Nanodegree Engenheiro de Machine Learning

Projeto Final

Analise e Predição em Consultas Médicas Marcadas

Aron Stall
23 de setembro de 2018

I. Definição

Visão Geral do Projeto

Este projeto visa analisar e realizar uma predição de milhares de consultas medicas, analisando e predizendo se o paciente irá comparecer a consulta medica ou não. Esta analise será realizada através do *dataset* disponível no *Kaggle*, chamado **Medical Appointment No Shows**, disponível neste link.

A analise será realizada afim de entender quais são os principais tipos de pessoas que costumam faltar nas consultas medicas, com base nos dados que estão disponível no *dataset*. A analise identificará se existe uma relação entre esses tipos de pessoas, se são homens ou mulheres que mais costumam faltar nas consultas médicas, pessoas que receberam lembrete em SMS costumam faltar nas consultas medicas, entre outras analises.

A predição dos dados será realizada através do *dataset*, utilizando algorítimos de *Machine Learning*, que ira predizer se o paciente irá comparecer a consulta médica ou não. Esta predição analisara os valores das colunas do *dataset*, encontrando padrões nos dados que podem definir se o paciente irá faltar na consulta medica ou não.

O conjunto de dados, o *dataset*, contem 14 colunas e um total de 110.527 linhas. Este *dataset* é perfeito para a analise e predição descrita, pois ele foi disponibilizado para este proposito, incluindo uma coluna que identifica, se o paciente compareceu a consulta medica ou não. As colunas do *dataset* são as seguintes *PatientId*, *AppointmentID*, *Gender*, *ScheduledDay*, *AppointmentDay*, *Age*, *Neighbourhood*, *Scholarship*, *Hipertension*, *Diabetes*, *Alcoholism*, *Handcap*, *SMS_received* e *No-show*.

Descrição do Problema

Inúmeras consultas medicas são marcadas, e os pacientes acabam faltando a elas. Será realizado uma analise no *dataset* identificando padrões, e uma predição nos dados do *dataset*, utilizando Machine Learning, predizendo se o paciente irá comparecer a consulta médica ou não.

Será utilizado a linguagem de programação *Python* para realizar tanto a analise quanto a predição dos dados. As bibliotecas *Pandas*, *NumPy*, *Matplotlib* serão utilizadas para a analise dos dados. A biblioteca *scikit-learn* será utilizada para realizar a predição nos dados com os algorítimos de *Machine Learning GaussianNB*, *DecisionTreeClassifier*, *RandomForestClassifier*, *AdaBoostClassifier*, *GradientBoostingClassifier*, *SVC*, *LogisticRegression* e *SGDClassifier*. A calibração do modelo será feito através do *scikit-learn*, utilizando o algorítimo *GridSearchCV*. Todo o desenvolvimento será utilizando o ambiente de desenvolvimento em *Python* chamado *Jupyter Notebook*.

Ao final da analise e da predição, será possível identificar quais são os principais pacientes que mais costumam faltar nas consultas medicas e o melhor algorítimo de Machine Learning para predizer se o paciente irá comparecer a consulta medica ou não. Toda a analise será descrita de forma fácil e utilizando gráficos para melhor entendimento.

Métricas

O projeto utilizara técnicas de aprendizado supervisionado, por este motivo utilizara os principais métodos de avaliação de métricas para este tipo de *Machine Learning*, que são os mais utilizados pelo mercado, e estão disponível na biblioteca do *scikit-learn*. Será utilizado os métodos *accuracy_score*, *fbeta_score e roc_auc_score* para calcular a pontuação que o algorítimo de *Machine Learning* conseguir atingir. A técnica de *Train/Test Split* será utilizada para dividir os dados do *dataset* em treino e teste.

O projeto ira predizer se o paciente irá comparecer a consulta medica, utilizando *Machine Learning*, com uma *pontuação* de no mínimo de 0.70 em *accuracy*, *fbeta* e *roc/auc*. Será realizado inúmeros testes com diversos algorítimos, para saber qual será o melhor algorítimo, o que terá a maior pontuação.

II. Análise

Exploração dos Dados

Os dados do *dataset* estão distribuídos em 14 colunas e 110.527 linhas, as colunas são descritas da seguinte forma:

- **PatientId** – A identificação do paciente.
- **AppointmentID** – A identificação da consulta médica.
- **Gender** – Sexo do paciente, sendo “M” para masculino e “F” para feminino.
- **ScheduledDay** – O dia em que a consulta foi marcada.
- **AppointmentDay** – O dia em que a consulta foi ou iria ser realizada.
- **Age** – Idade do paciente.
- **Neighbourhood** – Bairro onde a consulta foi ou iria ser realizada.
- **Scholarship** – Verdadeiro ou falso, referente se o paciente esta ingresso no programa social Bolsa Família.
- **Hipertension** – Verdadeiro ou falso, referente se o paciente é hipertenso.
- **Diabetes** – Verdadeiro ou falso, referente se o paciente é diabético.
- **Alcoholism** – Verdadeiro ou falso, referente se o paciente bebe bebidas alcoólicas.
- **Handcap** – Verdadeiro ou falso, referente se o paciente tem algum problema físico, mental ou social.
- **SMS_received** – Valor numérico com a quantidade de mensagem que o paciente recebeu para lembrar da consulta médica.
- **No-show** – Verdadeiro ou falso, referente se o paciente faltou na consulta médica ou não.

Uma amostra das cinco primeiras linhas do *dataset* pode ser visualizados na ***Figura 1*** e ***Figura 2***.

	PatientId	AppointmentID	Gender	ScheduledDay	AppointmentDay	Age	Neighbourhood
0	2.987250e+13	5642903	F	2016-04-29T18:38:08Z	2016-04-29T00:00:00Z	62	JARDIM DA PENHA
1	5.589978e+14	5642503	M	2016-04-29T16:08:27Z	2016-04-29T00:00:00Z	56	JARDIM DA PENHA
2	4.262962e+12	5642549	F	2016-04-29T16:19:04Z	2016-04-29T00:00:00Z	62	MATA DA PRAIA
3	8.679512e+11	5642828	F	2016-04-29T17:29:31Z	2016-04-29T00:00:00Z	8	PONTAL DE CAMBURI
4	8.841186e+12	5642494	F	2016-04-29T16:07:23Z	2016-04-29T00:00:00Z	56	JARDIM DA PENHA

***Figura 1**: Amostra das 5 primeiras linhas, com as colunas **PatientId**, **AppointmentID**, **Gender**, **ScheduledDay**, **AppointmentDay**, **Age** e **Neighbourhood**.*

Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received	No-show
0	1	0	0	0	0	No
0	0	0	0	0	0	No
0	0	0	0	0	0	No
0	0	0	0	0	0	No
0	1	1	0	0	0	No

***Figura 2**: Amostra das 5 primeiras linhas, com as colunas* **Scholarship**, **Hipertension**, **Diabetes**, **Alcoholism**, **Handcap**, **SMS_received** e **No-show**.

A biblioteca *Pandas* do *Python*, utilizada neste projeto, contem uma função chamada *describe*, que retorna as principais informações estatística do *dataset*, informações como quantidade, média, desvio padrão, valores máximo e mínimo, entre outros, conforme pode ser visualizado na ***Figura 3***.

	PatientId	AppointmentID	Age	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received
count	1.105270e+05	1.105270e+05	110527.000000	110527.000000	110527.000000	110527.000000	110527.000000	110527.000000	110527.000000
mean	1.474963e+14	5.675305e+06	37.088874	0.098266	0.197246	0.071865	0.030400	0.022248	0.321026
std	2.560949e+14	7.129575e+04	23.110205	0.297675	0.397921	0.258265	0.171686	0.161543	0.466873
min	3.921784e+04	5.030230e+06	-1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	4.172614e+12	5.640286e+06	18.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	3.173184e+13	5.680573e+06	37.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	9.439172e+13	5.725524e+06	55.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
max	9.999816e+14	5.790484e+06	115.000000	1.000000	1.000000	1.000000	1.000000	4.000000	1.000000

***Figura 3**: Informações estatísticas do dataset.*

Os dados estão bem distribuídos entre as colunas, sendo muito fácil identificar a coluna pelo seu nome, existe uma grande quantidade de linhas no *dataset*, todo o conjunto de dados e seus características são muito bem definidas, sendo um ótimo *dataset* para a analise e realizar predições com *Machine Learning*.

Visualização Exploratória

O objetivo do projeto consiste em realizar uma analise nos dados do *dataset* e realizar uma predição utilizando *Machine Learning*, para identificar os paciente que irão comparecer a consulta médica. O gráfico visualizado na ***Figura 4***, representa os dados certos, representando se o paciente compareceu a consulta médica ou não, é possível visualizar que um total de 79.8% compareceram a consulta médica e 20.2% não compareceram.

