Relatório Final

Equipe: Lucas Henrique de Assis Monteiro
Curso: Engenharia de Software
Período: 6°
Disciplina: IA
Professor: Cleyton Rodrigues

Sumário

Base de dados escolhida

Descrição

A base de dados escolhida foi a ***"Diabetes Health Indicators Dataset"***, que possui dados de 253.680 respostas da pesquisa do BRFSS 2015. O Behavioral Risk Factor Surveillance System (BRFSS) é um sistema de pesquisas telefônicas relacionadas à saúde que coleta dados estaduais sobre os residentes dos EUA e seus comportamentos de risco relacionados à saúde, condições crônicas de saúde e uso de serviços preventivos.

Esse dataset contém 22 colunas com dados de cada uma das respostas coletadas durante a pesquisa. Essas respostas estão relacionadas a condições de saúde e financeiras dos respondentes, além de dados pessoais como idade, escolaridade, etc.

Ele pode ser acessado através do Kaggle em: https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset?select=diabetes_binary_health_indicators_BRFSS2015.csv.

Entradas

As entradas dos algoritmos são as seguintes 21 colunas do dataset:

HighBP	0 = não tem pressão sanguínea alta; 1 = tem pressão sanguínea alta.
HighChol	0 = não tem colesterol alto; 1 = tem colesterol alto.
CholCheck	0 = não realizou verificação de colesterol em 5 anos; 1 = realizou verificação de colesterol em 5 anos.
BMI	Body Mass Index -> Índice de massa corporal.
Smoker	"Você fumou pelo menos 100 cigarros em toda a sua vida?" 0 = não e 1 = sim.
Stroke	Já teve um derrame. 0 = não e 1 = sim.
HeartDiseaseorAttack	Doença Cardíaca Coronária (DAC) ou Infarto do Miocárdio (IM). 0 = não e 1 = sim.
PhysActivity	Atividade física nos últimos 30 dias - não inclui trabalho. 0 = não e 1 = sim.
Fruits	Consome frutas 1 ou mais vezes por dia. 0 = não e 1 = sim.
Veggies	Consome vegetais 1 ou mais vezes por dia. 0 = não e 1 = sim.

HvyAlcoholConsump	Consumo alto de álcool (homens adultos >=14 doses por semana e mulheres adultas>=7 doses por semana). 0 = não e 1 = sim.
AnyHealthcare	Possui algum tipo de cobertura de saúde, incluindo plano de saúde, planos pré-pagos, etc. 0 = não e 1 = sim
NoDocbcCost	“Houve algum momento nos últimos 12 meses em que você precisou consultar um médico, mas não pôde devido ao custo?” 0 = não e 1 = sim.
GenHlth	“Você diria que em geral sua saúde é:“ 1 = excelente; 2 = muito boa; 3 = boa; 4 = regular; 5 = ruim.
MentHlth	Dias de saúde mental ruim: escala de 1 a 30 dias.
PhysHlth	Dias de doença física ou lesão nos últimos 30 dias. Escala de 1 a 30.
DiffWalk	“Você tem sérias dificuldades para caminhar ou subir escadas?” 0 = não e 1 = sim.
Sex	0 = feminino e 1 = masculino.
Age	Categoria de idade (escala de 1 a 13): ex. 1 = 18-24; 9 = 60-64; 13 = 80 ou mais.
Education	Nível de escolaridade (escala de 1 a 6). 1 = Nunca frequentou a escola ou apenas o jardim de infância; 2 = fundamental etc.
Income	Escala de renda (escala de 1 a 8). Ex. 1 = menos de $10.000; 5 = menos de $35.000; 8 = $75.000 ou mais.

Saída

A saída do algoritmo será a variável ***Diabete_binary***. Ela indica se uma pessoa tem diabetes ou pré diabetes e será utilizada para prever se um indivíduo possui diabetes. Os valores são divididos em:

- 0, significa que o indivíduo **não possui** diabetes;
- 1, significa que o indivíduo **possui** pré diabetes ou diabetes.

Semana 1

No projeto da primeira semana foi realizada a primeira aplicação do algoritmo através do método de classificação chamado de *Decision Tree.* O colab para o projeto da primeira semana pode ser acessado AQUI.

Testando Projeto I

Primeiramente, foi printado o "*shape*" da base que ajudou a confirmar que o dataset possuía 253.680 linhas de registro e 22 colunas. Depois, utilizou-se da função *head(150)* para observação dos 150 primeiros valores das linhas dessas colunas. Também foi usado o *describe()* para exibição de um resumo de métricas como: a contagem de registro, a média, os valores mínimo e máximo, bem como alguns percentis.

Outro ponto observado foi a distribuição por classe da viável de saída. Essa distribuição permite a visualização de quantos registros existem por classe no dataset. Como podemos ver na figura abaixo, a classe 0 (pessoas SEM diabetes ou pré-diabetes) possui muito mais casos que a classe 1 (pessoas COM diabetes ou pré-diabetes). Portanto, pode-se concluir que a base está desbalanceada.

```
Diabetes_binary
0.0    218334
1.0     35346
dtype: int64
```

Após isso, foi plotado o gráfico de caixas (*boxplot*). Nele, foi possível observar a presença de *outliers* (pontos fora da curva), que são dados que se diferenciam drasticamente de todos os outros. No entanto, no caso da variável de saída, que é classificada binariamente, os valores que possuem menos ocorrências (nesse caso seria a classe 1) foram identificados erroneamente como *outliers*. Isso pode ter sido causado pelo desbalanceamento.

