Engenharia de Computação

Inteligência Artificial

Algoritmos de Aprendizado Supervisionado - Implementação

Helena Tavares

Junho de 2022

1. Introdução

Existe uma série de algoritmos que manipulam bases de dados, a partir de um modelo padrão. Aprendizagem de máquina é o que define esse conjunto de algoritmos.

A partir de um conjunto de dados inteiro, se define parte dele para treinamento e o restante para testes, relacionando um padrão de valores para entrada e saída, para poder analisar o quanto o algoritmo "acertou" na classificação.

Dentro do grupo de algoritmos de *machine learning*, temos três divisões: algoritmos supervisionados, algoritmos não supervisionados e aprendizado por reforço. Neste trabalho trataremos apenas dos supervisionados, que trabalham com conjuntos de dados que tem rótulos, o que dá uma maior precisão no resultado. Já os não supervisionados não são rotulados, e o algoritmo tem que fazer a classificação por conta própria.

Os algoritmos utilizados foram: *Logistic Regression*, *Random Forest Classifier*, *K-Neighbors Classifier*.

2. Algoritmos de aprendizado supervisionado

2.1. *Logistic Regression*

É um dos algoritmos de classificação mais utilizados. Esse algoritmo faz classificações de dados em diversas categorias, não em apenas duas. Trabalha também com conceitos de probabilidade e estatística.

Regressão logística mede a relação entre a variável dependente categórica e uma ou mais variáveis independentes, estimando as probabilidades usando uma função logística. Ou seja, analisa diferentes aspectos e variáveis do dado e então define a classe mais adequada.

Existem três variações desse algoritmo, que são: regressão logística binomial, ordinal e multinominal.

2.2. *Random Forest Classifier*

Também é um dos algoritmos de aprendizagem de máquina mais utilizados por entregar ótimos resultados, mesmo sem ajustes de hiperparâmetros e ser de fácil entendimento e utilização. Além de poder ser aplicado em tarefas de classificação e também de regressão.

O algoritmo Floresta Aleatória cria uma combinação de árvores de decisão, de forma aleatória, e na maioria dos casos utiliza o método de *bagging*.

Esse algoritmo adiciona aleatoriedade ao modelo enquanto cria as árvores de decisão, assim, ao invés de buscar pela melhor característica ao fazer a divisão dos nodos, ele procura a melhor característica em um subconjunto aleatório das características. Esse processo gera uma vasta diversidade de possibilidades, o que possibilidade a criação de modelos melhores.

2.3. *K-Neighbors Classifier*

O algoritmo KNN, ou algoritmo do vizinho mais próximo, para classificação, também é muito utilizado e muito simples em relação aos cálculos.

É conhecido como algoritmo de aprendizado lento ou preguiçoso (*lazy)* por não precisar de dados de treinamento para obter um modelo. Isso facilita o processo inicial, mas em contrapartida necessita de uma análise mais detalhada posteriormente. Todos

os dados são utilizados na fase de teste, já que não há fase de treinamento, porém, a validação pode ser mais demorada.

Esse algoritmo recebe um dado não classificado, mede a distância (Euclidiana, Manhattan, Minkowski ou Ponderada) do novo e compara com os outros que já foram classificados. Obtém as menores distâncias e verifica a classe de cada dado já analisado e classificado com a menor distância, e verifica quantas classes já foram criadas. A classe que mais apareceu nos dados, com as menores distâncias, será o resultado, então classifica o novo dado baseado nessa classificação.

3. Banco de Dados

3.1. Banco utilizado

A base escolhida para o trabalho foi retirada da plataforma *Keagle,* e disponibilizada pelo Instituto Nacional de Diabetes e Doenças Digestivas e Renais dos Estados Unidos.

O conjunto de informações do banco tem como objetivo prever diagnosticamente casos de diabetes entre mulheres com pelo menos 21 anos com origem indígena Pima. Para o diagnostico, foram levadas em consideração algumas condições físicas e de saúde dessas mulheres como: o número de gestações que a paciente teve, seu IMC, nível de insulina, idade, pressão arterial, espessura da pele e *Diabetes Pedigree Function* (DPF) que é dado pelo conjunto de valores que medem a tendência ao desenvolvimento de diabetes com base nas características genéticas do indivíduo. Essas são as variáveis preditoras.