A partir desta informação, é possível identificar que grande parte dos pacientes compareceram a consulta médica, apenas uma pequena faixa de pouco mais de 20%

não compareceram, o que indica que o algorítimo de *Machine Learning* terá uma boa distribuição dos dados, melhorando sua predição em acertar se o paciente irá comparecer a consulta médica ou não.

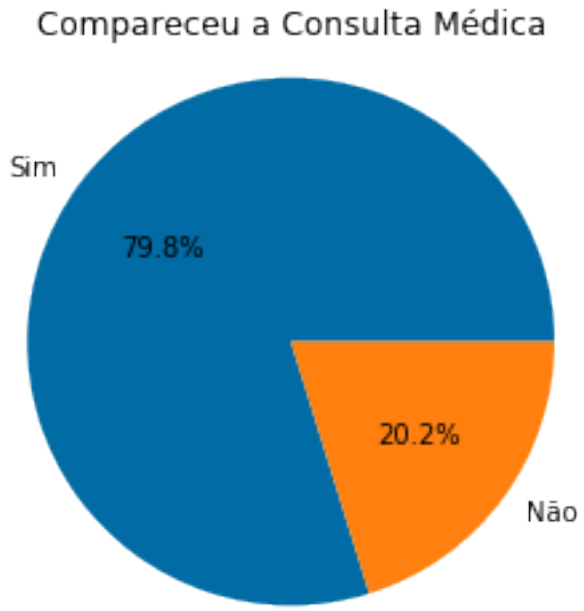

***Figura 4**: Gráfico Compareceu a Consulta Médica.*

Comparando homens e mulheres que marcaram consulta medicas, é obtido o resultado que pode ser visualizado na ***Figura 5***. É possível visualizar que grande parte das consultas médicas foram marcadas para mulheres, somando um total de 65%, contra 35% para as consultas médicas marcadas para os homens.

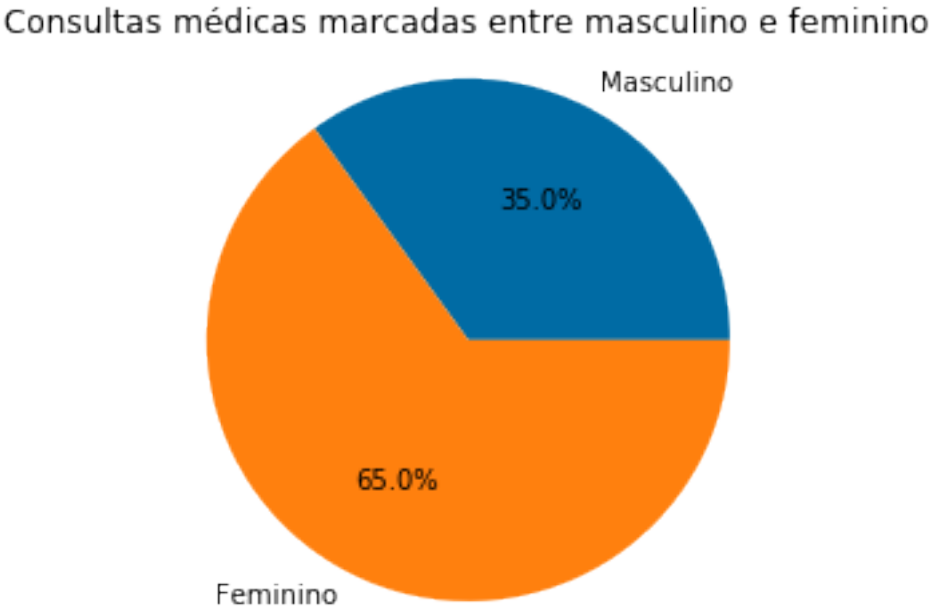

***Figura 5**: Gráfico com as consultar médicas marcadas para homens e mulheres.*

Comparando a quantidade de homens e mulheres que marcaram consultas médicas, e compareceram a elas, é possível visualizar este resultado na ***Figura 6***, homens compareceram a 30.962 consultas médicas e faltaram em 7.725, já as mulheres, compareceram a 57.246 e faltaram em 14.594.

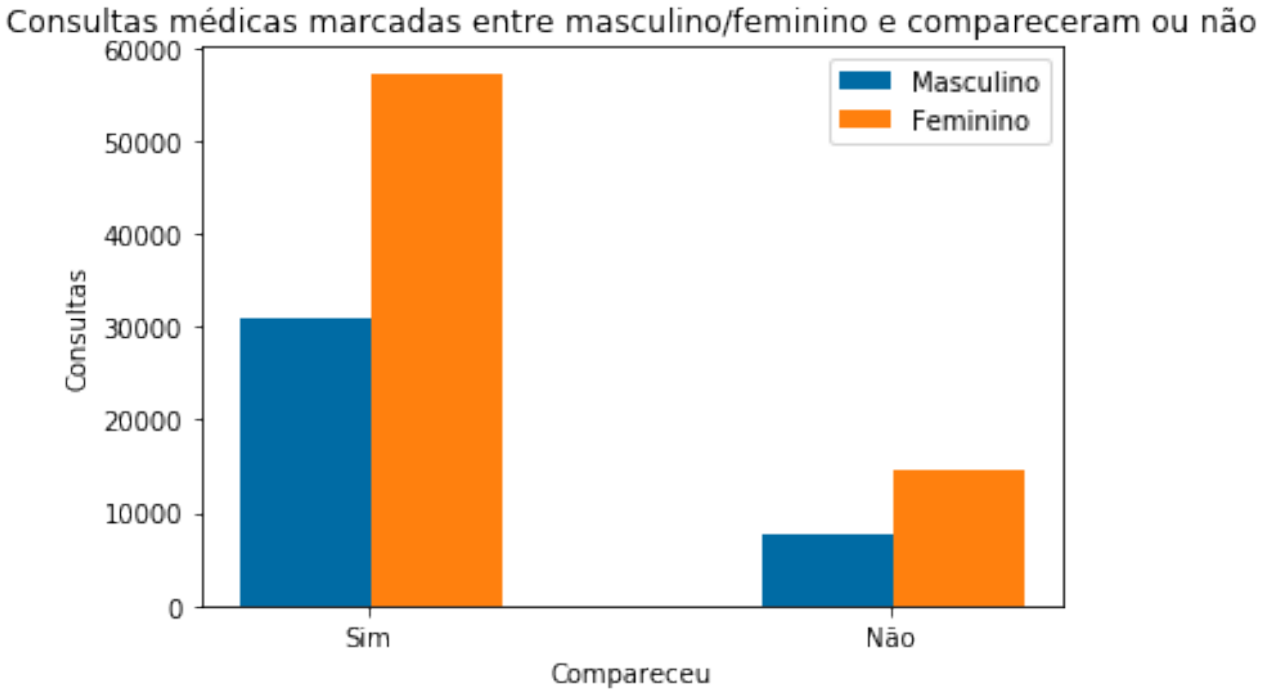

***Figura 6**: Gráfico comparando quantidade de homens e mulheres que compareceram a consultas médicas ou não.*

Com base nos dados, é possível verificar que as pessoas estão interessadas em ter uma boa saúde, marcando e comparecendo a consultas médicas, pois existe uma grande quantidade de dados que demonstra isso. Também é possível verificar que existe uma grande quantidade de pessoas que estão aderindo a tecnologia, ate para lembrar das consultas médicas, recebendo lembretes via SMS, ajudando a evitar o esquecimento dela, conforme é possível visualizar na **Figura 7**.

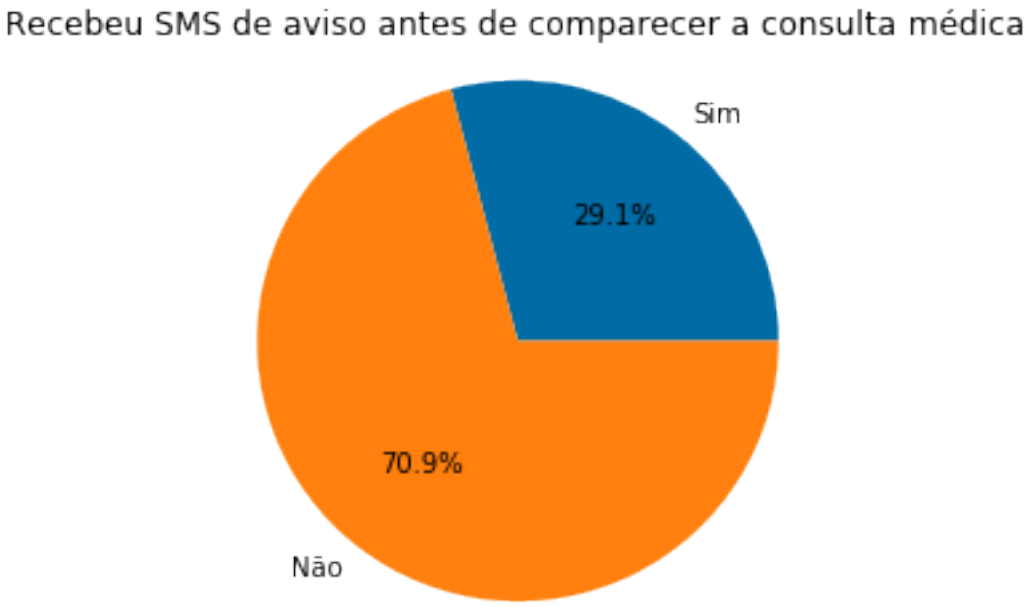

***Figura 7**: Gráfico comparando pacientes que receberam ao menos um SMS antes de comparecer a consulta médica.*

Algoritmos e Técnicas

As bibliotecas que serão usadas neste projeto são *Pandas*, *NumPy*, *Matplotlib* e *scikit-learn*. Todos eles foram desenvolvido para ser utilizados com a linguagem de programação *Python*, e são usados desde a analise dos dados, geração dos gráficos e *Machine Learning* para este projeto.