Ademais, foram plotados: um histograma, onde foi apresentada uma visão geral da distribuição de todas as classes; e um gráfico de dispersão, que por ter muitas colunas e os valores das colunas não variarem muito, sua análise ficou bem difícil. Todos esses gráficos estão presentes no projeto do Colab.

Para a aplicação do algoritmo, o dataset foi dividido para que uma parte fosse utilizada para os treinos (80%) e outra para a validação (20%). Após isso, foi

chamado o Decision Tree Classifier para realização do treinamento, através da função fit(). Após isso, as previsões foram feitas utilizando *predict()*.

Finalmente, foram printadas a *Confusion Matrix* e o *Classification Report*, para avaliar o desempenho do algoritmo. Os resultados são apresentados na figura abaixo.

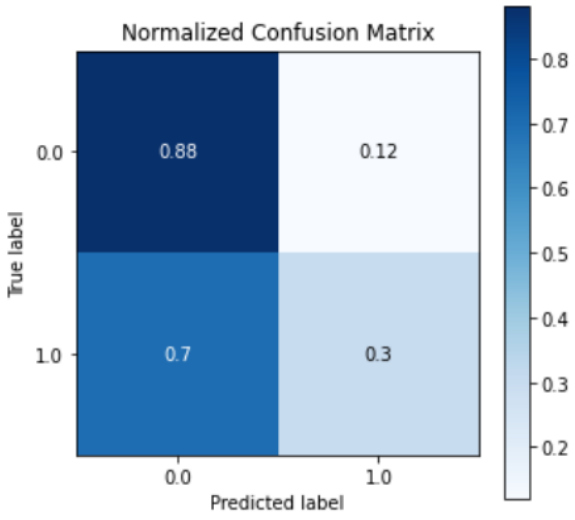
```
[[38485  5184]
 [ 4920  2147]]
              precision    recall  f1-score   support

         0.0       0.89      0.88      0.88     43669
         1.0       0.29      0.30      0.30      7067

    accuracy                           0.80     50736
   macro avg       0.59      0.59      0.59     50736
weighted avg       0.80      0.80      0.80     50736
```

Com isso, pode-se observar que a acurácia do algoritmo é de 0.80. No entanto, apenas isso não é o suficiente para avaliar a eficiência do algoritmo. A precisão e o recall da classe 0 foram bons, mas o mesmo não ocorreu para a classe 1 que teve valores bem baixos. Isso indica que o modelo está apresentando muitos erros para essa segunda classificação.

Para melhor análise, foi plotada a matriz de confusão que permite a realização de uma análise visual dos resultados.



Pode-se visualizar que para a classificação 0 (sem diabetes) o algoritmo conseguiu prever corretamente (Verdadeiros Positivos) 88% dos casos de teste, o que já é um resultado bom. Já para a classificação 1 (com diabetes ou pré-diabetes), o algoritmo acertou apenas 30% dos casos.

Dessa forma, o algoritmo não está apresentando uma boa eficiência para classificação de pessoas com diabetes ou pré-diabetes, apresentando muitos Falsos Negativos para essa classe. Isso pode ser considerado um problema crítico, já que as consequências de classificar erroneamente uma pessoa com a doença como sadia são bem maiores que o contrário.

Verificação de valores nulos

A verificação de valores nulos permite identificar quantos valores ausentes estão presentes no dataset e excluí-los para não interferir no algoritmo. Dessa forma, a função info() foi utilizada para a visualização de informações sobre esses valores. A partir disso, foi possível identificar que todos os dados do dataset eram representados por números do tipo *float64* e que todos os valores eram não-nulos.

Para comprovar essa última afirmação foram utilizadas as funções *isna().sum()*, para printar a soma dos valores nulos de cada coluna. O resultado de todas foi zero, o que comprova o fato de que o dataset não possui nulos.

```
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Diabetes_binary       253680 non-null  float64
 1   HighBP                253680 non-null  float64
 2   HighChol              253680 non-null  float64
 3   CholCheck             253680 non-null  float64
 4   BMI                   253680 non-null  float64
 5   Smoker                253680 non-null  float64
 6   Stroke                253680 non-null  float64
 7   HeartDiseaseorAttack  253680 non-null  float64
 8   PhysActivity          253680 non-null  float64
 9   Fruits                253680 non-null  float64
 10  Veggies               253680 non-null  float64
 11  HvyAlcoholConsump     253680 non-null  float64
 12  AnyHealthcare         253680 non-null  float64
 13  NoDocbcCost           253680 non-null  float64
 14  GenHlth               253680 non-null  float64
 15  MentHlth              253680 non-null  float64
 16  PhysHlth              253680 non-null  float64
 17  DiffWalk              253680 non-null  float64
 18  Sex                   253680 non-null  float64
 19  Age                   253680 non-null  float64
 20  Education             253680 non-null  float64
 21  Income                253680 non-null  float64
dtypes: float64(22)
memory usage: 42.6 MB
```

```
Número de valores nulos em diferentes colunas:
-------------------------------------------------
Diabetes_binary         0
HighBP                  0
HighChol                0
CholCheck               0
BMI                     0
Smoker                  0
Stroke                  0
HeartDiseaseorAttack    0
PhysActivity            0
Fruits                  0
Veggies                 0
HvyAlcoholConsump       0
AnyHealthcare           0
NoDocbcCost             0
GenHlth                 0
MentHlth                0
PhysHlth                0
DiffWalk                0
Sex                     0
Age                     0
Education               0
Income                  0
dtype: int64
-------------------------------------------------
```

Balanceamento

O desbalanceamento de dados é um fator que pode impactar diretamente o algoritmo de classificação. Isso porque ele irá realizar muito bem a classificação da classe majoritária, mas não terá o mesmo desempenho na minoritária.

