Universidade Federal de Goiás
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

Prof. Nádia Félix Felipe da Silva

Modelos de predição de cobertura de planos de saúde

Vinícius Gabriel Santos Vidal
Ronan Vieira do Carmo Junior

Dezembro / 2022

Universidade Federal de Goiás
Instituto de Informática

Disciplina: Inteligência Computacional

Relatório

Primeiro Relatório da participação dos Alunos Vinícius Gabriel e Ronan Vieira do Curso de Engenharia de Computação da Universidade Federal de Goiás, como requisito parcial para Aprovação da Disciplina Inteligência Computacional.

Conteúdo

1. Resumo

Toda solicitação feita através de um plano de saúde é analisada por um sistema que, após análise automática, determina se o procedimento foi autorizado ou não ou se a solicitação depende de um auditor. A operadora (empresa habilitada a comercializar planos de saúde), tem como principal objetivo controlar os gastos gerados pelos beneficiários (pessoa física que contrata a operadora), garantindo que sejam realizados por eles somente serviços do serviço contratados e pertinentes à saúde.

Os auditores são profissionais que geram custos elevados para a operadora. E para tentar resolver esse problema, foi proposto durante o desafio a criação de uma ferramenta capaz de analisar o comportamento histórico dos auditores para prever o provável desfecho para a solicitação. Para isso, foi disponibilizada uma base de dados de treinamento e de teste, ambas preenchidas com dados de solicitações reais.

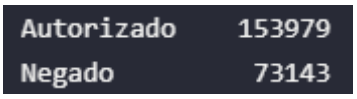
2. Descrição do Conjunto de dado

- A base possui um total de 31 colunas

Coluna	Descrição
NR_SEQ_REQUISICAO	Identificador da Requisição
NR_SEQ_ITEM	Identificador item
DT_REQUISICAO	Data da Requisição
DS_TIPO_GUIA	Descrição do Tipo de Guia
DT_NASCIMENTO	Data de nascimento do solicitante
NR_PRODUTO	Identificador do Produto
DS_TIPO_PREST_SOLICITANTE	Classificação do prestador
DS_CBO	Código do solicitante
DS_TIPO_CONSULTA	Tipo de Consulta
QT_TEMPO_DOENCA	Tempo de Doença
DS_UNIDADE_TEMPO_DOENCA	unidade de tempo da doença
DS_TIPO_DOENCA	Descrição do tipo da doença
DS_INDICACAO_ACIDENTE	Indicação de acidente

Coluna	Descrição
DS_TIPO_SAIDA	Descrição do tipo de Saída
DS_TIPO_INTERNACAO	Descrição do tipo de Internação
DS_REGIME_INTERNACAO	Descrição do regime de internação
DS_CARATER_ATENDIMENTO	Descrição do caráter de atendimento
DS_TIPO_ACOMODACAO	Descrição do tipo de acomodação
DS_INDICACAO_CLINICA	indicação clínica
DS_TIPO_ITEM	Descrição do tipo do item
CD_ITEM	Código item na tabela de plano
DS_ITEM	Descrição do item na tabela de plano
DS_CLASSE	Descrição classe
DS_SUBGRUPO	Descrição Subgrupo
DS_GRUPO	Descrição Grupo
QT_SOLICITADA	Quantidade solicitada
DS_STATUS_ITEM	Status da solicitação

- A base de treinamento é desbalanceada. A quantidade de solicitações autorizadas é maior que o dobro das solicitações não autorizadas



- A coluna DS_TIPO_SAIDA, responsável pela descrição do desfecho do atendimento, está completamente vazia
- As colunas QT_TEMPO_DOENCA, DS_UNIDADE_TEMPO_DOENCA, DS_TIPO_DOENCA estão praticamente vazias. Todas as citadas possuem menos de 1000 ocorrências
- Possui muitos dados descritivos, o que pode dificultar um pouco a análise do modelo

3 Descrição de atividades

Primeiramente os dados foram observados para entendê-los melhor. Em seguida foram feitos alguns testes de tratamento de dados. Durante o primeiro pré-processamento, primeiramente foram escolhidas algumas colunas consideradas relevantes para a análise.