A biblioteca *Pandas* será utilizada para visualizar e manipular as estruturas de dados, e também para realizar analises, pois existe inúmeros métodos prontos para este proposito, como por exemplo, obter a média, desvio padrão, valores máximo e mínimo, entre muitas outros métodos para analise e estatística.

A biblioteca *NumPy* será utilizada para criar arrays e matrizes multidimensionais, de tamanho definido pelo usuário, que também inclui uma enorme quantidade de métodos para analise e estatística, que serão utilizadas neste projeto para esse propósito.

A biblioteca *Matplotlib* será utilizada para a geração dos gráficos, pois ela possui uma grande quantidade de tipos de gráficos, por exemplo, gráficos de barras, pizza, linhas

e muitos outros. Está biblioteca será utilizada neste projeto para este proposito, a geração dos gráficos.

A biblioteca *scikit-learn* é uma biblioteca de *Machine Learning*, uma das principais bibliotecas desta categoria para a linguagem de programação *Python*. Nela é possível realizar inúmeros tipos de *Machine Learning*, como aprendizado supervisionado, no qual será utilizado neste projeto, com diversos algorítimos desta categoria, todos inclusos na biblioteca *scikit-learn*, são eles, *GaussianNB*, *DecisionTreeClassifier*, *RandomForestClassifier*, *AdaBoostClassifier*, *GradientBoostingClassifier*, *SVC*, *LogisticRegression* e *SGDClassifier*, os principais algorítimos para essa técnica de *Machine Learning*. O *scikit-learn* inclui outros tipos de aprendizado, como aprendizado não supervisionado, técnicas de classificação, regressão, *cluster* e muitas outras.

A técnica de *Machine Learning*, aprendizado supervisionado, será a técnica utilizada neste projeto, sendo que a maquina aprende conforme os dados são disponibilizados a ela. A biblioteca *scikit-learn* irá analisar, realizar cálculos matemáticos e encontrar os padrões nos dados, utilizando o *dataset* como a fonte de dados, e a partir disto classificar o paciente, em se ele irá comparecer a consulta médica ou não.

O algorítimo *DecisionTreeClassifier* é baseado na técnica de *Decision Trees*. Ele é um suporte de decisão, utilizando diagramas em forma de uma arvore, onde os dados de origem estão no tronco da arvore, e com base em analises, usando estimativas e probabilidades de associação, são criados os galhos, que nos leva a resposta final da classificação do algorítimo. Vantagens deste algorítimo é sua simplicidade para entender e interpretar, é possível gerar uma imagem e visualizar o formato da arvore de decisão, arvores de decisão são rápidas e contem uma fácil configuração, funciona muito bem com vários tipos de dados e com grandes ou pequenas quantidades de dados. As desvantagens deste algorítimo é que arvores de decisão podem ser grandes e complexas, e não predizer muito bem, isso se chama *overfitting*, e variações nos dados podem afetar totalmente a arvore de decisão.

O algorítimo *GaussianNB* é baseado na técnica de *Naive Bayes*. Este algorítimo irá classificar os dados utilizando o Teorema de Bayes, que utiliza a formula desenvolvida pelo inglês Thomas Bayes. A formula descreve a probabilidade de algum evento, com base no conhecimento *a priori* pode estar relacionada ao evento. O teorema mostra como alterar as probabilidades com novas evidencias para obter a probabilidade *a posteriori*. Vantagens deste algorítimo é sua simples implementação, com poucas configurações e é consideravelmente muito rápido. As desvantagens são quando os dados tem muitos atributos, a classificação pode não ser muito boa e os atributos são independentes.

O algorítimo *LogisticRegression* é um algorítimo de classificação, ao contrario de que diz seu nome, referenciando como regressão. Este algorítimo utiliza a técnica *Logistic Regression*, também conhecida como regressão *logit*, classificação de máxima entropia *MaxEnt* ou classificador *log-linear*. Este algorítimo utiliza probabilidades que descrevem os possíveis resultados, e estes são modelados usando uma função

logística. As vantagens deste algorítimo é sua simplicidade e seu ótimo desempenho, funcionando muito bem com grandes e pequenas quantidade de dados. Uma desvantagem é que ele não pode ser utilizado quando os dados não são binários.

O algorítimo *SVC* é um algorítimo de classificação que utiliza o a técnica de *Support Vector Machines* (SVM), implementado o *Support Vector Classification* (SVC) para casos de *kernel* linear. O *SVM* encontra uma separação nos dados disponíveis, está separação maximiza a distancia entre os dados, para encontrar a melhor forma para classificar os dados da melhor maneira possível. As vantagens deste algorítimo é ser muito eficiente quando o número de dimensões é maior que que o número de amostras, é eficiente em termos de desempenho e processamento computacional, é versátil, existem diferentes funções no *kernel* que podem ser especificas em funções de decisão, e é possível utilizar *kernels* personalizados. Desvantagens deste algorítimo é que ele não fornece estimativas de probabilidades, pois são calculadas usando validação cruzada e se o numero de recursos for muito maior que o numero de amostra, muitos dos ajuste no *kelnel* irá prejudicar a classificação dos valores.

O algorítimo *SGDClassifier* utiliza a técnica *Stochastic Gradient Descent* (SGD), que é uma abordagem simples, mas muito eficiente para *Machine Learning* em classificação, principalmente para grandes volumes de dados. Muito utilizado em classificação de texto e em processamento de linguagem natural. Quando os dados são em menor quantidade, ele consegue escalar muito bem, melhorando a classificação. Vantagens deste algorítimo incluem ser muito eficiente em grande quantidade de dados e ser muito fácil a sua implementação. Uma desvantagem é que é difícil dimensionar os seus recursos e para refinar seu algorítimo, reque uma grandes quantidades de parâmetros.

O algorítimo *RandomForestClassifier* utiliza a técnica de *Decision Trees*, mas utilizando varias combinações, introduzindo um conceito de aleatoriedade, cada uma com diferentes configurações, e a classificação é definida com a média dos classificadores individuais. Uma vantagem é que utiliza a técnica de *Decision Trees* com inúmeras configurações. A desvantagem é que quando a dados com muito variações, ele pode não trabalhar muito bem.

O algorítimo *AdaBoostClassifier* utiliza a técnica *AdaBoost*, que tem como objetivo ajustar modelos fracos em versões repetidamente modificadas dos dados. As previsões de todos eles são combinadas através de um voto majoritário para produzir a previsão final. As modificações de dados utiliza o conceito de aplicar pesos em cada uma das amostras de treinamento afim de encontrar a melhor forma para trabalhar com os dados. As vantagens é sua poderosa forma de classificar os dados e trabalhar com vários tipos de dados. Uma desvantagem é que pequenas variações nos dados pode prejudicar sua classificação.

O algorítimo *GradientBoostingClassifier* utiliza a técnica *Gradient Boosted Regression Trees* (GBRT), que generaliza o estimulo para funções de perdas arbitrarias e diferenciais, deixando um procedimento pronto que pode ser usado em classificação e

regressão nos dados. As vantagens é a possibilidade de trabalhar com dados de vários tipos e tem uma ótima performance. As desvantagens dele é sua difícil escalabilidade e sua dificuldade em fazer paralelização.

Será utilizado a técnica de *Train/Test Split*, que consiste em dividir o *dataset* em duas partes, os dados de treino e os dados de teste. Esta divisão será definida em 80% dados de treino e 20% dados de teste para este projeto.

O algorítimo *GridSearchCV*, disponível na biblioteca do *scikit-learn*, será utilizado para realizar o refinamento dos modelos. Durante o desenvolvimento será realizado teste com seis algorítimos de *Machine Learning*, e os três melhores receberam o refinamento. Este refinamento é feito testando inúmeros parâmetros entre si, para decidir quais são os melhores parâmetros e o que retorna a melhor pontuação para este *dataset*.