O que pode acontecer nesse caso é que esse desequilíbrio (mais casos sem diabetes) acaba tendenciando o algoritmo, o que faz ele classificar erroneamente

entradas para classe minoritária. Isso foi visto anteriormente, onde mesmo com acurácia alta, o algoritmo teve um desempenho mais baixo na previsão de indivíduos com diabetes ou pré-diabetes do que os sem.

Para contornar o problema do mal balanceamento da base de dados, é possível utilizar 2 abordagens. A primeira, chamada de *Over Sampling*, pretende criar novos valores para a classe minoritária a partir das informações dos dados originais. Já a segunda, se chama *Under Sampling* e pretende reduzir os valores da classe majoritária.

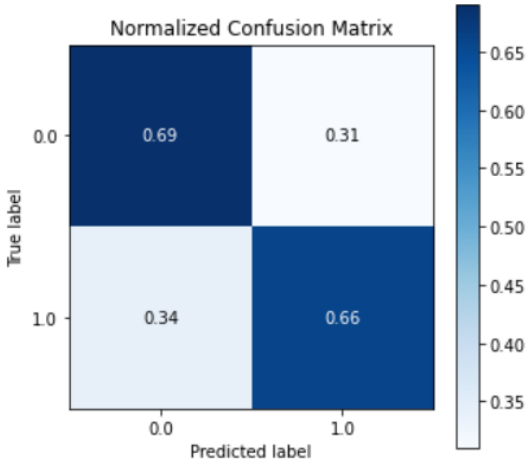
Under Sampling

O primeiro método a ser testado foi o *Under Sampling.* Nesse caso, as classes foram divididas da coluna “Diabetes_binary” para que a **classe 0** fosse **diminuída** com base no número de registros da classe 1 (o processo pode ser visto no Colab). Assim, foram selecionados aleatoriamente alguns registros da classe majoritária até que ambas as classes ficaram com a quantidade de 35346. A partir disso, foi realizado um novo treinamento com essa base balanceada, seguindo os mesmos processos.

```
[[4922 2176]
 [2401 4640]]
              precision    recall  f1-score   support

         0.0       0.67      0.69      0.68      7098
         1.0       0.68      0.66      0.67      7041

    accuracy                           0.68     14139
   macro avg       0.68      0.68      0.68     14139
weighted avg       0.68      0.68      0.68     14139
```



É possível observar que a acurácia **diminuiu** de **0.80** para **0.68**. Sobre as outras métricas, apesar de terem sido melhoradas para classe 1, elas foram reduzidas para a classe 0. No geral, houve um certo balanceamento na quantidade de acertos para as duas classes, mas a eficiência do modelo foi significativamente reduzida.

Over Sampling

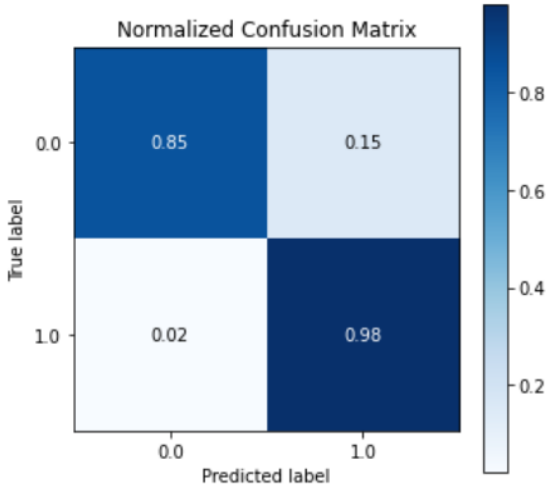
O outro método utilizado foi o de *Over Sampling.* As classes foram novamente divididas da coluna “Diabetes_binary”, mas dessa vez era para que a **classe 1** fosse

aumentada com base no número de registros da classe 0. Assim, foram replicados aleatoriamente alguns registros da classe minoritária até que ambas as classes ficaram com a quantidade de 218334. O processo de testagem foi o mesmo.

```
[[37284  6511]
 [  816 42723]]
              precision    recall  f1-score   support

         0.0       0.98      0.85      0.91     43795
         1.0       0.87      0.98      0.92     43539

    accuracy                           0.92     87334
   macro avg       0.92      0.92      0.92     87334
weighted avg       0.92      0.92      0.92     87334
```



Nesse caso, a acurácia **aumentou** de **0.80** para **0.92**, o que é um resultado bem satisfatório. Para a classe 1 o número de Verdadeiros Positivos aumentou muito, o modelo conseguiu prever 98% dos casos corretamente. A classe 0 apresentou uma pequena redução de 3% no número de valores previstos corretamente.

