Survived, dtype: int64
```

Imagem 3 – Retorno da função *.value_counts()* que mostra quantos valores se repetem no atributo "Survived" do arquivo titanic_teste, sendo o resultado 0 para mortos e 1 para sobreviventes.

Sabendo dessa informação, foi interessante relacionar a coluna "Survived" com as outras para ver uma possível dependência ou impacto em seu resultado:

```
In [362]: #Grafico de sobreviventes em relação as outras colunas

cols = ['Survived', 'Sex', 'Pclass', 'SibSp', 'Parch', 'Embarked']

n_rows = 2
n_cols = 3

# A grade subplot e o tamanho da figura de cada gráfico
# Isso retorna uma imagem (fig) e os eixos Object (axs)
fig, axs = plt.subplots(n_rows, n_cols, figsize=(n_cols*4,n_rows*4))

for r in range(0,n_rows):
    for c in range(0,n_cols):

        i = r*n_cols+ c #Percorre colunas
        ax = axs[r][c] #Posição de cada subplot
        sns.countplot(titanic_treino[cols[i]], hue=titanic_treino["Survived"], ax=ax)
        ax.set_title(cols[i])
        ax.legend(title="Sobreviventes", loc='upper right')

plt.tight_layout()   #tight_layout
```

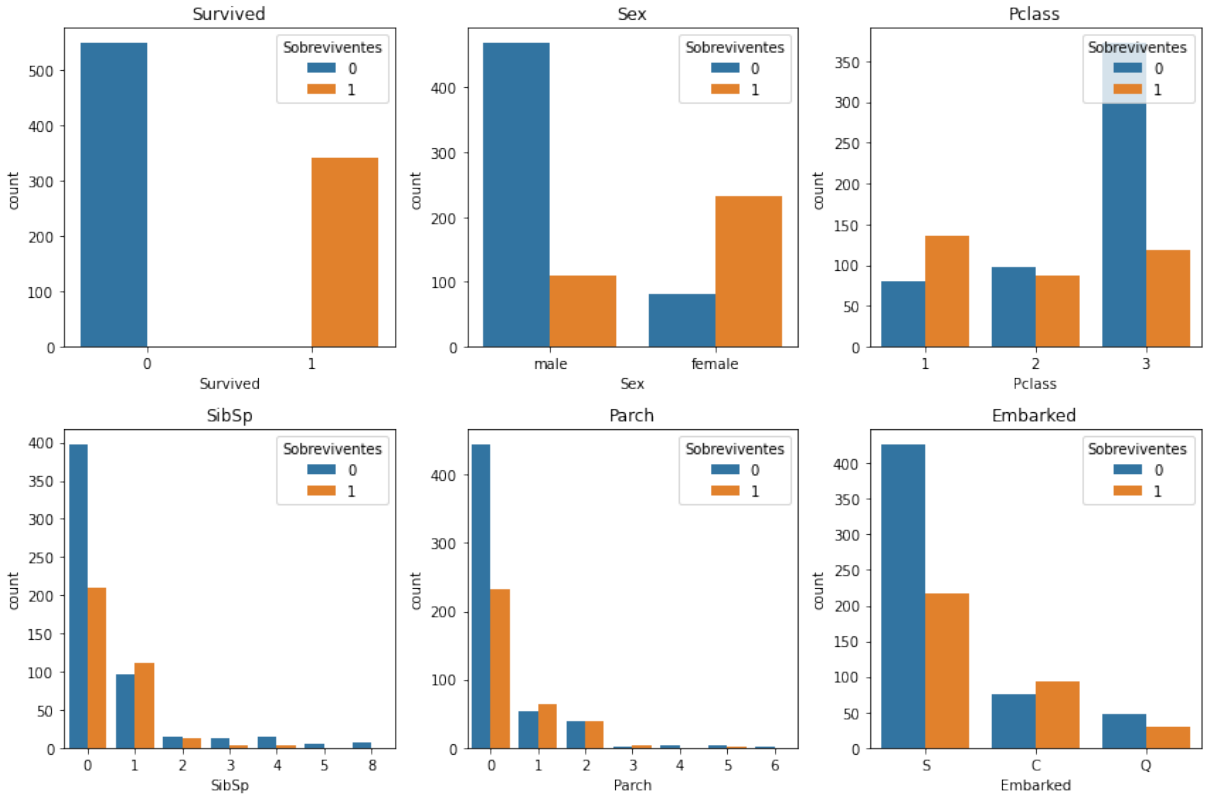

Imagem 4 – Retorno da célula 362, onde relaciona a coluna “Survived” com outras do arquivo tinanic_teste.

Essa relação mostrada agora, vai ter uma grande importância mais para frente para determinar se uma pessoa vai viver ou não.

3. Pré-processamento dos dados

Com os dados analisados, vem uma etapa muito importante para fazer o modelo, que é preparar os dados. Essa parte visa remover inconsistências de informações, como posições vazias na tabela, ou informações únicas que não agregaram nada para a decisão final, como por exemplo o Id do passageiro ou seu nome pessoal.

Primeiramente, foi visto quantas linhas vazias tem cada coluna:

```
In [366]: #Quantas posições vazias tem cada coluna da tabela

          titanic_treino.isnull().sum()

Out[366]: PassengerId      0
          Survived         0
          Pclass           0
          Name             0
          Sex              0
          Age            177
          SibSp            0
          Parch            0
          Ticket           0
          Fare             0
          Cabin          687
          Embarked         2
          dtype: int64
```

Imagem 5 – Retorno da função *Isnull().sum()* que retorna a quantidade de posições vazias de cada atributo

Tendo essa infromação em mão, o proximo passo foi eliminar as colunas que seriam desnecessárias para a análise final, sendo elas: 'Cabin', 'Name', 'Ticket' e ‘PassengerId'.

A coluna ‘Cabin foi eliminada também por faltar a maioria dos dados, na qual foi analisado que seria ineficiente preencher esses espaços, já que teria uma grande chance de serem errôneos.

```
In [367]: #Removendo a coluna muito vazias ou indiferentes

          titanic_treino = titanic_treino.drop(['Cabin', 'Name', 'Ticket', 'PassengerId'], axis=1)
```

Imagem 6 – Código da remoção das colunas selecionadas

Depois de remover as colunas indiferentes, teve-se um empasse em relação as posições vazias na coluna “Age” e “Embarked”, a qual foi decidido eliminar essas posições vazias, mas que poderia também ser preenchidas, por exemplo com uma média, já que eram a minoria nesse caso.

```
In [369]: #Removendo as linhas que tem valores faltando das colunas
          #Poderia usar o fillna tbm, mas prefiri mandar embora
          titanic_treino = titanic_treino.dropna(subset =['Embarked', 'Age'])
```

Imagem 7 – Eliminação das linhas vazias das colunas selecionadas

Depois de eliminar inconsistências no arquivo, foi notado que duas colunas (“Sex” e “Embarked”) tem valores não numéricos como opção para linha. Para melhorar isso, foi usado uma função de codificação da biblioteca Sklearn, para altear esses objetos para valores numéricos:

```
In [373]: #Valores possiveis nas colunas não numericas
          print(titanic_treino['Sex'].unique())
          print(titanic_treino['Embarked'].unique())

          ['male' 'female']
          ['S' 'C' 'Q']

In [374]: #Mudando ass variaveis não numericas de Sex e Embarked para valores numericos

          #Encoding categorical data values (Transforming object data types to integers)
          from sklearn.preprocessing import LabelEncoder
          labelencoder = LabelEncoder()

          #Encode sex column
          titanic_treino.iloc[:,2]= labelencoder.fit_transform(titanic_treino.iloc[:,2].values)
          #print(labelencoder.fit_transform(titanic.iloc[:,2].values))

          #Encode embarked
          titanic_treino.iloc[:,7]= labelencoder.fit_transform(titanic_treino.iloc[:,7].values)
          #print(labelencoder.fit_transform(titanic.iloc[:,7].values))

In [375]: #Novos valores possiveis das colunas
          print(titanic_treino['Sex'].unique())
          print(titanic_treino['Embarked'].unique())

          [1 0]
          [2 0 1]
```

Imagem 8 – Codificação das variáveis “male” e “female” para 1 e 0, e “S”, “C” e “Q” para 2, 0 e 1 respectivamente.

Bom, com as informações já reprocessadas, basta agora escolher qual modelo de decisão vai ser usado nesse programa. Para essa escolha, foi usado a análise feita pelo “randerson112358”, em um artigo na internet (Titanic Survival Prediction Using Machine Learning), no qual o autor faz uma comparação bem detalhada de vários modelos, e mostrando o resultado que eles obtiveram:

```
[0]Logistic Regression Training Accuracy: 0.7978910369068541
[1]K Nearest Neighbor Training Accuracy: 0.8664323374340949
[2]Support Vector Machine (Linear Classifier) Training Accuracy: 0.7768014059753954
[3]Support Vector Machine (RBF Classifier) Training Accuracy: 0.8506151142355008
[4]Gaussian Naive Bayes Training Accuracy: 0.8031634446397188
[5]Decision Tree Classifier Training Accuracy: 0.9929701230228472
[6]Random Forest Classifier Training Accuracy: 0.9753954305799648
```

Imagem 9 – Resultado obtido pelo “randerson112358” em seus testes

Agora, como saber se esses modelos previram bem? Para ter essa informação, foi usado uma matriz de confusão neles, a qual tem o intuito de mostrar as frequências de classificação para cada classe do modelo.

		Prediction	
		0	1
Actual	0	TN	FP
	1	FN	TP

Imagem 10 - Matriz de confusão

- Verdadeiro positivo (true positive — TP): ocorre quando no conjunto real, a classe que estamos buscando foi prevista corretamente.

- Falso positivo (false positive — FP): ocorre quando no conjunto real, a classe que estamos buscando prever foi prevista incorretamente.

- Falso verdadeiro (true negative — TN): ocorre quando no conjunto real, a classe que não estamos buscando prever foi prevista corretamente.

- Falso negativo (false negative — FN): ocorre quando no conjunto real, a classe que não estamos buscando prever foi prevista incorretamente.

Tendo entendido como funciona a matriz de confusão, usamos seus resultados para calcular a acurácia de cada método:

$$accurary = \frac{TP + TN}{TP + FP + TN + FN} = \frac{predições\ corretas}{todas\ as\ predições}$$

```
[[73  9]
 [18 43]]
Model[0] Testing Accuracy = "0.8111888111888111 !"

[[71 11]
 [20 41]]
Model[1] Testing Accuracy = "0.7832167832167832 !"

[[70 12]
 [18 43]]
Model[2] Testing Accuracy = "0.7902097902097902 !"

[[75  7]
 [22 39]]
Model[3] Testing Accuracy = "0.7972027972027972 !"

[[69 13]
 [23 38]]
Model[4] Testing Accuracy = "0.7482517482517482 !"

[[60 22]
 [10 51]]
Model[5] Testing Accuracy = "0.7762237762237763 !"

[[67 15]
 [13 48]]
Model[6] Testing Accuracy = "0.8041958041958042 !"
```

Imagem 11 – Matriz de confusão e acurácia de cada modelo. Dados obtidos pelo "randerson112358" em seus testes.

Com esses dados do artigo "Titanic Survival Prediction Using Machine Learning", foi escolhido o modelo de Random florest para ser usado no programa, pois ele obteve um valor bem alto de precisão quando foi testado nos dados de treino e o segundo melhor valor de acurácia quando testado pela matriz de confusão.

4. Teste com o modelo Random Florest

Com o terreno preparado, agora só resta aplicar e testar o modelo Random Florest no código e na base de dados. Para isso, foi separado a coluna "Survived" das demais, já que sua variavel é dependentes das outras, e já são os resultados finais que desejamos buscar. Os valores das outras colunas, variaveis independentes, foram armazenadas em um grupo.

```
In [377]: #Agora dividir os dados independentes e dependentes em X e Y

          X = titanic_treino.iloc[:, 1:8].values
          Y = titanic_treino.iloc[:, 0].values
```

Imagem 12 – Separação dos valores em dois grupos

Após a separação, foi dividido o banco de dados em 80% para treino do modelo e 20% para teste do mesmo:

```
In [378]: #80% do banco de dados para treino e 20% para teste

          from sklearn.model_selection import train_test_split

          X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
```

Imagem 13 – Separação dos dados para teste e treino

Feito isso, o proximo passo e treinar o modelo:

```
In [379]: #Usando o modelo de random florest agora para analisar os dados
          #https://gist.github.com/BetterProgramming/efec68d9ee0aeb988aa73abf74560ff7#file-models-py

          def modelo(X_train,Y_train):
              from sklearn.ensemble import RandomForestClassifier
              forest = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
              forest.fit(X_train, Y_train)
              print('[0]Random Forest Classifier Training Accuracy:', forest.score(X_train, Y_train))
              return forest