Benchmark

No *dataset* existe uma coluna chamada *No-show*, que contem o resultado certo, o paciente compareceu a consulta médica, ou faltou a ela, e a partir disto, será realizado uma predição, utilizando *Machine Learning*, para classificar este valor em compareceu a consulta médica ou não.

Após as execuções dos algorítimos de *Machine Learning*, utilizando dados de treino, será obtido a pontuação do algorítimo, utilizando os dados de teste, para verificar se o resultado obtido foi acima dos 0.70 em *accuracy, fbeta e roc/auc*, sendo esse o resultado esperado para essa predição, o objetivo de nosso projeto.

A pontuação esperada para o projeto foi definida em 0.70 com base em analises realizadas nos *kernels* deste *dataset* no *Kaggle*. Como este *dataset* não esta em nenhuma competição, e também não tem nenhum modelo de referencia para ele que seria possível se basear, foi definido este valor para o projeto.

Será utilizado a técnica de *Train/Test Split* para dividir os dados em treino e teste, e *accuracy, fbeta e roc/auc* para validar essa pontuação, utilizando a biblioteca de *Machine Learning scikit-learn*, que contem todos esses métodos, desde dividir os dados e apurar a pontuação de *accuracy*, *fbeta e roc/auc*.

III. Metodologia

Pré-processamento de Dados

O *dataset* esta muito bem organizado, mas para executar os algorítimos de *Machine Learning*, será feita algumas alterações, renomeado algumas colunas, para ficar mais fácil seu entendimento. A coluna *PatientId* será renomeada para *Patient_ID*, a coluna *AppointmentID* será renomeada para *Appointment_ID*, a coluna *ScheduledDay* será renomeada para *Scheduled_Day*, a coluna *AppointmentDay* será renomeada para *Appointment_Day*, a coluna *SMS_received* será renomeada para *SMS_Received* e a coluna *No-show* será renomeada para *Show*.

As coluna de data, agora com os nomes de *Scheduled_Day* e *Appointment_Day* estão como se fosse apenas um valor de texto, este valor será alterado para que o *Python* reconheça este valor como uma data, para isso será aplicado ao seu tipo de valor para *datetime64[ns]*, utilizando o método *astype* presente na biblioteca Pandas.

Será criada uma nova coluna neste *dataset*, utilizando os valores das colunas *Scheduled_Day* e *Appointment_Day* e será chamada de *Difference_Date_Medical_Consultation*, que será a quantidade de dias entre o dia que a consulta médica foi marcada e o dia que a consulta médica será realizada, e este valor será dividido por cinco, retornando um numero inteiro, diminuindo a quantidade de dias, para não haver muitos dados distorcidos, agrupando os dados a cada cinco dias. Isto pode ajudar e muito os algorítimos de *Machine Learning*, pois quanto mais distante a consulta médica foi marcada comparada ao dia que ela será realizada, mais chance de o paciente faltar a ela.

Como é muito difícil algorítimos de *Machine Learning* utilizarem data para realizar predições, a coluna *Scheduled_Day* e a coluna *Appointment_Day* será transformada para receber o valor do dia da semana, como segunda-feira por exemplo, estes valores vão estar em inglês. Isso pode ajudar muito a classificação do valor, pois é mais fácil um algorítimo acertar que o paciente irá faltar a uma consulta médica numa segunda-feira, do que acertar que ele vai faltar no dia 03/09/2018, que é uma segunda-feira.

A coluna *Gender*, que contem o sexo do paciente terá os valores alterados para quando o valor for M (masculino), receber o valor de 1, e quando for igual a F (feminino), receber o valor de 0.

A coluna de *Age*, que representa a idade do paciente, também irá utilizar a divisão por cinco, para não haver muitos dados distorcidos, e sim os dados agrupados a cada cinco anos.

A antiga coluna *No-show*, que agora tem o nome de *Show* terá mudanças em seu valor, o atual valor dela é *Yes* e *No. E*sse texto será alterado para valores numéricos,

sendo *True* quando o valor for igual a *No*, e *False* quando o valor for igual a *Yes*. Esta alteração pode confundir, pois *True* esta recebendo o valor de *No*, mas neste *dataset*, a coluna original se chama *No-show*, sendo que se o valor for *Yes*, significa que o paciente não compareceu a consulta e quando for *No*, significa que o paciente compareceu a consulta.

Duas colunas serão removidas para realizar os testes de *Machine Learning*, pois elas podem dificultar os algorítimos a classificar o valor corretamente, pois estas colunas contem valores irrelevantes para a classificação. A coluna *Patient_ID* contem o ID do paciente, não ajudara em nada em nossa classificação. A coluna *Appointment_ID* contem o ID da consulta marcada, novamente, ID não irá ajudar em nada durante a classificação, mas pode atrapalhar.

Todos os dados que serão classificados com os algorítimos de *Machine Learning* estão com valores numéricos, possibilitando testar inúmeros algorítimos de *Machine Learning* sobre estes dados.

Implementação

Antes de inciar os testes com os dados nos algorítimos de *Machine Learning*, foi criado um novo *dataset* a partir do *dataset* original. Este novo *dataset* tem os dados divididos. Como o *dataset* original tem a maioria dos dados de pacientes que compareceram as consultas médicas, este novo *dataset* tem uma divisão em 50%, são pegados todos os pacientes que não compareceram a consulta médica, e será pegado a mesma quantidade de pacientes que compareceram a consulta médica, criando um novo *dataset*, conforme o Código ***1***. Este novo *dataset* tem 44.638 linhas.

```
ml_data_not_show = data[data['Show'] == False]
ml_data_show = data[data['Show'] == True].head(len(ml_data_not_show))
ml_data = ml_data_not_show.append(ml_data_show)
```

Código 1*. Criação do novo dataset.*

Para a predição nos dados com *Machine Learning* foi criado as variáveis *X* e *y*, sendo a variável *X* com todas as colunas do *dataset*, menos a coluna de *Show*, que ficou armazenado na variável *y*. Apos isso, foi utilizando o método *train_test_split* do *scikit-learn*, para dividir os dados em treino e teste, conforme o ***Código 2***.

```
y = ml_data.Show
X = ml_data.drop('Show', axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=8)
```

Código 2*. Criação e divisão dos dados de treino e teste para X e y.*

A divisão dos dados ficou em 35.710 linhas do *dataset* para os dados de treino, enquanto 8.928 para os dados de teste.

O primeiro algorítimo de *Machine Learning* testado foi usando o método de *Naive Bayes*, utilizando o algorítimo *GaussianNB* do *scikit-learn*. Em seguida foi realizado o treinamento com os dados de treino, utilizando o método *fit*, e por ultimo, obtido a pontuação de *accuracy*, *fbeta* e *roc/auc* através dos dados de teste. O resultado ficou em **0.5267** para a *accuracy*, **0.5666** para o *fbeta* e **0.5277** para *roc/auc*. É possível visualizar o código no item ***Código 3***.

```
gaussian_nb = GaussianNB()
gaussian_nb.fit(X_train, y_train)

pred = gaussian_nb.predict(X_test)

print(accuracy_score(y_test, pred, normalize=True))
print(fbeta_score(y_test, pred, beta=0.5))
print(roc_auc_score(y_test, pred))
```

***Código 3**. Criação, treino e resultado do teste de Machine Learning utilizando o algorítimo GaussianNB.*

O segundo algorítimo de *Machine Learning* testado foi usando o método de *Stochastic Gradient Descent*, utilizando o algorítimo *SGDClassifier* do *scikit-learn*. O único parâmetro passado durante a criação do algorítimo foi o valor de *random_state*, definido com o valor de 8. Em seguida foi realizado o treinamento com os dados de treino, utilizando o método *fit*, e por ultimo, obtido a pontuação de *accuracy*, *fbeta* e *roc/auc* através dos dados de teste. O resultado ficou em **0.6658** para a *accuracy*, **0.6634** para o *fbeta* e **0.6659** para *roc/auc*. É possível visualizar o código no item ***Código 4***.

```
sgd_classifier = SGDClassifier(random_state=8)
sgd_classifier.fit(X_train, y_train)

pred = sgd_classifier.predict(X_test)

print(accuracy_score(y_test, pred, normalize=True))
print(fbeta_score(y_test, pred, beta=0.5))
print(roc_auc_score(y_test, pred))
```