Após a escolha, o primeiro teste de tratamento foi feito: das colunas as escolhidas, os valores vazios foram substituídos por 0. Em seguida, para valores numéricos, foi utilizado o método de tratamento *label encoder* e para valores categóricos o modelo foi *one hot encode.* Esse pré-processamento foi testado nos seguintes modelos de predição: árvore de decisão, *random forest,* KNN, MLP, *Naive Bayes* e DBSCAN. Os resultados desses modelos foram bem próximos, porém o que mais se destacou para esse pré-processamento foi o modelo de MLP.

```
def preproc(df):
    #NUMERIC
    df.QT_DIA_SOLICITADO = df.QT_DIA_SOLICITADO.fillna(0)
    df.QT_TEMPO_DOENCA = df.QT_TEMPO_DOENCA.fillna(0)
    df.CD_GUIA_REFERENCIA = df.CD_GUIA_REFERENCIA.fillna(0)
    df.NR_SEQ_REQUISICAO = df.NR_SEQ_REQUISICAO.fillna(0)
    df.CD_GUIA_REFERENCIA = df.CD_GUIA_REFERENCIA.fillna(0)
    df.CD_ITEM = df.CD_ITEM.fillna(0)
    #CATEGORIC
    df.DS_INDICACAO_ACIDENTE = df.DS_INDICACAO_ACIDENTE.fillna('0')
    df.DS_TIPO_INTERNACAO = df.DS_TIPO_INTERNACAO.fillna('0')
    df.DS_TIPO_ATENDIMENTO = df.DS_TIPO_ATENDIMENTO.fillna('0')
    df.DS_UNIDADE_TEMPO_DOENCA = df.DS_UNIDADE_TEMPO_DOENCA.fillna('0')
    df.DS_TIPO_CONSULTA = df.DS_TIPO_CONSULTA.fillna('0')
    df.DS_TIPO_DOENCA = df.DS_TIPO_DOENCA.fillna('0')
    df.DS_REGIME_INTERNACAO = df.DS_REGIME_INTERNACAO.fillna('0')
    df.DS_TIPO_ACOMODACAO = df.DS_TIPO_ACOMODACAO.fillna('0')
    df.DS_INDICACAO_CLINICA = df.DS_INDICACAO_CLINICA.fillna('0')
    df.CD_CID = df.CD_CID.fillna('0')
    return df
```

Primeiro Pré-processamento

Durante o segundo teste de tratamento de dados, foi feita a utilização do *simple imputer* nas colunas numéricas, utilizando *strategy=means* como parâmetro. Durante os testes, os resultados pioraram consideravelmente, então essa estratégia foi descartada.

```
def preproc(df):
    #NUMERIC
    imp = SimpleImputer(missing_values=0.0, strategy='mean')
    dbTrain[['QT_SOLICITADA', 'QT_DIA_SOLICITADO', 'QT_TEMPO_DOENCA']] = imp.fit_transform(dbTrain[['QT_SOLICITADA', 'QT_DIA_SOLICITADO', 'QT_TEMPO_DOENCA']])

    #CATEGORIC
    df.DS_INDICACAO_ACIDENTE = df.DS_INDICACAO_ACIDENTE.fillna('0')
    df.DS_TIPO_INTERNACAO = df.DS_TIPO_INTERNACAO.fillna('0')
    df.DS_TIPO_ATENDIMENTO = df.DS_TIPO_ATENDIMENTO.fillna('0')
    df.DS_UNIDADE_TEMPO_DOENCA = df.DS_UNIDADE_TEMPO_DOENCA.fillna('0')
    #df.DS_TIPO_CONSULTA = df.DS_TIPO_CONSULTA.fillna('0')
    df.DS_TIPO_DOENCA = df.DS_TIPO_DOENCA.fillna('0')
    df.DS_REGIME_INTERNACAO = df.DS_REGIME_INTERNACAO.fillna('0')
    df.DS_TIPO_ACOMODACAO = df.DS_TIPO_ACOMODACAO.fillna('0')
    df.DS_INDICACAO_CLINICA = df.DS_INDICACAO_CLINICA.fillna('0')
    df.CD_CID = df.CD_CID.fillna('0')

    return df
```

