PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Rodrigo Marques Pessoa

PREDIÇÃO DE NECESSIDADE DE INTERNAÇÃO DE PACIENTES A PARTIR DA INFORMAÇÃO DAS UNIDADES BÁSICAS DE SAÚDE

Campinas
2022

Rodrigo Marques Pessoa

PREDIÇÃO DE NECESSIDADE DE INTERNAÇÃO DE PACIENTES A PARTIR DA INFORMAÇÃO DAS UNIDADES BÁSICAS DE SAÚDE

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Campinas
2022

SUMÁRIO

1. Introdução

1.1. Contextualização

As técnicas avançadas de análise de dados e aprendizagem de máquina trazem um ganho exponencial: permitem olhar adiante e auxiliar a medicina preventiva. Com base em dados de pacientes, é possível usar a análise de dados de uma maneira mais inteligente para sugerir cuidados preventivos em determinadas épocas do ano, ou até mesmo promover um estilo de vida mais saudável.

O repertório de informações que os cientistas precisam podem ser encontrados em softwares e equipamentos médicos que emitem dados.

Podemos minerar os registros de dados que os hospitais emitem sobre seus pacientes por meio de um CRM. A partir das informações selecionadas, é possível criar campanhas e ações preventivas.

Com algorítimos matemáticos cada vez mais precisos e baseado nesse histórico de dados, a máquina consegue alertar sobre possíveis doenças, identificar grupos de tendências, entre outros fatores.

1.2. O problema proposto

Para tentar prever a necessidade de internação utilizaremos os dados das unidades de saúde com os dados dos pacientes e dados dos diagnósticos realizados pelos médicos. O modelo preditivo utilizará os dados públicos da prefeitura de Curitiba que está disponível para download através do seguinte endereço eletrônico:
https://www.curitiba.pr.gov.br/dadosabertos/busca/?pagina=7.

1.3. Objetivos

A falta de leito em hospitais é um problema constante encontrado em várias cidades brasileiras. Gerenciar essas vagas é cada vez mais essencial e ter essa informação o quanto antes pode ajudar a estabelecer prioridades de atendimento.

A medida que consultas são realizadas nas diversas unidades de saúde da cidade, essas informações poderiam ser enviadas a um servidor central que mediante a estes dados pode aplicar um modelo preditivo e contabilizar em tempo real a necessidade de utilização de leitos em hospitais, ou tomar ações preventivas, por exemplo.

2. Coleta de Dados

Para tentar prever a necessidade de internação utilizaremos os dados das unidades de saúde com os dados dos pacientes e dados dos diagnósticos realizados pelos médicos. O modelo preditivo utilizará os dados públicos da prefeitura de Curitiba que está disponível para download através do seguinte endereço eletrônico: http://www.curitiba.pr.gov.br/dadosabertos/consulta/?grupo=1.

Segue abaixo um dicionário de dados com as variáveis disponíveis e as que foram escolhidas para serem preditoras do modelo. As variáveis em verde foram escolhidas pois podem apresentar alguma correlação com a variável classe(em azul), como por exemplo dados socioeconômicos dos pacientes (pacientes com menor poder aquisitivo ou que não tem infraestrutura como coleta de lixo, água tratada poderia necessitar de internações constantes). Já as variáveis em vermelhos foram descartadas pois não apresentam nenhuma correlação para o estudo deste projeto:

Dicionário de Dados	
Nome do Campo	**Descrição**
Data do Atendimento	Data de Realização do Atendimento
Data de Nascimento	Data de Nascimento do Paciente
Sexo	Sexo do Paciente
Código do Tipo de Unidade	Código do Tipo de Unidade de Atendimento
Tipo de Unidade	Tipo de Unidade de Atendimento
Código da Unidade	Código da Unidade de Atendimento
Descrição da Unidade	Descrição da Unidade de Atendimento
Código do Procedimento	Código do Procedimento Realizado
Descrição do Procedimento	Descrição do Procedimento Realizado
Código do CBO	Código da Ocupação do Profissional
Descrição do CBO	Descrição da Ocupação do Profissional
Código do CID	Código do Diagnóstico
Descrição do CID	Descrição do Diagnóstico
Solicitação de Exames	Indica se ocorreu solicitação de Exames
Qtde Prescrita Farmácia Curitibana	Qtde medicamentos prescritos na Farmácia
Qtde Dispensada Farmácia Curitibana	Qtde medicamentos dispensados na Farmácia
Qtde de Medicamento Não Padronizado	Qtde de Medicamento Não Padronizado
Encaminhamento para Atendimento Especialista	Indica se houve encaminhamento para Atendimento de Especialista
Área de Atuação	Área de Atuação
Desencadeou Internamento	Indica se desencadeou Internamento
Data do Internamento	Data do Internamento do paciente
Estabelecimento Solicitante	Estabelecimento que solicitou internamento
Estabelecimento Destino	Estabelecimento que houve a internação
CID do Internamento	Código do diagnóstico do internamento
Tratamento no Domicílio	Tipo de Tratamento de Água no domicílio