***Código 4**. Criação, treino e resultado do teste de Machine Learning utilizando o algorítimo SGDClassifier.*

O terceiro algorítimo de *Machine Learning* testado foi usando o método de *Decision Trees*, utilizando o algorítimo *DecisionTreeClassifier* do *scikit-learn*. O único parâmetro passado durante a criação do algorítimo foi o valor de *random_state*, definido com o valor de 8. Em seguida foi realizado o treinamento com os dados de treino, utilizando o método *fit*, e por ultimo, obtido a pontuação de *accuracy*, *fbeta* e *roc/auc* através dos dados de teste. O resultado ficou em **0.6805** para a *accuracy*, **0.6806** para o *fbeta* e **0.6805** para *roc/auc*. É possível visualizar o código no item ***Código 5***.

```
decision_tree_classifier = DecisionTreeClassifier(random_state=8)
decision_tree_classifier.fit(X_train, y_train)

pred = decision_tree_classifier.predict(X_test)

print(accuracy_score(y_test, pred, normalize=True))
print(fbeta_score(y_test, pred, beta=0.5))
print(roc_auc_score(y_test, pred))
```

***Código 5**. Criação, treino e resultado do teste de Machine Learning utilizando o algorítimo DecisionTreeClassifier.*

O quarto algorítimo de *Machine Learning* testado foi usando o método de *Forests of Randomized Trees*, utilizando o algorítimo *RandomForestClassifier* do *scikit-learn*. O único parâmetro passado durante a criação do algorítimo foi o valor de *random_state*, definido com o valor de 8. Em seguida foi realizado o treinamento com os dados de treino, utilizando o método *fit*, e por ultimo, obtido a pontuação de *accuracy*, *fbeta* e *roc/auc* através dos dados de teste. O resultado ficou em **0.6967** para a *accuracy*, **0.7026** para o *fbeta* e **0.6966** para *roc/auc*. É possível visualizar o código no item ***Código 6***.

```
random_forest_classifier = RandomForestClassifier(random_state=8)
random_forest_classifier.fit(X_train, y_train)

pred = random_forest_classifier.predict(X_test)

print(accuracy_score(y_test, pred, normalize=True))
print(fbeta_score(y_test, pred, beta=0.5))
print(roc_auc_score(y_test, pred))
```

***Código 6**. Criação, treino e resultado do teste de Machine Learning utilizando o algorítimo RandomForestClassifier.*

O quinto algorítimo de *Machine Learning* testado foi usando o método de *AdaBoost*, utilizando o algorítimo *AdaBoostClassifier* do *scikit-learn*. O único parâmetro passado durante a criação do algorítimo foi o valor de *random_state*, definido com o valor de 8. Em seguida foi realizado o treinamento com os dados de treino, utilizando o método *fit*, e por ultimo, obtido a pontuação de *accuracy*, *fbeta* e *roc/auc* através dos dados de teste. O resultado ficou em **0.6822** para a *accuracy*, **0.6822** para o *fbeta* e **0.6822** para *roc/auc*. É possível visualizar o código no item ***Código 7***.

```
ada_boost_classifier = AdaBoostClassifier(random_state=8)
ada_boost_classifier.fit(X_train, y_train)

pred = ada_boost_classifier.predict(X_test)

print(accuracy_score(y_test, pred, normalize=True))
```

```
print(fbeta_score(y_test, pred, beta=0.5))
print(roc_auc_score(y_test, pred))
```

***Código 7**. Criação, treino e resultado do teste de Machine Learning utilizando o algorítimo AdaBoostClassifier.*

O sexto algorítimo de *Machine Learning* testado foi usando o método de *Gradient Tree Boosting*, utilizando o algorítimo *GradientBoostingClassifier* do *scikit-learn*. O único parâmetro passado durante a criação do algorítimo foi o valor de *random_state*, definido com o valor de 8. Em seguida foi realizado o treinamento com os dados de treino, utilizando o método *fit*, e por ultimo, obtido a pontuação de *accuracy*, *fbeta* e *roc/auc* através dos dados de teste. O resultado ficou em **0.6798** para a *accuracy*, **0.6821** para o *fbeta* e **0.6798** para *roc/auc*. É possível visualizar o código no item ***Código 8***.

```
gradient_boosting_classifier =
GradientBoostingClassifier(random_state=8)
gradient_boosting_classifier.fit(X_train, y_train)

print(accuracy_score(y_test, pred, normalize=True))
print(fbeta_score(y_test, pred, beta=0.5))
print(roc_auc_score(y_test, pred))
```

***Código 8**. Criação, treino e resultado do teste de Machine Learning utilizando o algorítimo GradientBoostingClassifier.*

O sétimo algorítimo de *Machine Learning* testado foi usando o método de *Support Vector Machines*, utilizando o algorítimo *SVC* do *scikit-learn*. Os parâmetros passados durante a criação do algorítimo foram o valor de *random_state*, definido com o valor de 8 e *max_iter* recebendo o valor de 1000. Em seguida foi realizado o treinamento com os dados de treino, utilizando o método *fit*, e por ultimo, obtido a pontuação de *accuracy*, *fbeta* e *roc/auc* através dos dados de teste. O resultado ficou em **0.5300** para a *accuracy*, **0.5634** para o *fbeta* e **0.5307** para *roc/auc*. É possível visualizar o código no item ***Código 9***.

```
svc = SVC(random_state=8, max_iter=1000)
svc.fit(X_train, y_train)

pred = svc.predict(X_test)

print(accuracy_score(y_test, pred, normalize=True))
print(fbeta_score(y_test, pred, beta=0.5))
print(roc_auc_score(y_test, pred))
```

***Código 9**. Criação, treino e resultado do teste de Machine Learning utilizando o algorítimo SVC.*

O oitavo e ultimo algorítimo de *Machine Learning* testado foi usando o método de *Logistic Regression*, utilizando o algorítimo *LogisticRegression* do *scikit-learn*. O único

parâmetro passado durante a criação do algorítimo foi o valor de *random_state*, definido com o valor de 8. Em seguida foi realizado o treinamento com os dados de treino, utilizando o método *fit*, e por ultimo, obtido a pontuação de *accuracy*, *fbeta* e *roc/auc* através dos dados de teste. O resultado ficou em **0.6859** para a *accuracy*, **0.6847** para o *fbeta* e **0.6859** para *roc/auc*. É possível visualizar o código no item ***Código 10***.

```
logistic_regression = LogisticRegression(random_state=8)
logistic_regression.fit(X_train, y_train)

pred = logistic_regression.predict(X_test)

print(accuracy_score(y_test, pred, normalize=True))
print(fbeta_score(y_test, pred, beta=0.5))
print(roc_auc_score(y_test, pred))
```

Código 10*. Criação, treino e resultado do teste de Machine Learning utilizando o algorítimo LogisticRegression.*

Analisando os oito algorítimos de *Machine Learning* utilizados para esta predição, é possível verificar que os quatro melhores algorítimos são *DecisionTreeClassifier*, *RandomForestClassifier*, *AdaBoostClassifier* e *LogisticRegression*, todos eles conseguiram um ótimo resultado de *accuracy, fbeta e* roc/auc, quase atingindo os **0.70**, que era o nosso objetivo. Os outros quatro algorítimos também tiveram uma ótimo pontuação para o *accuracy, fbeta e roc/auc*, mas não foram tão bons nesse *dataset*, são *GaussianNB*, *GradientBoostingClassifier*, *SVC* e *SGDClassifier*. O Refinamento dos algorítimos será realizada apenas nos quatro melhores algorítimos, que esta disponível no próximo item deste documento.

Refinamento

A refinamento dos modelos foi feito utilizando o algorítimo *GridSearchCV* disponível no *scikit-learn*, a partir dele foi passado qual algorítimo de *Machine Learning* ele irá utilizar, e seus parâmetros, no qual ele vai testar, e retornar quais são o melhores parâmetros, para por ultimo, calcular sua pontuação de *accuracy, fbeta e* roc/auc, para então analisar se a pontuação melhorou, piorou, ou continuou na mesma pontuação de que quando não foi utilizado o refinamento.