Conclusão

Pode-se concluir, a partir das abordagens aplicadas, que o método de *Over Sampling* apresentou resultados bons. Além de conseguir aumentar a acurácia do modelo, as métricas da classe 1 foram melhoradas e o problema crítico da alta quantidade de Falsos Negativos foi solucionado. No entanto, há de se considerar se os altos valores apresentados, na verdade, não representam um *overfitting.*

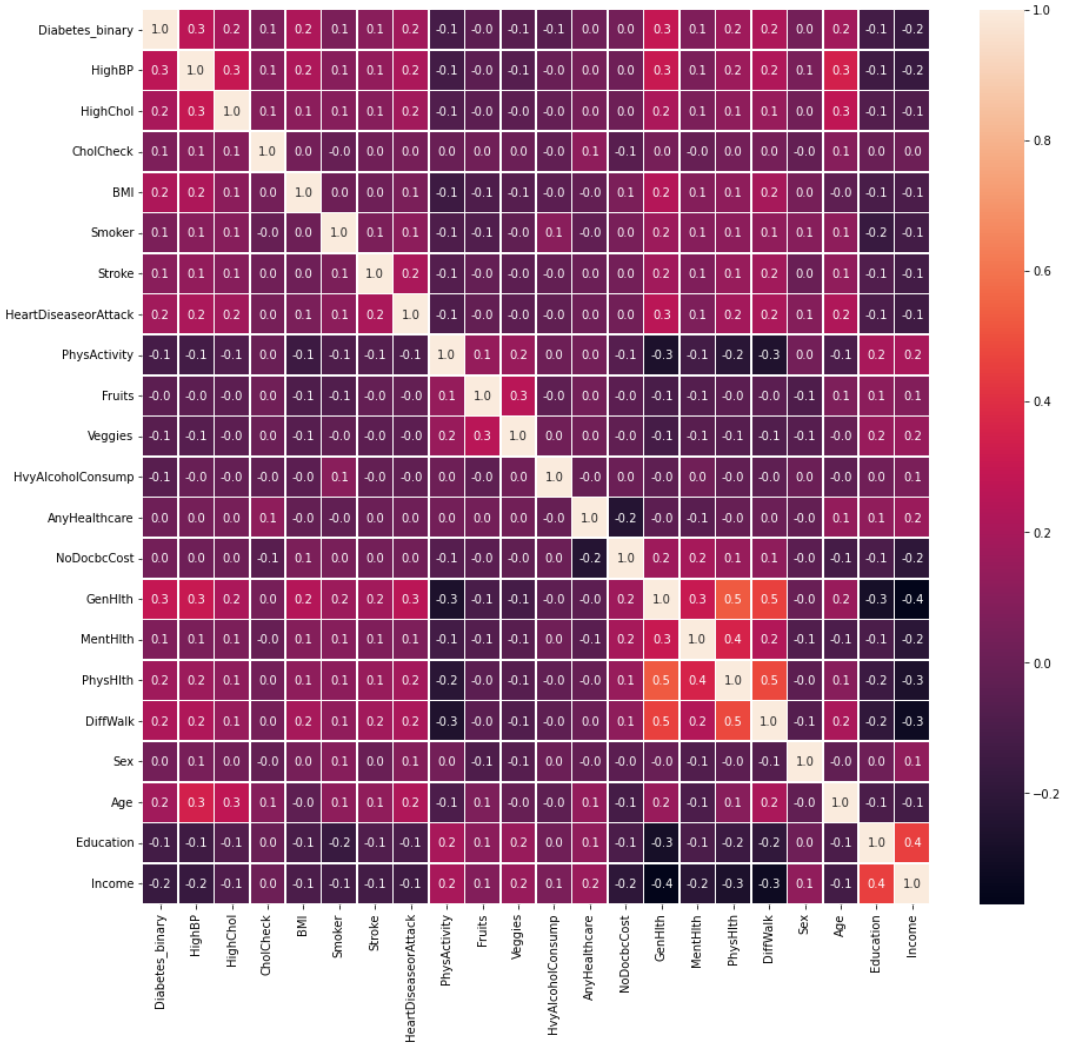
Sobre a diminuição da eficiência do modelo pelo uso do *Under Sampling*, esse resultado foi obtido devido a base possuir uma grande quantidade de dados. A redução dos registros da classe 0 no método de *Under Sampling* ocasionou a perda de muitas informações que fizeram diferença para o treinamento do modelo.

Semana 2

Na segunda semana foram testados o Projeto II e o Projeto III. No III havia algumas técnicas para melhorar o dataset e no II foi feita uma comparação dos métodos de classificação utilizando a Validação Cruzada (K-Fold). Os 2 projetos estão em um único notebook para facilitar a análise e podem ser acessados clicando AQUI.

