Engenheiro da Computação

Caio Henrique Oliveira Cunha

Desafio Técnico Portal Telemedicina

Montes Claros - MG
2021

SUMÁRIO

1 Introdução

Esse relatório tem por objetivo demonstrar o desenvolvimento do desafio técnico proposto pela empresa Portal Telemedicina. Seguindo o enunciado do problema proposto pela empresa, foi desenvolvido um modelo capaz de predizer o sexo de pacientes com base em algumas características do mesmo.

Os capítulos 2,3,4,5,6,7 explicam os passos realizados para a construção do arquivo python 'treinador_modelo.ipynb', que por sua vez tem como funcionalidade analisar, tratar e treinar um modelo de classificação. A seção 8 explica os passos realizados para a construção do arquivo 'sex_predictor.py', que realiza a leitura de um csv pelo terminal, trata os dados e realiza predições com o modelo treinado.

2 Tratamento dos dados

O csv disponibilizado foi lido e transformado em um *DataFrame*, utilizando a biblioteca do *Pandas*. Com os dados dispostos em um *DataFrame*, uma função nomeada como 'tratar_separar_dados()', foi desenvolvida com o objetivo de tratá-los, tais tratamentos são: Tratamentos de NAN (Not a Number) utilizando o método *fillna* da biblioteca do *Pandas*; Valores da coluna 'sex', M e F, foram transformados em 1 e 0, respectivamente; Separação dos atributos (*features*) e label. Após a analíse, explicada na próxima seção, os tratamentos da análise, ou seja, remoção de *features* foi realizada na função mencionada anteriormente.

3 Normalização dos dados

Utilizando o método *StandardScaler()* da biblioteca *scikit-learning*, os dados foram normalizados, entretanto antes de realizar a normalização, a função 'tratar_separar_dados()' tem que ser executada. Assim, foi desenvolvido uma função, nomeada como 'normalizar_dados()', responsável em normalizar os dados das *features* de treinamento e teste.

4 Separação dos dados de treinamento e testes

Foi desenvolvido uma função, nomeada de 'separar_dados_treinamento_teste()' no intuito de separar os dados de treinamento e testes. Um argumento é passado para a função, definindo a porcentagem de amostras de treinamento.

5 Análise dos dados

A partir dos dados disponíveis no csv 'test_data_CANDIDADE.csv' foi possível fazer algumas análises com objetivo de encontrar os melhores atributos (*features*) para o treinamento do modelo. Visualizando os dados é possível notar que a coluna 'cp' não representa uma *feature* significante, pois só possui um único valor, logo foi desconsiderada. A coluna 'slope' possui mais de 50% de valores NAN (Not a Number), indicativo que pode não ser significante ao modelo, logo essa *feature* foi desconsiderada.

A correlação, ou seja interdependência entre as variáveis, foi utilizada para melhorar o modelo, portanto um gráfico da correlação entre as *features* e *label* foi gerado, conforme

a Figura 01. Para alguns métodos de *machine learning* esse processo de encontrar as *features* menos correlacionadas com o *label* é da natureza do algoritmo, porém, mesmo assim, pode-se conseguir melhores resultados descartando aquelas *features* independentes. Com base no gráfico, os atributos 'cp', 'slope', 'trestbps','fbs','restecg','thalach','sk' podem ser removidos do treinamento do modelo, pois a sua corelação em relação ao *label* estão bem distantes de 0.