O primeiro algorítimo que foi refinado utilizando o algorítimo *GridSearchCV* foi o *DecisionTreeClassifier*, todos os parâmetros e códigos utilizados para testar o refinamento estão descritos no item ***Código 11***. Para utilizar o algorítimo *GridSearchCV*, primeiro foi criado o algorítimo de *Machine Learning DecisionTreeClassifier*, e em seguida foi criado os parâmetros para testar durante o refinamento. Depois foi criado o algorítimo *GridSearchCV*, passando como parâmetros o algorítimo *DecisionTreeClassifier* e os parâmetros de refinamento, e por fim treinado o modelo, refinando para identificar os melhores parâmetros. Ao fim, temos o resultado, o *accuracy* ficou com o valor de **0.6851**, o *fbeta* ficou com o valor de **0.6857**

e o *roc/auc* ficou com o valor de **0.6851**. O *accuracy* teve uma melhora de **0.0045**, o *fbeta* teve uma melhora de **0.0051** e o *roc/auc* teve uma melhora de **0.0045**, utilizando o algorítimo *GridSearchCV* para refinar o modelo.

```
refined_decision_tree_classifier = DecisionTreeClassifier()
parameters = {'criterion': ['gini', 'entropy'], 'splitter': ['best',
'random'], 'min_samples_split': [2, 3, 4], 'min_samples_leaf': [1, 2,
3], 'min_weight_fraction_leaf': [0, 0.25, 0.5],
'min_impurity_decrease': [0, 0.25, 0.5], 'presort': [True, False],
'random_state': [8]}
grid_search_cv = GridSearchCV(refined_decision_tree_classifier,
parameters)
grid_search_cv.fit(X_train, y_train)
refined_decision_tree_classifier = grid_search_cv.best_estimator_

pred = refined_decision_tree_classifier.predict(X_test)

print(accuracy_score(y_test, pred, normalize=True))
print(fbeta_score(y_test, pred, beta=0.5))
print(roc_auc_score(y_test, pred))
```

***Código 11**. Criação, treino, refinamento e resultado de Machine Learning utilizando o algorítimo DecisionTreeClassifier.*

O segundo algorítimo que foi refinado utilizando o algorítimo *GridSearchCV* foi o *RandomForestClassifier*, todos os parâmetros e códigos utilizados para testar o refinamento estão descritos no item ***Código 12***. Para utilizar o algorítimo *GridSearchCV*, primeiro foi criado o algorítimo de *Machine Learning RandomForestClassifier*, e em seguida foi criado os parâmetros para testar durante o refinamento. Depois foi criado o algorítimo *GridSearchCV*, passando como parâmetros o algorítimo *RandomForestClassifier* e os parâmetros de refinamento, e por fim treinado o modelo, refinando para identificar os melhores parâmetros. Ao fim, temos o resultado, o *accuracy* ficou com o valor de **0.7090**, o *fbeta* ficou com o valor de **0.7118** e o *roc/auc* ficou com o valor de **0.7089**. O *accuracy* teve uma melhora de **0.0122**, o *fbeta* teve uma melhora de **0.0091** e o *roc/auc* teve uma melhora de **0.0122**, utilizando o algorítimo *GridSearchCV* para refinar o modelo.

```
refined_random_forest_classifier = RandomForestClassifier()
parameters = {'criterion': ['gini', 'entropy'], 'min_samples_leaf':
[1, 2, 3], 'oob_score': [True, False], 'random_state': [8]}
grid_search_cv = GridSearchCV(refined_random_forest_classifier,
parameters)
grid_search_cv.fit(X_train, y_train)
refined_random_forest_classifier = grid_search_cv.best_estimator_

pred = refined_random_forest_classifier.predict(X_test)

print(accuracy_score(y_test, pred, normalize=True))
```

```
print(fbeta_score(y_test, pred, beta=0.5))
print(roc_auc_score(y_test, pred))
```

***Código 12**. Criação, treino, refinamento e resultado do Machine Learning utilizando o algorítimo RandomForestClassifier.*

O terceiro algorítimo que foi refinado utilizando o algorítimo *GridSearchCV* foi o *AdaBoostClassifier*, todos os parâmetros e códigos utilizados para testar o refinamento estão descritos no item ***Código 13***. Para utilizar o algorítimo *GridSearchCV*, primeiro foi criado o algorítimo de *Machine Learning AdaBoostClassifier*, e em seguida foi criado os parâmetros para testar durante o refinamento. Depois foi criado o algorítimo *GridSearchCV*, passando como parâmetros o algorítimo *AdaBoostClassifier* e os parâmetros de refinamento, e por fim treinado o modelo, refinando para identificar os melhores parâmetros. Ao fim, temos o resultado, o *accuracy* ficou com o valor de **0.6839**, o *fbeta* ficou com o valor de **0.6828** e o *roc/auc* ficou com o valor de **0.6839**. O *accuracy* teve uma melhora de **0.0016**, o *fbeta* teve uma melhora de **0.0005** e o *roc/auc* teve uma melhora de **0.0017**, utilizando o algorítimo *GridSearchCV* para refinar o modelo.

```
refined_ada_boost_classifier = AdaBoostClassifier()
parameters = {'n_estimators': [50, 100, 150], 'learning_rate': [1.0,
2.0, 3.0], 'algorithm': ['SAMME', 'SAMME.R'], 'random_state': [8]}
grid_search_cv = GridSearchCV(refined_ada_boost_classifier,
parameters)
grid_search_cv.fit(X_train, y_train)
refined_ada_boost_classifier = grid_search_cv.best_estimator_

pred = refined_ada_boost_classifier.predict(X_test)

print(accuracy_score(y_test, pred, normalize=True))
print(fbeta_score(y_test, pred, beta=0.5))
print(roc_auc_score(y_test, pred))
```

***Código 13**. Criação, treino, refinamento e resultado do Machine Learning utilizando o algorítimo AdaBoostClassifier.*

O quarto e ultimo algorítimo que foi refinado utilizando o algorítimo *GridSearchCV* foi o *LogisticRegression*, todos os parâmetros e códigos utilizados para testar o refinamento estão descritos no item ***Código 14***. Para utilizar o algorítimo *GridSearchCV*, primeiro foi criado o algorítimo de *Machine Learning LogisticRegression*, e em seguida foi criado os parâmetros para testar durante o refinamento. Depois foi criado o algorítimo *GridSearchCV*, passando como parâmetros o algorítimo *LogisticRegression* e os parâmetros de refinamento, e por fim treinado o modelo, refinando para identificar os melhores parâmetros. Ao fim, temos o resultado, o *accuracy* ficou com o valor de **0.6857**, o *fbeta* ficou com o valor de **0.6844** e o *roc/auc* ficou com o valor de **0.6857**. O *accuracy* teve uma perda de **0.0002**, o *fbeta* teve uma perda de **0.0002** e o *roc/auc* teve uma perda de **0.0002**, utilizando o algorítimo *GridSearchCV* para refinar o modelo.

```
refined_logistic_regression = LogisticRegression()
parameters = {'C': [0.1, 1, 10], 'fit_intercept': [True, False],
'intercept_scaling': [1, 10, 100], 'random_state': [8], 'solver':
['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']}
grid_search_cv = GridSearchCV(refined_logistic_regression,
parameters)
grid_search_cv.fit(X_train, y_train)
refined_logistic_regression = grid_search_cv.best_estimator_

pred = refined_logistic_regression.predict(X_test)

print(accuracy_score(y_test, pred, normalize=True))
print(fbeta_score(y_test, pred, beta=0.5))
print(roc_auc_score(y_test, pred))
```

***Código 14**. Criação, treino, refinamento e resultado do Machine Learning utilizando o algorítimo LogisticRegression.*

IV. Resultados

Modelo de Avaliação e Validação

O modelo final utiliza o algorítimo *RandomForestClassifier* para realizar suas predições, este modelo conseguiu atingir o objetivo do projeto, que era alcançar no mínimo 0.70 de *accuracy,* 0.70 de *fbeta* e 0.70 de *roc/auc*, utilizando os dados de treino para realizar este calculo.

Os demais algorítimos *GaussianNB*, *DecisionTreeClassifier*, *AdaBoostClassifier*, *GradientBoostingClassifier*, *SVC*, *LogisticRegression* e *SGDClassifier* não obtiveram o mínimo de 0.70 em *accuracy*, *fbeta* e *roc/auc*.

Para realizar o calculo de *accuracy, fbeta e roc/auc* é utilizado a biblioteca *scikit-learn*, que contem esses cálculos já prontos, apenas precisando chamar seu método e passando os parâmetros na programação.

Os parâmetros utilizados para conseguir essa pontuação, utilizando o algorítimo de *Machine Learning RandomForestClassifier*, estão os representados no item ***Código 15***. Esses parâmetros foram definidos através do refinamento, utilizando o algorítimo *GridSearchCV*, disponível na biblioteca *scikit-learn*, e pode ser concluído que são parâmetros apropriados, pois conseguiram cumprir o objetivo do projeto.

```
{'bootstrap': True, 'class_weight': None, 'criterion': 'entropy',
'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None,
'min_impurity_decrease': 0.0, 'min_impurity_split': None,
'min_samples_leaf': 2, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'n_estimators': 10, 'n_jobs': 1,
'oob_score': True, 'random_state': 8, 'verbose': 0, 'warm_start':
False}
```

Código 15*. Parâmetros do melhor modelo do projeto, utilizando o algorítimo RandomForestClassifier.*

O modelo foi testado utilizando os dados de treino, que representa 20% de todo o novo *dataset*, um total de 8.928 linhas, uma boa quantidade de dados para realizar esta tarefa, e conseguiu o resultado de **0.7090** para o *accuracy*, **0.7118** para o *fbeta* e **0.7089** para *roc/auc*.

O modelo também foi testado com todos os dados do *dataset*, que representa um total de 110.527 linhas, e obtive o seguinte resultado, **0.6288** para o *accuracy,* **0.8273** para o *fbeta* e **0.6972** para o *roc/auc*.

Com base nesses resultados, pode se concluir que o modelo é confiável e cumpre o objetivo do projeto, que era ter no mínimo 0.70 de *accuracy,* 0.70 de *fbeta* e 0.70 de *roc/auc*, utilizando os dados de treino para realizar este calculo.