Abastecimento	Tipo de Abastecimento de Água no domicílio
Energia Elétrica	Indica se há energia elétrica no domicílio
Tipo de Habitação	Tipo de habitação no domicílio
Destino Lixo	Destino do lixo no domicílio
Fezes/Urina	Destino das fezes/urina no domicílio
Cômodos	Qtde de Cômodos no domicílio
Em Caso de Doença	Serviços procurados em caso de doença
Grupo Comunitário	Grupo Comunitário que o paciente participa
Meio de Comunicação	Meios Comunicação utilizados no domicílio
Meio de Transporte	Meios de Transporte utilizados no domicílio
Município	Município do paciente
Bairro	Bairro do paciente

Automatizando a captura de dados com Airflow

Utilizaremos o airflow para criar um pipeline para importar os dados automaticamente do site da prefeitura conforme indicado no código abaixo:

```
with DAG(
    'dag-pipeline-Interna-v1',
    schedule_interval=timedelta(minutes=100),
    catchup=False,
    default_args=default_args
    ) as dag:

     start = DummyOperator(task_id="start")

     with TaskGroup("etl", tooltip="etl") as etl:
         t0 = BashOperator(
                 dag=dag,
                 task_id='download_dataset',
                 bash_command="""
                 cd {0}/featurestore
                curl -o interna.csv  https://mid.curitiba.pr.gov.br/dadosabertos/sespaenfermagem/2021-09-06_Sistema_E-Saude_Enferma
                 """.format(pathScript)
         )
         [t0]
```

3. Processamento/Tratamento de Dados

Como o os dados apresentam as informações dos últimos 3 meses de atendimento das unidades de saúde, contendo quase 1.000.000 de registros utilizaremos uma amostragem para realizar o modelo preditivo. Trabalharei com uma amostra de 664 registros, que foi calculado levando em consideração 5% de erro amostral e 99% de nível de confiança. Como os dados não foram coletados com o intuito de utilização para o modelo de predição existe uma grande quantidade de dados ausentes e um trabalho de limpeza e padronização será necessário. Outra questão é fazer o balanceamento das classes já que a proporção de necessidade de internações comparada as da não necessidade são muito desproporcionais.

Normalizando atributos numéricos

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
campos_numericos = ['Comodos', 'idade']

dados_model[campos_numericos] = scaler.fit_transform(dados_model[campos_numericos])
```

Pré-processamento dos dados categóricos

```
dados_model_final = pd.get_dummies(dados_model)
dados_model_final

encoded = list(dados_model_final.columns)
print "{} total features after one-hot encoding.".format(len(encoded))

print encoded
```

Automatizando pré-processamento com Airflow

Utilizaremos o airflow para criar um pipeline para realizar o pré-processamento dos dados automaticamente conforme indicado no código abaixo:

```
with TaskGroup("preProcessing", tooltip="preProcessing") as preProcessing:
    t1 = BashOperator(
        dag=dag,
        task_id='encoder_dataset',
        bash_command="""
        cd {0}
        python etl_preprocessing.py {1} {2} {3}
        """.format(pathScript, pathPre, pathPos, pathEncoder)
    )
    [t1]
```

4. Análise e Exploração dos Dados

Explorando os dados

```
import numpy as np
import pandas as pd
from time import time
from datetime import datetime
from IPython.display import display # Permite a utilização da função display() para DataFrames.