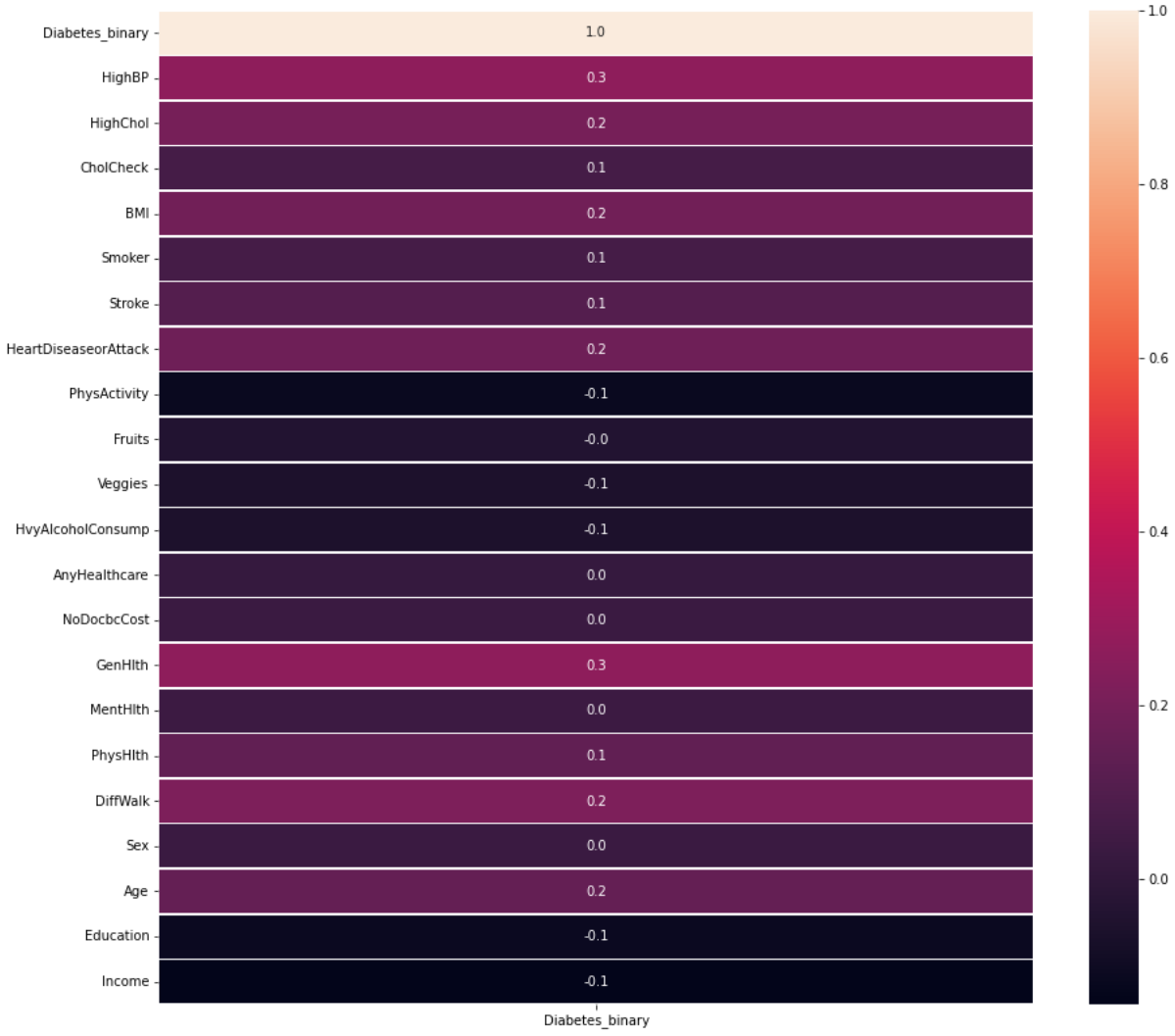
Testando Projetos II e III

Inicialmente, além das funções e gráficos que foram testados no Projeto I, também foi feita uma análise de correlação utilizando o método de *Pearson* com a função *corr()*. Um gráfico da matriz de correlação foi plotado (figura abaixo) para que fosse possível a representação visual da correlação entre as colunas do dataset.



A partir dele, é possível concluir que nenhuma das colunas possui correlação forte. A maioria não possui correlação entre si, outras apresentaram correlação fraca e moderada. As maiores correlações são as de *PhyHlth*, *GenHlth* e *DiffWalk que* possuem correlação moderada (0.5) entre si.

Para realizar uma análise mais voltada para a variável de saída, foi *plotado* mais um gráfico de correlação entre ela e as variáveis de entrada.



Aqui é possível observar que a presença de duas correlações fracas (0.3) e as outras não possuem correlação. Cinco variáveis apresentaram correlação 0. Portanto, não há nenhuma correlação forte com a variável de saída.

Por último, foi realizado um teste para avaliação dos métodos de classificação, utilizando o K-Fold para treinamento dos modelos. Foram avaliados os algoritmos: *Logistic Regression (LR), Linear Discriminant Analysis (LDA), KNeighbors Classifier*

(KNN), DecisionTreeClassifier (CART), GaussianNB (NB) e Support Vector Machine (SVM). Os resultados podem ser vistos na figura abaixo.