Justificativa

O modelo final utilizando o algorítimo *RandomForestClassifier* e todos os outros modelos atingiram o objetivo, que era ter no mínimo 0.70 em *accuracy, fbeta e roc/auc*, utilizando os dados de treino para realizar esse calculo. O modelo final, descrito no tópico acima é o melhor de todos os outros modelos utilizado nesse projeto.

Foram testados oito algorítimos de *Machine Learning*, todos disponíveis na biblioteca *scikit-learn*, nesse tópico será apenas descrito sete, pois o oitavo, que é o melhor modelo, foi descrito no tópico acima.

O calculo do *accuracy, fbeta* e *roc/auc* foram feitos através da biblioteca *scikit-learn*, utilizando os métodos da biblioteca. Esse calculo é realizado após o modelo já estiver treinado, utilizando os dados de treino, que representa 20% de todas as linhas do novo dataset, um total de 8.928 linhas.

O algorítimo *GaussianNB* teve seu modelo treinado e conseguiu este resultado, **0.5267** para *accuracy*, **0.5666** para *fbeta* e **0.5277** para *roc/auc*, e os parâmetros utilizados por este algorítimo são os que estão descrito no item ***Código 16***.

```
{'priors': None}
```

Código 16*. Parâmetros do modelo treinado utilizando o algorítimo GaussianNB.*

O algorítimo *SGDClassifier* teve seu modelo treinado e conseguiu este resultado, **0.6658** para *accuracy*, **0.6634** para *fbeta* e **0.6659** para *roc/auc*, e os parâmetros utilizados por este algorítimo são os que estão descrito no item **Código 17**.

```
{'alpha': 0.0001, 'average': False, 'class_weight': None, 'epsilon':
0.1, 'eta0': 0.0, 'fit_intercept': True, 'l1_ratio': 0.15,
'learning_rate': 'optimal', 'loss': 'hinge', 'max_iter': None,
'n_iter': None, 'n_jobs': 1, 'penalty': 'l2', 'power_t': 0.5,
'random_state': 8, 'shuffle': True, 'tol': None, 'verbose': 0,
'warm_start': False}
```

Código 17*. Parâmetros do modelo treinado utilizando o algorítimo SGDClassifier.*

O algorítimo *GradientBoostingClassifier* teve seu modelo treinado e conseguiu este resultado, **0.6798** para *accuracy*, **0.6821** para *fbeta* e **0.6798** para *roc/auc*, e os parâmetros utilizados por este algorítimo são os que estão descrito no item ***Código 18***.

```
{'criterion': 'friedman_mse', 'init': None, 'learning_rate': 0.1,
'loss': 'deviance', 'max_depth': 3, 'max_features': None,
'max_leaf_nodes': None, 'min_impurity_decrease': 0.0,
'min_impurity_split': None, 'min_samples_leaf': 1,
'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_estimators': 100, 'presort': 'auto', 'random_state': 8,
```

```
'subsample': 1.0, 'verbose': 0, 'warm_start': False}
```

***Código 18**. Parâmetros do modelo treinado utilizando o algorítimo GradientBoostingClassifier.*

O algorítimo *SVC* teve seu modelo treinado e conseguiu este resultado, **0.5300** para *accuracy*, **0.5634** para *fbeta* e **0.5307** para *roc/auc*, e os parâmetros utilizados por este algorítimo são os que estão descrito no item ***Código 19***.

```
{'C': 1.0, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0,
'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'auto',
'kernel': 'rbf', 'max_iter': 1000, 'probability': False,
'random_state': 8, 'shrinking': True, 'tol': 0.001, 'verbose': False}
```

***Código 19**. Parâmetros do modelo treinado utilizando o algorítimo SVC.*

O algorítimo *DecisionTreeClassifier* teve seu modelo treinado e conseguiu este resultado, **0.6851** para *accuracy*, **0.6857** para *fbeta* e **0.6851** para *roc/auc*, e os parâmetros utilizados por este algorítimo são os que estão descrito no item ***Código 20***, e eles foram definidos através o algorítimo de refinamento utilizado nesse projeto chamado *GridSearchCV*.

```
{'class_weight': None, 'criterion': 'gini', 'max_depth': None,
'max_features': None, 'max_leaf_nodes': None,
'min_impurity_decrease': 0, 'min_impurity_split': None,
'min_samples_leaf': 3, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0, 'presort': True, 'random_state': 8,
'splitter': 'random'}
```

***Código 20**. Parâmetros do modelo treinado utilizando o algorítimo DecisionTreeClassifier.*

O algorítimo *AdaBoostClassifier* teve seu modelo treinado e conseguiu este resultado, **0.6839** para *accuracy*, **0.6828** para *fbeta* e **0.6839** para *roc/auc*, e os parâmetros utilizados por este algorítimo são os que estão descrito no item ***Código 21***, e eles foram definidos através o algorítimo de refinamento utilizado nesse projeto chamado *GridSearchCV*.

```
{'algorithm': 'SAMME.R', 'base_estimator': None, 'learning_rate':
1.0, 'n_estimators': 100, 'random_state': 8}
```

***Código 21**. Parâmetros do modelo treinado utilizando o algorítimo AdaBoostClassifier.*

O algorítimo *LogisticRegression* teve seu modelo treinado e conseguiu este resultado, **0.6857** para *accuracy*, **0.6844** para *fbeta* e **0.6857** para *roc/auc*, e os parâmetros utilizados por este algorítimo são os que estão descrito no item ***Código 22***, e eles foram definidos através o algorítimo de refinamento utilizado nesse projeto chamado *GridSearchCV*.

```
{'C': 1, 'class_weight': None, 'dual': False, 'fit_intercept': False,
'intercept_scaling': 1, 'max_iter': 100, 'multi_class': 'ovr',
'n_jobs': 1, 'penalty': 'l2', 'random_state': 8, 'solver': 'newton-
cg', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}
```

***Código 22**. Parâmetros do modelo treinado utilizando o algorítimo LogisticRegression.*

A solução desenvolvida nesse projeto, testando diferentes tipos de algorítimos de *Machine Learning*, realizando um refinamento nos melhores algorítimos foi completada com sucesso, conseguindo completar o objetivo de maneira simples e direta.

V. Conclusão

Foma Livre de Visualização

Utilizando o melhor modelo do projeto, obtive o seguinte resultado **0.7090** para o *accuracy,* **0.7118** para o *fbeta* e **0.7089** para *roc/auc*, significa que nosso modelo ira acertar **70.9%** dos dados do *dataset*. A cada 1.000 consultas médicas marcadas, nosso modelo consegue predizer se o paciente realmente irá comparecer a consulta médica ou irá faltar a ela em 709 consultas médicas.

Esse modelo pode evitar que empresas tenha gastos desnecessários com a falta do paciente em consultas médicas. Vamos supor que nosso modelo identificou que um paciente pode faltar a uma consulta médica, automaticamente um sistema pode enviar um aviso via mensagem de texto ou ligar para o paciente, lembrando ele de sua consulta médica.

Através do modelo pode-se criar um sistema identificando quais são os melhores dias e horários para marcar uma consulta médica para um paciente. Com os dados do paciente em um sistema, este identifica qual é a próxima consulta médica disponível, e automaticamente define o melhor dia e horário para o paciente, evitando possíveis faltas que poderiam acontecer.

A ***Figura 8*** mostra uma amostra da arvore de decisão gerada utilizando os dados do *dataset*. A arvore gerada ficou enorme, mas já é possível ter uma ideias de quais serão os melhores horários e dias para marcar a consulta para um paciente.

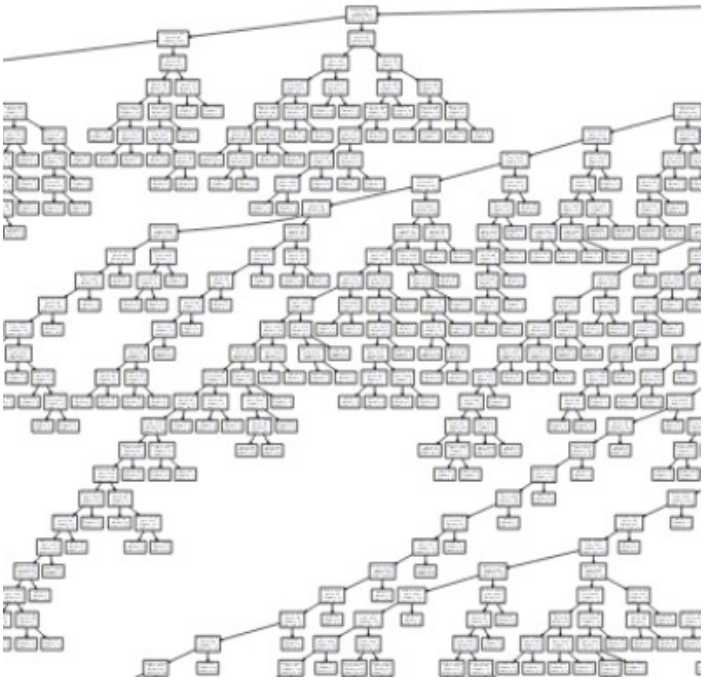

***Figura 8**: Amostra da imagem de arvore de decisão gerada utilizando a classificação utilizada pelo algorítimo DecisionTreeClassifier.*

Reflexão

O primeiro passo do projeto foi escolher o *dataset* que iria ser utilizado, gostaria que esse *dataset* fosse na área da saúde, e fosse de classificação. Procurei em vários site com *datasets*, ate que encontrei um que acreditava que iria ser ótimo para este projeto, este *dataset* foi encontrado no *Kaggle*, e se chama **Medical Appointment No Shows**, e esta disponível neste link.