Segundo Pré-processamento

No terceiro teste de tratamento, foi feita a utilização das colunas DT_REQUISICAO e DT_NASCIMENTO, com o objetivo de criar uma nova coluna de idade para fazer a análise. Para isso, foi feita a conversão dessas duas colunas de JD(Julian Date) para data gregoriana. Foi calculada a diferença entre as duas colunas com o objetivo de obter a idade do solicitante

no momento da requisição. Essa diferença foi armazenada em uma coluna chamada IDADE e após isso foi feito o mesmo pré-processamento a ser testado primeiro (substituição de valores vazios por 0). Os modelos foram aplicados e percebeu-se uma leve piora nos resultados com relação ao primeiro pré-processamento. Então, após os 3 testes, concluiu-se que o melhor processamento dentre os 3 foi o primeiro.

```
dbTest['DT_REQUISICAO'] = pd.to_datetime(dbTest['DT_REQUISICAO'], origin='julian', unit='D')
dbTest['DT_NASCIMENTO'] = pd.to_datetime(dbTest['DT_NASCIMENTO'], origin='julian', unit='D')
datas = pd.DataFrame((dbTrain['DT_REQUISICAO'] - dbTrain['DT_NASCIMENTO']).astype('timedelta64[D]')/365, columns=['IDADE'])
datas.loc[np.isnan(datas['IDADE']), 'IDADE'] = 0
```

Terceiro Pré-processamento

Após decidir o melhor modelo de predição e pré-processamento, foram feitos testes de alteração de parâmetros do modelo MLP. Para o valor do atributo *optimizer,* foram testados os seguintes valores: SGD, RMSprop, *Adam*, *AdamW, Adadelta, Adagrad, Adamax, Adafactor, Nadam e* Ftrl. Os melhores resultados obtidos foram resultados da utilização de *optimizer=Nadam.* Para *loss,* os valores testados foram: *binary_crossentropy, categorical_crossentropy e mean_squared_error.* Os melhores resultados foram obtidos com *loss=mean_squared_error.*

Foi também efetuado testes com diferentes tamanho de *batch e* quantidades de épocas. Os melhores resultados foram obtidos com *batch_size=256* e *epoch=10.*

```
def create_model(optimizer="adam", dropout=0.1, init='uniform', nbr_features=141, dense_nparams=256):
    model = Sequential()
    model.add(Dense(dense_nparams, activation='relu', input_shape=(nbr_features,), kernel_initializer=init,))
    model.add(Dropout(dropout), )
    model.add(Dense(2, activation='sigmoid'))
    model.compile(loss='mean_squared_error', optimizer='Nadam',metrics=["accuracy"])
    return model
```

```
keras_estimator.fit(x_train, y_train, epochs = 15, batch_size = 256)
```

Modelo final utilizado durante o treinamento e teste.

4 Análise dos Resultados

Após todos os testes e análises que foram feitos, houve uma melhora de cerca de 0.2 de F1-score em relação ao primeiro modelo, o que é uma melhora considerável. O primeiro F1-score foi de 0,69476 e o mais alto foi de 0,71074.

Ao participar da competição, percebeu-se que há uma enorme dificuldade em tratar os dados de forma que o tratamento feito seja relevante para o modelo. Alguns tratamentos mais complicados que outros podem resultar até mesmo em uma piora no modelo de predição.
Os resultados dos testes mostram que apesar de ainda ser necessário uma maior quantidade de dados e um melhor tratamento dos mesmos, a realidade em que são utilizados computadores para análise de solicitações para o plano de saúde está bem próxima de se tornar realidade.

Bibliografia

- Documentação Keras. https://keras.io. Acessado em: 09/12/2022
- 6.4. Imputation of missing values. https://scikit-learn.org/stable/modules/impute.html#:~:text=For%20various%20reasons%2C%20many%20real,all%20have%20and%20hold%20meaning. Acessado em: 07/12/2022
- sklearn.ensemble.RandomForestClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. Acessado em: 07/12/2022