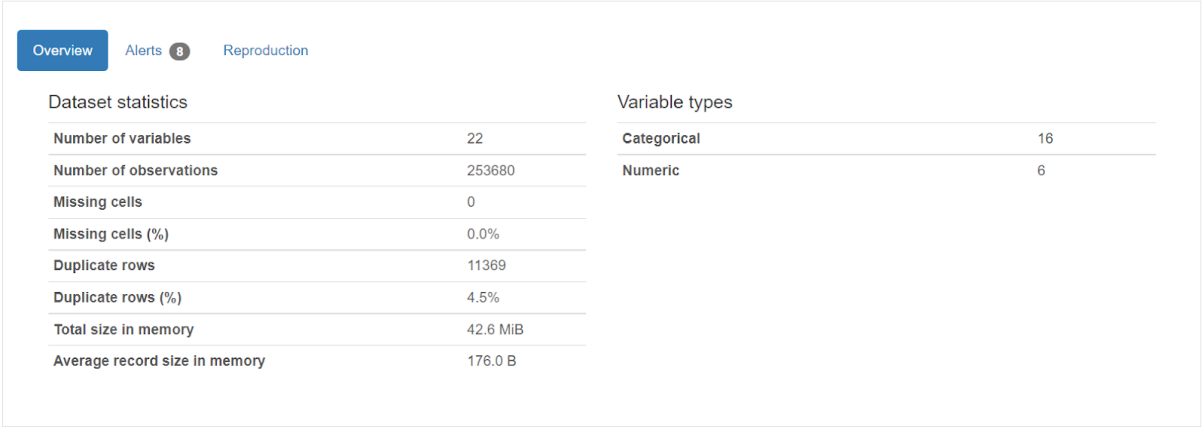
```
LR: 0.863024 (0.007478)
LDA: 0.861069 (0.008356)
KNN: 0.847323 (0.005973)
CART: 0.803508 (0.006391)
NB: 0.774980 (0.010501)
```

Dentre os métodos avaliados, o *LR* e o *LDA* foram os que apresentaram os melhores resultados. Houve uma diferença bem pequena na acurácia dos dois. O método SVM teve que ser retirado do teste, pois seu desempenho foi bem ruim. Passaram-se mais de 4 horas e o algoritmo ainda estava em execução, o que o tornou inviável.

O SVM plota cada dado como um ponto no espaço com n dimensões. Então, a classificação é realizada procurando a melhor "fronteira de separação" (ou hiperplano) possível entre classes para um dado conjunto de dados (que precisam ser linearmente separáveis). Dessa forma, para bases como a em questão, que possuem muitos registros e muitas colunas, o tempo que o algoritmo precisa para o treinamento é muito grande.

Overview do dataset

Para a verificação da base de dados, foi utilizada uma biblioteca chamada *pandas-profiling* que gera relatórios de perfil para *DataFrames* do pandas. Ele automaticamente apresenta informações como: tipo de variáveis, estatísticas, valores mais frequentes, histogramas, correlações (Spearman, Pearson e Kendall), valores nulos, etc. Todas elas podem ser vistas no Colab.



Nessa parte do relatório constata-se, de forma resumida, algumas estatísticas gerais da base. Ele também emite alguns alertas (figura abaixo) que podem servir como um ponto de partida para começar a implementar melhorias.

Primeiramente, há um aviso sobre o número de zeros nas colunas *MenHlth* e *PhysHlth.* Observando o contexto dessas variáveis, elas representam a quantidade de dias que o respondente apresentou algum problema mental ou físico, respectivamente, nos 30 dias precedentes à pesquisa. Dessa forma, é completamente normal que uma boa parte dessas pessoas não tenha apresentado nenhum desses problemas durante esse período.

Removendo registros duplicados

Sobre os registros duplicados, eles foram removidos para ver de que forma eles impactam o desempenho dos modelos. Para isso, foi utilizada a função *drop_duplicates.* Após a exclusão dos registros, a base ficou distribuída dessa forma:

```
Diabetes_binary
0.0    194377
1.0     35097
dtype: int64
```

Para ver os efeitos dessa alteração, foi realizada uma nova avaliação dos métodos. Os resultados podem ser vistos abaixo.

```
LR: 0.849626 (0.009301)
LDA: 0.848732 (0.009770)
KNN: 0.831759 (0.008544)
CART: 0.781522 (0.009449)
NB: 0.756870 (0.013931)
```

As acurácias de todos os modelos foram reduzidas, o que aparentemente não representa um bom resultado. Esse fato provavelmente ocorreu, pois as linhas repetidas estavam sendo benéficas à base. A replicação desses dados pode, na verdade, estar representando a probabilidade dos dados ocorrerem no mundo real.

Por exemplo, um caso onde o mesmo conjunto de entradas tem um registro de saída para classe 0 e um para classe 1 simultaneamente. O algoritmo terá uma probabilidade de 50:50 de acertar a predição, já que ele terá que decidir qual das classes será apresentada.

No entanto, se você possui 4 registros repetidos que apontam para classe 0 e apenas 1 registro para classe 1, isso pode significar que no cenário das respostas a que apareceu mais vezes é a que representa melhor aquela situação. Assim o modelo terá uma probabilidade de 80% para 1 classe e 20% para segunda classe. Isso o fará errar menos na classificação. Conclui-se, então, que é melhor manter as linhas duplicadas.

Removendo colunas

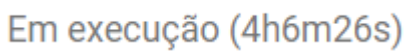
Avaliando os resultados da análise de correlação é possível ver que algumas colunas não se correlacionam com a variável de saída. Então, uma possibilidade seria excluir essas colunas de forma que a quantidade de dados fosse reduzida para melhorar o desempenho do algoritmo.