Após a escolha do *dataset* foi escrito a proposta para o projeto **Capstone**, utilizando o modelo disponibilizado pela *Udacity*. Neste documento, foi explicado qual é a proposta do projeto, qual seria o assunto, uma descrição do problema, a estrutura do *dataset*, uma descrição da solução, o modelo de referencia, as métricas que seria utilizadas e por fim o design do projeto.

Depois de enviado a proposta e ela ser aprovada, foi iniciado o documento e desenvolvimento pratico do projeto **Capstone**. Por primeiro foi criado o projeto no ambiente de desenvolvimento em *Python* chamado *Jupyter Notebook*, e começado a parte envolvendo programação, carregando e visualizando os dados do *dataset*, analisando informações como quantidade de comparecimento e não nas consultas médicas e os tipos de dados no *dataset*.

Em conjunto com a parte de programação, o documento começou ser escrito também. Quando a parte em programação estava pronta referente ao item do documento, este era escrito no documento, assim avançando todas as etapas do projeto nessa maneira.

Foram gerados alguns gráficos, que estão representados nesse documento, mostrando a porcentagem dos pacientes que compareceram em suas consultas médicas, a porcentagem dos pacientes que receberam ao menos um SMS lembrando da consulta médica, a porcentagem de homens e mulheres que marcaram consultas médicas e a quantidade de homens e mulheres que marcaram as consultas médicas e compareceram e não compareceram a ela.

Houve dois aspectos importantes neste projeto para mim, o primeiro aspecto envolve a estrutura do *dataset*, que esta organizado, mas para mim podia ser diferente, com nomes ainda mais amigáveis e o formato de como os dados eram retornando definidos de outra maneira. Por este motivo foi alterado os nomes de inúmeras colunas e como os dados eram retornados. Toda essa etapa foi feita em tempo de execução, não alterando o arquivo do *dataset*.

O segundo aspecto importante desse projeto para mim foi a parte de *Machine Learning*, que começou criando um novo *dataset*, com quantidade de consultas médicas comparecidas e não comparecidas em quantidades iguais, depois dividindo os dados em dados de treino e teste, ficando com 80% e 20% de toda a quantidade de linhas do *dataset* respectivamente. Outra parte foi escolher quais seriam os algoritmos que seriam testados para este *dataset*. Após escolhidos, foi iniciado o desenvolvimento dos modelos de aprendizado. Apos os modelos terem sido desenvolvidos e estavam prontos, foi desenvolvido um processo que salva este aprendizado em arquivos do tipo *.pkl*, evitando toda vez que executa o *notebook* no *Jupyter Notebook* precisar realizar o treinamento novamente.

O aspecto mais difícil neste projeto não foi nem o documento e nem a programação, mas sim aguardar o treinamento dos dados e encontrar os melhores parâmetros para eles, alguns dos algorítimos utilizado demoram muito tempo para realizar estas tarefas.

Foi realizado o teste de nossos modelos no *dataset* completo, verificando se nossos modelos realmente estavam bom, e o modelo treinado no algorítimo *GaussianNB* foi o melhor, conseguiu uma pontuação de **0.7572** para *accuracy*, **0.8234** para *fbeta* e **0.5099** para *roc/auc*. É possível visualizar esta informação no arquivo de programação, desenvolvido no Jupyter Notebook.

Foi gerado uma imagem em um arquivo PDF no formato de arvore de decisão, utilizando o modelo refinado do algorítimo *DecisionTreeClassifier*,para visualizar como foi definido a classificação dos dados para este algorítimo.

Ao final, foi treinado novos modelo, mas agora com todos os dados do *dataset*, sendo 110527 linhas, e o algorítimo *GradientBoostingClassifier* foi o que conseguiu a melhor pontuação, tendo **0.7972** para *accuracy*, **0.8309** Para *fbeta* e **0.5023** para *roc/auc*. É possível visualizar esta informação no arquivo de programação, desenvolvido no Jupyter Notebook.

O modelo e a solução final com o algorítimo *RandomForestClassifier* alinharam com o objetivo do projeto, que erá conseguir a pontuação de no mínimo 0.70 tanto em *accuracy*, *fbeta* e *roc/auc*. Os outros algorítimos conseguiram uma boa pontuação, mas não conseguiram o objetivo do projeto.

Melhorias

Algumas das melhorias futuras que poderiam ser implementadas no projeto seria utilizar mais algorítimos de *Machine Learning*, para ver se haveria melhora na pontuação do *accuracy, fbeta e roc/auc, e u*tilizar o algorítimo *GridSearchCV* em todos esses novos algorítimos de *Machine Learning*, para identificar os melhores parâmetros para eles.

Outra possibilidade seria implementar um algorítimo de *deep learning*, para ver se ele iria desempenhar melhor que os demais algorítimos de *Machine Learning*.

Possivelmente deve existir algum outro método para predizer os dados desse *dataset* de forma que obtenha uma melhor pontuação. Como a área de *Machine Learning* é enorme, existindo inúmeras possibilidades para a resolução de problemas, foi escolhido o método descrito nesse documento e utilizando esses algorítimos de *Machine Learning.* Esse método foi muito bom em minha opinião, atingindo a pontuação de **0.7090** para *accuracy*, **0.7118** para *fbeta* e **0.7089** para *roc/auc*.

VI. Referências

Kaggle – **Medical Appointment No Shows**. Disponível em: <https://www.kaggle.com/joniarroba/noshowappointments>. Acesso em: 03 de setembro de 2018.

Pandas – **Python Data Analysis Library**. Disponível em: <https://pandas.pydata.org/index.html>. Acesso em: 05 de setembro de 2018.

Matplotlib – **Introduction - Documentation**. Disponível em: <https://matplotlib.org/users/intro.html>. Acesso em: 05 de setembro de 2018.

NumPy – **About NumPy**. Disponível em: <https://docs.scipy.org/doc/numpy-1.13.0/about.html>. Acesso em: 06 de setembro de 2018.

scikit-learn – **About Us**. Disponível em: <http://scikit-learn.org/stable/about.html>. Acesso em: 06 de setembro de 2018.

Paulo Vasconcellos — Cientista de Dados brasileiro – **Como saber se seu modelo de Machine Learning está funcionando mesmo**. Disponível em: <https://paulovasconcellos.com.br/como-saber-se-seu-modelo-de-machine-learning-est%C3%A1-funcionando-mesmo-a5892f6468b>. Acesso em: 09 de setembro de 2018.

Towards Data Science – **Train/Test Split and Cross Validation in Python**. Disponível em: <https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6>. Acesso em: 09 de setembro de 2018.

scikit-learn – **Decision Trees**. Disponível em: <http://scikit-learn.org/stable/modules/tree.html#tree>. Acesso em: 25 de setembro de 2018.

scikit-learn – **Gaussian Naive Bayes**. Disponível em: <http://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes>. Acesso em: 25 de setembro de 2018.

scikit-learn – **Logistic regression**. Disponível em: <http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression>. Acesso em: 25 de setembro de 2018.

scikit-learn – **Support Vector Machines - Classification**. Disponível em: <http://scikit-learn.org/stable/modules/svm.html#svm-classification>. Acesso em: 26 de setembro de 2018.

scikit-learn – **Stochastic Gradient Descent**. Disponível em: <http://scikit-learn.org/stable/modules/sgd.html#sgd>. Acesso em: 26 de setembro de 2018.

scikit-learn – **Forests of randomized trees**. Disponível em: <http://scikit-learn.org/stable/modules/ensemble.html#forest>. Acesso em: 26 de setembro de 2018.

scikit-learn – **AdaBoost**. Disponível em: <http://scikit-learn.org/stable/modules/ensemble.html#adaboost>. Acesso em: 27 de setembro de 2018.

scikit-learn – **Gradient Tree Boosting**. Disponível em: <http://scikit-learn.org/stable/modules/ensemble.html#gradient-boosting>. Acesso em: 27 de setembro de 2018.