%matplotlib inline

data = pd.read_csv("amostra_internacao.csv",delimiter=';')
data_internamento = data[data['Internamento'] =='Sim']

print('Dados antes do pré-processamento')
display(data.head(n=1))

print('Dados de pacientes com necessidade de internamento')
display(data_internamento.head(n=10))
```

Dados antes do pré-processamento

	datanasc	Sexo	ciddesc	solex	encesp	tratdom	energia	habitacao	lixo	Fezes/Urina	Comodos	Internamento
0	04/10/2012 00:00	F	EXAME MEDICO GERAL	Nao	Sim	SEM TRATAMENTO	Sim	TIJOLO/ALVENARIA COM REVESTIMENTO	COLETADO	SISTEMA DE ESGOTO	5	Nao

Dados de pacientes com necessidade de internamento

	datanasc	Sexo	ciddesc	solex	encesp	tratdom	energia	habitacao	lixo	Fezes/Urina	Comodos	Internamento
347	10/06/1990 00:00	F	ABDOME AGUDO	Sim	Nao	SEM TRATAMENTO	Sim	TIJOLO/ALVENARIA COM REVESTIMENTO	COLETADO	SISTEMA DE ESGOTO	5	Sim

- O número total de registros.
- O número de indivíduos que precisaram de internamento.
- O número de indivíduos que não precisaram de internamento.
- O percentual de indivíduos que precisou de internamento.

```
n_registros = data.shape[0]

internamento_raw = data['Internamento']
internamento_model = np.where(internamento_raw == 'Sim', 1, 0)


n_intenamento = np.where(internamento_raw == 'Sim', 1, 0).sum()
n_nao_intenamento = np.where(internamento_raw == 'Nao', 1, 0).sum()

internamento_percentual = float(n_intenamento * 100)/n_registros

# Exibindo os resultados
print "Total numero de registros: {}".format(n_registros)
print "Necessitou internamento: {}".format(n_intenamento)
print "Não Necessitou internamento: {}".format(n_nao_intenamento)
print "Percentual necessitou internamento: {:.2f}%".format(internamento_percentual)
```

```
Total numero de registros: 664
Necessitou internamento: 317
Não Necessitou internamento: 347
Percentual necessitou internamento: 47.74%
```

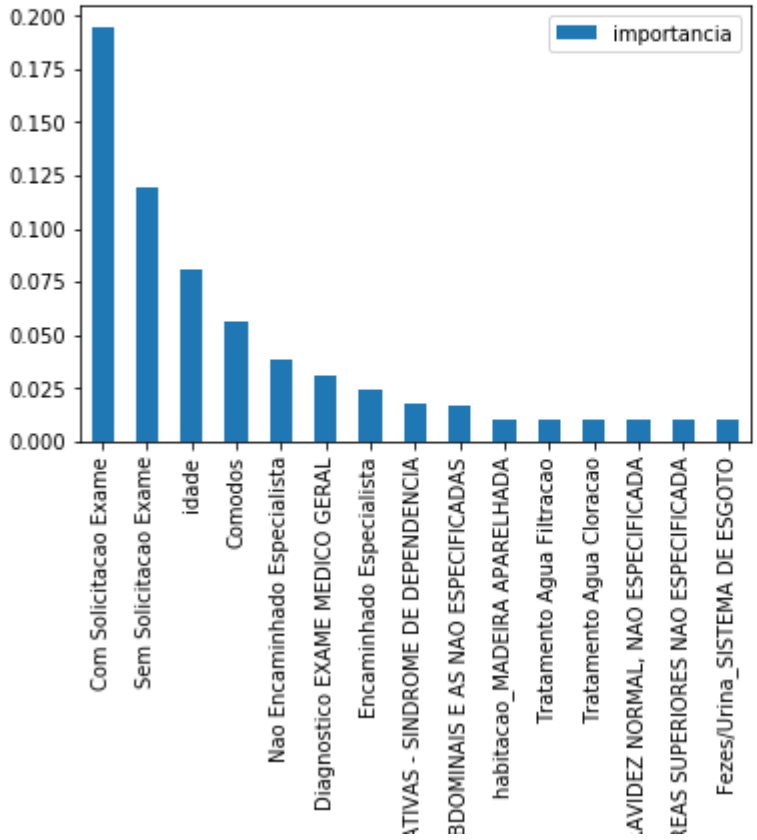
Avaliando a importância dos atributos

```
def define_nome_coluna(col):
    if col == 'solex_Sim':
        return 'Com Solicitacao Exame'
    elif col == 'solex_Nao':
        return 'Sem Solicitacao Exame'
    elif col == 'encesp_Nao':
        return 'Nao Encaminhado Especialista'
    elif col == 'encesp_Nao':
        return 'Nao Encaminhado Especialista'
    elif col == 'encesp_Sim':
        return 'Encaminhado Especialista'
    elif col == 'tratdom_FILTRACAO':
        return 'Tratamento Agua Filtracao'
    elif col == 'tratdom_CLORACAO':
        return 'Tratamento Agua Cloracao'
    return  col.replace("ciddesc_", "Diagnostico ")

importances = pd.DataFrame({'feature':X_train.columns,'importancia':np.round(clf_RandomForest.feature_importances_,3)})
importances = importances.sort_values('importancia',ascending=False).set_index('feature')
#importances['feature'] = importances['feature'].map(define_nome_coluna)
importances = importances.rename(lambda x: define_nome_coluna(x))
importances.head(15)
```