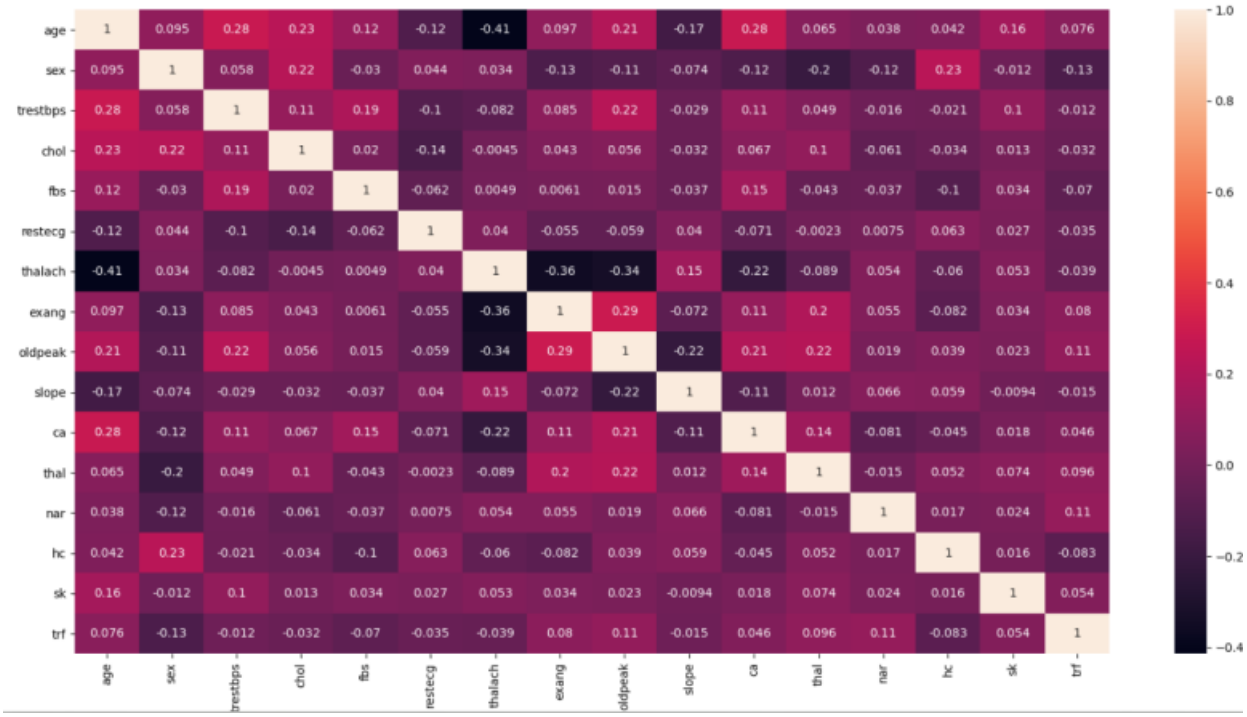

Figura 01 - Correlação entre as variáveis.
Fonte: O autor.

Com o método PCA (*principal component analysis*) da biblioteca *scikit-learn* foi possível tentar reduzir a dimensionalidade do problema. Portanto, foi criado uma função, nomeada como 'diminuir_dimensionalidade()' que recebe os dados como argumento e retorna o resultado do método PCA.

Como a quantidade de amostras para o sexo feminino é bem maior que o do sexo masculino, o método SMOTE da biblioteca *imbalanced learn* foi testado para os dados em questão, com o objetivo de aumentar a accuracia do modelo. Portanto, foi criado uma função, nomeada como, 'balanceamento_das_amostras()' que recebe os dados de treinamento (x e y) e retorna os dados balanceados.

6 Treinamento do modelo

Com base no projeto proposto, bem como os dados disponíveis, um algoritmo de classificação é suficiente para resolver o problema proposto. Os algoritmos de classificação testados foram: Rede neural *perceptron* multicamada e *Random Forest*.

Uma função, nomeada como 'treinar_modelo()', foi desenvolvida para realizar a criação e o treinamento dos modelos utilizando os dados devidamente analisados, tratados e normalizados.

Uma função, nomeada como 'encontrar_melhores_parametros()', foi desenvolvida para realizar o *hyperparameter tuning* dos algoritmos testados. Utilizando o método computacional *GridSearchCV*, da biblioteca do *Scikit-learning*, e setando o espaço de busca dos parametros do modelo foi possível encontrar bons *hyperparametros* para os algoritmos de classificação testados.

7 Resultados do modelo

Com os dados analisados, tratados, normalizados foi possível criar e treinar modelos computacionais fornecidos pela biblioteca do *Scikit-learning*.

7.1 RandomForestClassifier

Esse classificador é uma floresta aleatória bem utilizada na comunidade python para resolver problemas de classificação. A seção abaixo demonstra alguns resultados obtidos com esse algoritmo. O método utilizado para verificar a acurácia do modelo foi o *score* fornecida pela biblioteca *RandomForestClassifier* do *Scikit-learning*.

7.1.1 Testes

O **teste 1.1**, apresentado na tabela 01, foi realizado sem nenhuma análise sobre os dados, simplismente os dados foram tratados (sem NANs e qualitativos), normalizados e separados em treinamento e teste. É importante ressaltar que, para esse algoritmo não é necessário remover features com correlação baixa em relação o *label*, uma vez que é da natureza do *RandomForestClassifer* realizar essa tarefa.

O **teste 1.2** foi realizado conforme o teste anterior, no entanto a função de 'encontrar_melhores_parametros()' foi executada na esperança de encontrar os melhores parâmetros para o modelo. O espaço de busca setado é mostrado conforme o código abaixo. Os parâmetros encontrados na função mencionada são mostrados na tabela 01 no **teste 1.2**.

```
param_grid = {
    'n_estimators': [50, 200],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [2,4,5,6,7,8,10],
    'random_state' :[0,10]
}
```

O **teste 1.3** foi realizado conforme o teste **teste 1.2**, porém além de buscar os melhores parâmetros, as amostras foram balanceadas utilizando a função 'balanceamento_das_amostras()'.

Tabela 01				
Teste	**n_estimators**	**max_depth**	**random_state**	**acurácia**
1.1	100	4	10	0.7758 (77%)
1.2	50	5	10	0.8275 (83%)
1.3	50	5	10	0.7931 (79%)

Portanto, o melhor resultado encontrado para esse algoritmo foi de **83% de acurácia**, que por sua vez foi o melhor resultado encontrado com os testes realizados.