Para isso, foram removidas com a função *drop()* todas as colunas que apresentavam correlação 0.0 com a variável *Diabetes_binary,* sendo elas: *Fruits, AnyHealthcare, NoDocbcCost, MentHlth e Sex.* Após rodar o teste novamente, os resultados obtidos foram estes:

```
LR: 0.863024 (0.007321)
LDA: 0.861073 (0.008438)
KNN: 0.847848 (0.005447)
CART: 0.816123 (0.007540)
NB: 0.777858 (0.010320)
```

Comparando com os resultados do teste inicial, os valores de acurácia praticamente não mudaram em nada. Isso é positivo, pois mostra que a retirada das colunas não afetou a eficiência dos modelos. O ganho, portanto, seria na performance, já que a execução dos algoritmos foi mais rápida se comparada às outras avaliações.

Com essa melhoria, foi realizado um novo teste com o método SVM para avaliar a sua viabilidade. Como pode ser visto abaixo, o resultado ainda não foi bom. O algoritmo ultrapassou, novamente, mais de 4 horas de execução. O que o deixou de fora da avaliação.



Removendo Outliers

Os outliers são dados que se diferenciam de forma dastrica de todos os outros presentes no dataset. Portanto, eles representam valores que fogem da normalidade dos dados e que podem causar anomalias nos resultados obtidos nos algoritmos de predição.

Como foi descrito acima, a partir da análise do gráfico de caixas (boxplot) foi possível visualizar a presença de alguns outliers nos registros da base avaliada. Eles podem ter ocorrido devido a erros na coleta de dados ou até mesmo por individuos que apresentam característicasou casos mais específicos que distoam da norma.

Portanto, como tentativa de melhorar o desempenho do modelo, foi feita a remoção desses “pontos fora da curva” através de uma estratégia que utiliza o *Z-score*. Esse escore é um valor que ajuda a entender se um valor de dados é maior ou menor que a média e a que distância ele está da média.

Na maior parte dos casos, um limite de 3 ou -3 é usado para esse valor. Portanto, se o valor do *Z-score* for maior ou menor que 3 ou -3, respectivamente, esse ponto de dados será identificado como discrepante. Então, esse ponto de dados pode ser considerado um *Outlier*.

Para detectar os Z-scores dos dados da base, foi utilizada a função *zscore()* da biblioteca *spicy.* Na imagem abaixo, são apresentados uma parte desses valores para cada um dos registros. A saída completa pode ser vista no colab.

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	\
0	0.402355	1.153688	1.165254	0.196922	1.757936	1.120927	
1	0.402355	0.866785	0.858182	5.078164	0.511806	1.120927	
2	0.402355	1.153688	1.165254	0.196922	0.057858	0.892119	
3	0.402355	1.153688	0.858182	0.196922	0.209174	0.892119	
4	0.402355	1.153688	1.165254	0.196922	0.663122	0.892119	
...	...	...	...	...	...	...	
253675	0.402355	1.153688	1.165254	0.196922	2.514516	0.892119	
253676	2.485367	1.153688	1.165254	0.196922	1.571019	0.892119	
253677	0.402355	0.866785	0.858182	0.196922	0.057858	0.892119	
253678	0.402355	1.153688	0.858182	0.196922	0.814438	0.892119	
253679	2.485367	1.153688	1.165254	0.196922	0.511806	0.892119	

A partir disso, foi definido um *threshold* (valor limite) de 3 para o Z-score. Assim, todos os dados que possuem um Z-score maior que 3, seriam identificados como *Outliers.* Esses valores foram removidos do base pelo comando abaixo:

```
new_dataset_Z = new_dataset[(z < 3).all(axis=1)]
```

Após a remoção desses valores, o dataset ficou distribuído da seguinte forma:

```
(196579, 17)
```

```
Diabetes_binary
0.0    173122
1.0     23457
dtype: int64
```

Por último, foi realizado um novo teste para ver como essa mudança havia impactado a acurácia dos algoritmos. Os resultados podem ser vistos abaixo.

```
LR: 0.881503 (0.004982)
LDA: 0.880287 (0.005073)
KNN: 0.867722 (0.005322)
CART: 0.839261 (0.005573)
NB: 0.820103 (0.008010)
```

Observa-se que todos os algoritmos de classificação apresentaram um aumento no valor de suas acurácias. Dessa forma, a remoção dos *Outliers* contribuiu positivamente para o modelo.

Semana 3

Na terceira semana foi avaliado o uso do algoritmo LDA para a criação do modelo de classificação. A ideia era testar as funcionalidades do *Linear Discriminant Analysis,* realizar um teste e avaliar o desempenho. O colab pode ser acessado clicando AQUI.

Linear Discriminant Analysis

O LDA é um modelo de classificação e tem como ideia principal encontrar uma combinação linear de características que caracterizam ou separam duas ou mais classes para que seja possível tomar a melhor decisão para classificá-los. A biblioteca *sklearn* disponibiliza uma classe chamada *LinearDiscriminantAnalysis* que permite realizar classificações utilizando-se desse modelo. Essa classe possui alguns parâmetros, atributos e métodos adicionais que permitem mudar algumas métricas utilizadas durante a classificação.

O LDA faz os registros serem projetados em um espaço vetorial discriminante ideal para que seja possível obter o melhor efeito possível de extração de informações para classificação. O intuito é que os registros tenham uma distância menor dentro da classe à qual pertencem e que haja uma distância maior entre as classes. Assim sendo, ele preza pela maximização da separação entre as classes.

A representação da LDA é bastante direta. Consiste em propriedades estatísticas dos seus dados, calculados para cada classe. Para uma única variável de entrada, isso inclui:

- O valor médio para cada classe
- A variação calculada para todas as classes

As previsões são feitas calculando um valor diferenciado para cada classe e fazendo uma previsão para a classe com o maior valor.