```
importancesf = pd.DataFrame(importances[:15])
importancesf.plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f12e358>
```

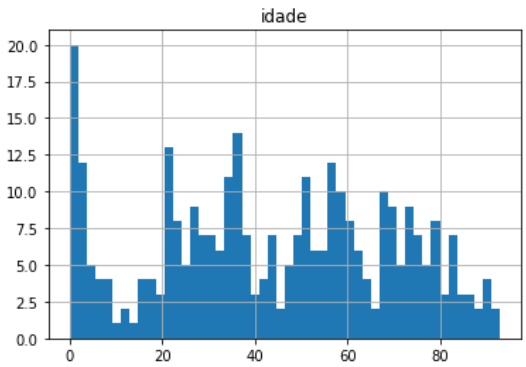


Distribuição do campo Idade

```
now = datetime.now()
data['idade'] = data['datanasc'].apply(lambda x: ((now - datetime.strptime(x, '%d/%m/%Y %H:%M') ).days / 365))

data_internamento = data[data['Internamento'] =='Sim']
data_internamento.hist(column="idade",bins=50)

dados_model_temp1 = data.drop('Internamento', axis = 1)
dados_model = dados_model_temp1.drop('datanasc', axis = 1)
```



Outra informação importante que podemos visualizar através do histograma de idade é que existe uma necessidade maior de internação nos primeiros anos de vida mostrando que as prefeituras devem investir em informação e prevenção das mães e crianças para diminuir o número de internação.

5. Criação de Modelos de Machine Learning

Optei por utilizar o algoritimo Random Forest ou floresta aleatória. Uma das vantagens da floresta aleatória é que ela pode ser usada para tarefas de regressão e classificação e que é fácil visualizar a importância relativa que atribui aos recursos de entrada.

O Random Forest também é considerado um algoritmo muito prático e fácil de usar, porque os hiperparâmetros padrão geralmente produzem um bom resultado de previsão. O número de hiperparâmetros também não é tão alto e eles são fáceis de entender.

Florestas Aleatórias também são muito difíceis de serem superadas em termos de desempenho. É claro que você pode sempre encontrar um modelo que possa ter um desempenho melhor, como uma rede neural, mas isso geralmente leva muito mais tempo no desenvolvimento. E além disso, eles podem lidar com vários tipos de recursos diferentes, como categórico e numérico que é o nosso caso.

Embaralhar e dividir os dados

```
from sklearn.cross_validation import train_test_split

X_train, X_test, y_train, y_test = train_test_split(dados_model_final,
                                                    internamento_model,
                                                    test_size = 0.2,
                                                    random_state = 0)

# Show the results of the split
print "Conjunto de Treino: {}".format(X_train.shape[0])
print "Conjunto de Teste: {}".format(X_test.shape[0])
```

```
Conjunto de Treino: 531
Conjunto de Teste: 133
```

Modelo Machine Learning

```
def train_predict(learner, sample_size, X_train, y_train, X_test, y_test, X_prod, y_prod):


    results = {}

    start = time()
    learner = learner.fit(X_train[:int(sample_size)], y_train[:int(sample_size)])
    end = time()

    results['train_time'] = end - start

    start = time()
    predictions_test = learner.predict(X_test)
    predictions_train = learner.predict(X_train)
    end = time()

    results['pred_time'] = end - start

    report = classification_report(y_test, predictions_test)
    print('-----------------------------')
    print('report')
    print(report)
    print('-----------------------------')
```

```
from sklearn.metrics import accuracy_score

results['acc_train'] = accuracy_score(y_train, predictions_train)

ac_teste = accuracy_score(y_test, predictions_test)

matrix = confusion_matrix(y_test, predictions_test)

# Imprimindo a Confusion Matrix
print('matrix confusao teste:')
print(matrix)
print(' ')


results['acc_test'] =  ac_teste


from sklearn.metrics import fbeta_score

results['f_train'] =  fbeta_score(predictions_train, y_train, beta=2)

fbs_teste =  fbeta_score(predictions_test, y_test, beta=2)
results['f_test'] = fbs_teste