In [380]: #Recebendo e treiando o modelo selecionado
          model = modelo(X_train,Y_train)

          [0]Random Forest Classifier Training Accuracy: 0.9753954305799648
```

Imagem 14 – Treinamento do modelo e acurácia em relação as variáveis de treino.

Agora, resta testar o modelo nos 20% que foi separado para esse fim, e calcular a acurácia pela matriz de confusão:

```
In [381]: #Usando agora a matrix de confusao para testar a acuracia

          from sklearn.metrics import confusion_matrix
          cm = confusion_matrix(Y_test, model.predict(X_test))
          #extracting TN, FP, FN, TP
          TN, FP, FN, TP = confusion_matrix(Y_test, model.predict(X_test)).ravel()
          print(cm)
          print('Acuracia do modelo testado = "{} !"'.format((TP + TN) / (TP + TN + FN + FP)))

          [[67 15]
           [13 48]]
          Acuracia do modelo testado = "0.8041958041958042 !"
```

Imagem 15 – Matriz de confusão e acurácia do modelo em relação as variáveis de teste.

Foi visto que o modelo se saiu bem, com 80% de precisão ao ser testado. Com esses reultados, agora é interresante saber qual dos atributos foram mais importantes para o modelo tomar da decisão se açguem morre ou vive:

Out[382]:

	Importancia
Atributo	
Age	0.300
Fare	0.296
Sex	0.183
Pclass	0.098
SibSp	0.050
Parch	0.044
Embarked	0.030

Imagem 16 – Resultado da célula 382 na qual foi calculado a importância de cada atributo na tomada de decisão

Com isso, todos os resultados necessários para validar esse modelo foram obrtidos, agora resta partir para o .csv de teste e ver como ele se sai em uma situação real.

5. Random Florest na prática

Como foi dito na introdução desse relatório, o site do desafio do titanic nos deu dois arquivos para serem usados, um de treino e um de teste. O de treino foi discedado para termos um modelo funcional para este caso, agora nos resta ver se ele realmente funciona, agora utilizando o .csv de teste.

Assim feito anteriormente, foi preciso pré-processar o arquivo de teste também, eliminando todos os atributos disnecessarios e as linhas vazias, assim como codificando as variaveis objeto para valores numericos (célula 386 à 398).

Depois disso, só restou prever o numero de mortos e vivos no modelo de teste:

```
In [401]: Z = titanic_teste.iloc[:, 0:7].values

In [404]: #Modelo prevendo no banco de dados de treino | 0 = Morreu / 1 = Viveu

          pred = model.predict(Z)
          print(pred)

[0 0 0 1 1 0 0 0 0 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1 0 0 0 0 0 1
 1 0 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1 0 0 1 1 0 0 1 1 0 1 1 1 1 0 0 1 0 1
 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0 0 1 1 1 0 1 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 1 0 0 0 1 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 1 1 0 1 1 1 0 1 1 0
 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 0 1 0
 1 0 1 1 1 1 1 1 0 1 1 0 1 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 1
 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0
 1 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 1 1 0 0 0 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 0 1
 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 0]
```

Imagem 17 – Resultado obtido pela previsão do modelo Random florest no arquivo de teste.

6. Conclusão

O objetivo de construir um modelo preditivo utilizando Python que possa informar se uma pessoa sobreviveria ou não ao desastre de 1912 foi alcançado com sucesso, tento um algoritmo de aprendizado funcional e preciso para isso.

A maior dificuldade passada no processo de construção desse código foi o entendimento de cada função usada, assim como isso tudo iria entregar um resultado palpável, mas que foi superada no final do processo com a ajuda dos artigos e sites citados acima. Sendo assim, o desafio foi concluído com sucesso.

Repositório do código: <https://github.com/Gubiar/Titanic>

7. Referências bibliográficas

Titanic: Machine Learning from Disaster. Kaggle. 2020. Disponível em: <https://www.kaggle.com/c/titanic/data?select=test.csv> Acesso em: 08/10/2020.

randerson112358. Titanic Survival Prediction Using Machine Learning. Medium. 2020. Disponível em: < https://medium.com/better-programming/titanic-survival-prediction-using-machine-learning-4c5ff1e3fa16> Acesso em: 08/10/2020.