7.2 MLPClassifier

Esse classificador é uma rede neural *perceptron* multicamada bem utilizada na comunidade python em problemas de classificação. A seção abaixo demonstra alguns dos resultados obtidos com esse algoritmo. O método utilizado para verificar a acurácia do modelo foi o *score* fornecida pela biblioteca *MPLClassifier* do *Scikit-learning*.

7.2.1 Testes

O **teste 1.1**, apresentado na tabela 02, foi realizado sem nenhuma análise sobre os dados, simplismente os dados foram tratados (sem NANs e qualitativos), normalizados e separados em treinamento e teste. É importante ressaltar que, para esse algoritmo é necessário remover features com correlação baixa em relação o *label*, uma vez que através de alguns testes o modelo apresentou melhores resultados.

O **teste 1.2** foi realizado conforme o teste anterior, no entanto a função de 'encontrar_melhores_parametros()' foi executada na esperança de encontrar os melhores parâmetros para o modelo. O espaço de busca setado é mostrado conforme o código abaixo. Os parâmetros encontrados na função mencionada é mostrada na tabela 02.

```
parameter_space = {
      'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,),(50,)],
      'solver': ['sgd', 'adam'],
      'random_state': [0,15],
      'max_iter': [100,1500]
 }
```

O **teste 1.3** foi realizado conforme o **teste 1.2**, porém além de buscar os melhores parâmetros, as amostras foram balanceadas utilizando a função 'balanceamento_das_amostras()'.

Tabela 02

Teste	**random_state**	**hidden_layer**	**tol**	**max_iter**	**solver**	**acurácia**
1.1	10	50	0.000001	1000	adam	0.7141 (71%)
1.2	0	(50, 50, 50)	0.000001	2000	adam	0.7413 (74%)
1.3	0	(50, 50, 50)	0.000001	2000	adam	0.7586 (75%)

O **teste 2.1** foi realizado após as análises, tratamentos, normalização, e dados separados em treinamento e testes. Após avaliar o gráfico de correlação e o PCA alguns atributos foram removidos do conjunto de dados, conforme explicado na seção de análise.

O **teste 2.2** foi realizado conforme o **teste 2.1**, porém a função de encontrar melhores parâmetros foi executada. O espaço de busca utilizado é conforme o código apresentado no **teste 1.2**.

O **teste 2.3** foi realizado conforme o **teste 2.2**, porém a função de balancear as amostras foi executada.

Teste	**random_state**	**hidden_layer**	**tol**	**max_iter**	**solver**	**acurácia**
2.1	10	50	0.000001	1000	adam	0.7413 (74%)
2.2	10	50	0.000001	100	adam	0.7413 (74%)
2.3	10	50	0.000001	1500	adam	0.7586 (75%)

Portanto, os resultados encontrados com esse método foram inferiores ao método do *Random Forest*. Assim, o método escolhido para treinar o modelo final do trabalho foi o *Random Forest*.

8 Inferência modelo

Uma vez treinado e salvo o modelo, seja *RandomForestClassifier* ou *MLPClassifier*, foi possível realizar predições do sexo dos pacientes baseado em algumas características. Portanto, foi construído um script nomeado de 'sex_predictor.py' com objetivo de realizar predições no modelo treinado utilizando os dados importados e tratados do arquivo 'newsample.csv'. É importante ressaltar que o formato e a disposição dos dados do csv tem que estar exatamente igual ao arquivo 'newsample.csv', logo caso queira colocar mais amostras bastar replicar as linhas do arquivo.

Uma função nomeada como 'parser()' foi desenvolvida para criação de uma interface de linha de comando amigável, basicamente essa função cria um argumento opcional nomeado como *'- -input_file'* responsável em obter o arquivo csv selecionado pelo usuário com dados para serem inferidos no modelo.

Os dados importados do csv foram armazenados em um *DataFrame* devidamente indexado, para isso foi construído uma função nomeada como 'transformar_csv_dataframe()'. Além disso, foi construido uma função, nomeado como 'tratar_dados_para_predicao()', para tratar valores vazios inseridos no csv, transformar as strings em valores reais e normalizar os dados com **scaler** salvo.

Com isso, foi contruido uma função, nomeado como 'predicao_modelo()', para realizar as predições com os dados devidamente importado e tratado. Com os resultados foi possível exportar um csv com os valores preditos pelo modelo.

9 Conclusão

Sendo assim, o projeto proposto pela empresa foi desenvolvido e os resultados obtidos foram bons. Com um modelo com acurácia boa (em torno de 83%) é possível realizar predições do sexo de pacientes. Entretanto, é possível melhorar os dados para que a acurácia seja melhor, porém devido ao tempo de desenvolvimento do projeto o modelo apresentado foi o obtido com os testes.