# Success
print("{} registros treinados = {}".format(learner.__class__.__name__, sample_size))
print("{} acuracia teste= {}".format(learner.__class__.__name__,ac_teste))
print("{} f-score(2) teste= {}".format(learner.__class__.__name__,fbs_teste))

# Return the results
return results
```

Implementação: Tuning do modelo

```
clf_comum = RandomForestClassifier(random_state=999)


parameters = {"criterion": ['gini','entropy'],
              "max_features":[5,10,15,20,'auto'],
              "max_depth":[1, 2, 3, 10],
              "n_estimators" : [1,2, 3, 10, 15,20],
              "bootstrap" : [True,False],
              "random_state" : [999]}


scorer = make_scorer(fbeta_score, beta=2)

grid_obj = GridSearchCV(clf_comum,param_grid=parameters,scoring=scorer)

grid_fit = grid_obj.fit(X_train, y_train)

clf_otimizado = grid_fit.best_estimator_

print "Melhor estimador:\n------"
print(clf_otimizado)

predicao_comum = (clf_comum.fit(X_train, y_train)).predict(X_test)
predicao_otimizada = clf_otimizado.predict(X_test)

# Reportar os scores de antes e de depois
print "Modelo não otimizado\n------"
print "Acuracia dos dados de teste: {:.4f}".format(accuracy_score(y_test, predicao_comum))
print "F-score dos dados de teste: {:.4f}".format(fbeta_score(y_test, predicao_comum, beta = 2))
print "\nModelo Otimizado\n------"
print "Acuracia dos dados de teste: {:.4f}".format(accuracy_score(y_test, predicao_otimizada))
print "F-score dos dados de teste: {:.4f}".format(fbeta_score(y_test, predicao_otimizada, beta = 2))
```

```
Melhor estimador:
------
RandomForestClassifier(bootstrap=False, class_weight=None, criterion='gini',
            max_depth=3, max_features=20, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=15, n_jobs=1, oob_score=False, random_state=999,
            verbose=0, warm_start=False)
Modelo não otimizado
------
Acuracia dos dados de teste: 0.8421
F-score dos dados de teste: 0.8615
```

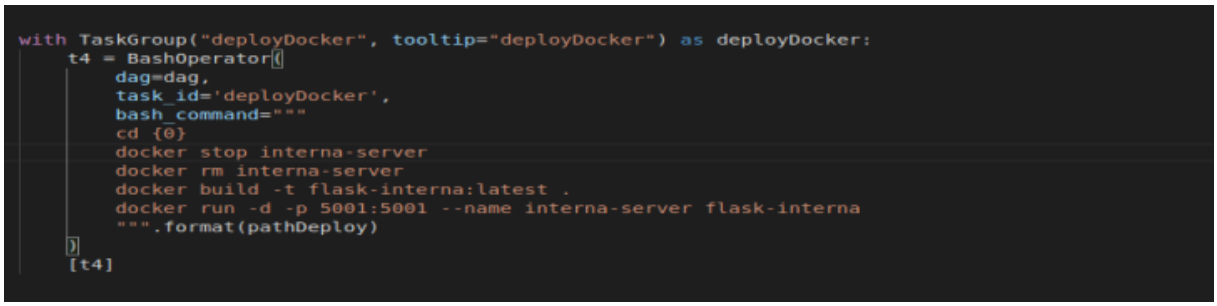
Automatizando processamento do modelo com Airflow

Utilizaremos o airflow para criar um pipeline para realizar o processamento do modelo automaticamente conforme indicado no código abaixo:

```
with TaskGroup("model", tooltip="model") as model:
    t2 = BashOperator(
        dag=dag,
        task_id='modelo',
        bash_command="""
        cd {0}
        python ml_sklearn.py {1} {2} {3} {4} {5}
        """.format(pathScript,pathPos, "InternaClassificacao", "ModeloInterna", 10, 0)
    )
    [t2]
```

Automatizando a disponibilização do Modelo com Airflow

Utilizaremos o airflow para criar um pipeline para realizar a disponibilização do modelo automaticamente conforme indicado no código abaixo. O modelo será disponibilizado em um contêiner docker e depois utilizado a tecnologia flask do python para disponibilizar o acesso ao modelo via api.