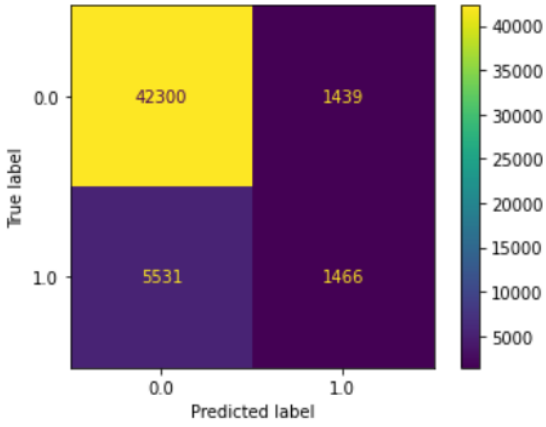
Testando LDA

Primeiramente, o dataset foi dividido para separar a parte que será utilizada nos teste da parte que será utilizada para validação. Depois, foi chamada a classe do algoritmo e os métodos *fit()* e *predict()* para realizar o ajuste e a predição do modelo, respectivamente. Nessa parte não foi utilizado nenhum parâmetro ou atributo adicional.

```
LDA = LinearDiscriminantAnalysis()
LDA.fit(X_train, Y_train)
LDA_pred=LDA.predict(X_test)
```

A partir disso, foram printados o *Classification Report* e a *Confusion Matrix,* para avaliar o desempenho. Os resultados iniciais podem ser visualizados abaixo.

	precision	recall	f1-score	support
0.0	0.88	0.97	0.92	43646
1.0	0.51	0.20	0.29	7090
accuracy			0.86	50736
macro avg	0.70	0.59	0.61	50736
weighted avg	0.83	0.86	0.83	50736

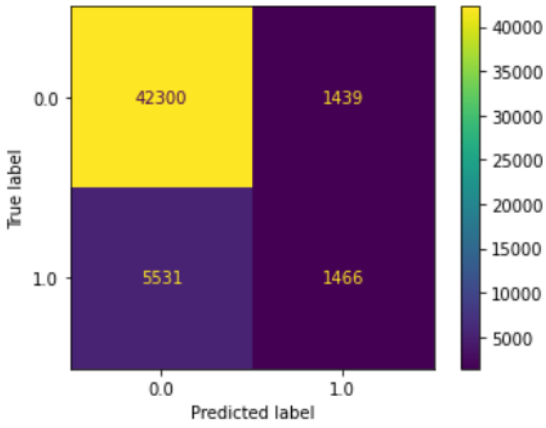


A classe LDA possui um parâmetro chamado de *solver* que possui os valores ***'svd', 'lsqr' e 'eigen'.*** O padrão é o ***'svd'*** e, portanto, foi o utilizado para os resultados acima. Ele significa Decomposição de Valor Singular e não calcula a matriz de covariância. Portanto, este solver é recomendado para dados com um grande número de características.

Para poder verificar se a escolha do solver poderia impactar o resultado do algoritmo, foram feitos alguns testes com os outros 2 comparando as métricas obtidas em cada um.

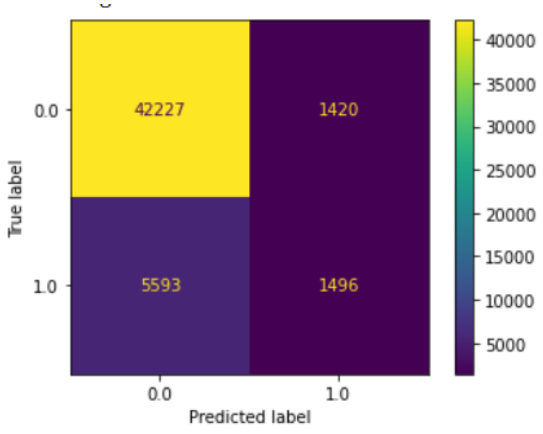
Primeiro, foi testado o ***'lsqr'*** ou Solução de Mínimos Quadrados. Como pode ser visto abaixo, não houveram diferenças entre ele e o ***'svd'***.

	precision	recall	f1-score	support
0.0	0.88	0.97	0.92	43646
1.0	0.51	0.20	0.29	7090
accuracy			0.86	50736
macro avg	0.70	0.59	0.61	50736
weighted avg	0.83	0.86	0.83	50736



O segundo solver testado foi o ***'eigen'*** ou Decomposição de Autovalor (Eigenvalue). Esse solver também não apresentou nenhuma mudança relevante nas métricas, quando comparada aos outros dois.

	precision	recall	f1-score	support
0.0	0.88	0.97	0.92	43647
1.0	0.51	0.21	0.30	7089
accuracy			0.86	50736
macro avg	0.70	0.59	0.61	50736
weighted avg	0.83	0.86	0.84	50736



Conclusão

Mesmo não havendo mudanças nas métricas quando foram alterados os *solvers*, os resultados apresentados pelo Linear Discriminant Analysis foram muito bons. Comparado aos outros métodos avaliados no projeto II, o LDA possui um dos melhores desempenhos para a realização da tarefa de classificação no contexto representado pela base. Dessa forma, ele apresenta-se como uma estratégia bastante viável no cenário apresentado (prever se a pessoa possui ou não diabetes).