```
with TaskGroup("deployDocker", tooltip="deployDocker") as deployDocker:
    t4 = BashOperator(
        dag=dag,
        task_id='deployDocker',
        bash_command="""
        cd {0}
        docker stop interna-server
        docker rm interna-server
        docker build -t flask-interna:latest .
        docker run -d -p 5001:5001 --name interna-server flask-interna
        """.format(pathDeploy)
    )
    [t4]
```

6. Interpretação dos Resultados

Um dado interessante é que a partir da análise das características importantes retornados pelo modelo Random Forest foi possível ver que o tipo de tratamento de água e o sistema de saneamento básico(destaca em vermelho acima) tem uma importância na determinação da necessidade de internação o que indica que as prefeituras deveriam investir mais em saneamento básico para tentar diminuir o número de internação e liberação das vagas dos hospitais para casos realmente que não podem ser evitados com saneamento.

Melhorias

Utilizamos os dados disponíveis para realizar a previsão. Estes dados não foram coletados com uma visão de utilização para predição e, portanto, podem não ser os mais indicados, mas como nosso intuito e apenas educacional isso não será um fator crítico. No entanto em um ambiente real poderíamos realizar um trabalho junto a prefeitura para obtermos outros dados que não estão disponíveis atualmente, como por exemplo, os dados dos resultados de exames de sangue, fezes, urina e eletrocardiograma que poderiam ser feitos nestas unidades básicas de atendimento a fim de melhorarmos a precisão das previsões.

7. Apresentação dos Resultados

Algoritmos e técnicas

Um dos grandes problemas no aprendizado de máquina é o overfitting, mas na maioria das vezes isso não acontece tão fácil para um classificador de floresta aleatório. Isso porque, se houver árvores suficientes na floresta, o classificador não preparará o modelo. A principal limitação da Random Forest é que um grande número de árvores pode tornar o algoritmo lento e ineficaz para previsões em tempo real. Em geral, esses algoritmos são rápidos de treinar, mas muito lentos para criar previsões depois de treinados. Uma previsão mais precisa requer mais árvores, o que resulta em um modelo mais lento. Na maioria das aplicações do mundo real, o algoritmo de floresta aleatória é rápido o suficiente e uma alternativa é a utilização de vários processadores com o hiperparâmetro n_jobs.

Como podemos ver nos dados abaixo o modelo conseguiu uma boa performance se comparada ao modelo de benchmark o que mostra que o modelo escolhido atende as nossas necessidades.

Métricas

Espera-se que o modelo seja capaz de predizer com um índice alto de precisão as consultas que estão sendo realizada nas diversas unidades de saúde da cidade se necessitam ou não de internação.
Utilizaremos o indicador F-score com beta de 2 pois precisamos de alto recall, ou seja, realmente identificar as consultas que precisam de internação, pois se identificarmos com baixo recall vamos precisar de mais leitos do que estamos realmente prevendo. Assim, uma precisão baixa não é uma situação preocupante pois estamos prevendo leitos a mais do que realmente vamos precisar.

Modelo Benchmark

```
clf_A = DummyClassifier(strategy='constant',random_state=999,constant=1)

samples = len(y_train)

results = {}

clf_name = clf_A.__class__.__name__
results[clf_name] = {}

train_predict(clf_A, samples, X_train, y_train, X_test, y_test, dados_model_final,internamento_model)
```

```
------------------------------
report
             precision    recall  f1-score   support

          0       0.00      0.00      0.00        69
          1       0.48      1.00      0.65        64

avg / total       0.23      0.48      0.31       133

------------------------------
matrix confusao teste:
[[ 0 69]
 [ 0 64]]

DummyClassifier registros treinados = 531
DummyClassifier acuracia teste= 0.481203007519
DummyClassifier f-score(2) teste= 0.536912751678
```

Modelo Escolhido

```
clf_RandomForest = RandomForestClassifier(random_state=999)


registros = len(y_train)

# Colete os resultados dos algoritmos de aprendizado
results = {}
clf_name = clf_RandomForest.__class__.__name__
results[clf_name] = {}
train_predict(clf_RandomForest, registros, X_train, y_train, X_test, y_test, dados_model_final,internamento_model)
```

```
------------------------------
report
             precision    recall  f1-score   support

          0       0.88      0.81      0.84        69
          1       0.81      0.88      0.84        64

avg / total       0.84      0.84      0.84       133

------------------------------
matrix confusao teste:
[[56 13]
 [ 8 56]]

RandomForestClassifier registros treinados = 531
RandomForestClassifier acuracia teste= 0.842105263158
RandomForestClassifier f-score(2) teste= 0.823529411765
```

APÊNDICE

Tela do Aiflow executando as tarefas