Os atributos utilizados referente as mulheres que fizeram parte da pesquisa, em ordem crescente da disposição das colunas são:

Pregnancies: número de gestações que teve

Glucose: nível de glicose

BloodPressure: pressão arterial

SkinThickness: espessura da pele

Insulin: nível de insulina

BMI: IMC (índice de massa corporal)

DiabetesPedigreeFunction: conjunto de valores que indicam tendência ao desenvolvimento de diabetes

Age: idade

Outcome: resultado da análise, sobre probabilidade de ter ou não diabetes. 0 se a probabilidade é baixa, de acordo com a combinação de atributos e 1, se a probabilidade é alta levando em consideração a combinação das mesmas métricas (desfecho).

4. Aperfeiçoamento dos dados

Temos algumas possibilidades para aperfeiçoar os dados em questão, para que o algoritmo tenha um melhor desempenho e resultado. Os escolhidos foram Exclusão de alguns atributos e Tratamento de dados faltantes.

4.1. Exclusão de alguns atributos

Nesse caso, foi excluída a coluna *SkinThickness,* que se refere a espessura da pele das mulheres pesquisadas. Determinei como critério menos relevante, em relação aos outros para o desfecho.

4.2. Tratamento de dados faltantes

No tratamento de dados faltantes, tratei como dado faltante onde na coluna *Insulin* apresentou valor zero, descartando essas linhas para um resultado melhor, por levar em consideração apenas onde essa informação foi apresentada de forma relevante, pois considero que os dados sobre a insulina de cada mulher, seja um fator importante para determinar a probabilidade de diabetes.

5. Desenvolvimento

5.1. Implementação em Python e resultados sem tratamento de dados

Abaixo o código referente a cada algoritmo de aprendizado de máquina apresentados nos itens anteriores.

Foi importada a biblioteca Panda, que permite análise e manipulação de dados, e através da importação da biblioteca **scikit**-learn (sklearn) que chamamos as funções específicas de cada algoritmo, assim como as funções de treinamento e teste, e acurácia.

```
1  import pandas as pd
2  from sklearn.linear_model import LogisticRegression
3  from sklearn.ensemble import RandomForestClassifier
4  from sklearn.neighbors import KNeighborsClassifier
5  from sklearn.model_selection import train_test_split
6  from sklearn.metrics import accuracy_score
```

Através da biblioteca Panda, foi feita a leitura do arquivo que contém o banco de dados diabetes.csv, já discriminando o nome de cada coluna presente.

```
8  # carrega dados do arquivo csv
9  data = pd.read_csv('diabetes.csv', encoding='utf-8', delimiter=',',usecols=['Pregnancies', 'Glucose', 'BloodPressure',
10                                                                            'SkinThickness', 'Insulin', 'BMI',
11                                                                           'DiabetesPedigreeFunction', 'Age','Outcome'])
12
13 print(data.head())
```

Foram printadas as primeiras 5 linhas, para verificar se o banco foi carregado corretamente.

```
PROBLEMAS   SAÍDA   TERMINAL   JUPYTER   CONSOLE DE DEPURAÇÃO

   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72             35        0  33.6                     0.627   50        1
1            1       85             66             29        0  26.6                     0.351   31        0
2            8      183             64              0        0  23.3                     0.672   32        1
3            1       89             66             23       94  28.1                     0.167   21        0
4            0      137             40             35      168  43.1                     2.288   33        1
```

As variáveis preditoras foram separadas da variável de desfecho e armazenadas, respectivamente em X e y. Após essa separação, os dados foram divididos em dados X e y de treinamento e X e y de teste, utilizando 30% dos dados para teste e 70% para treinamento, conforme solicitado no trabalho.

```
15  # separa as variáveis preditoras do desfecho
16  y = data['Outcome']
17  # todas as colunas, exceto a Outcome e grava em x
18  X = data.drop('Outcome', axis = 1)
19
20  # test_size 30% 0.3 30% dos valores foram utilizados para teste
21  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Com os dados devidamente separados, eles são submetidos as funções específicas de cada algoritmo de aprendizado.

O modelo resultante de cada algoritmo é avaliado e comparado com os dados de teste e então é calculada a acurácia.

```
41  # avalia o modelo de acordo com os dados de teste
42  yhat1 = model1.predict(X_test)
43  yhat2 = model2.predict(X_test)
44  yhat3 = model3.predict(X_test)
45  yhat4 = model4.predict(X_test)
46
47  # avalia a precisão comparando o modelo criado yhat, com o conjunto de dados de teste y_test
48  accuracy1 = accuracy_score(y_test, yhat1)
49  print('Accuracy LogisticRegression: %.3f' % (accuracy1*100))
50
51  accuracy2 = accuracy_score(y_test, yhat2)
52  print('Accuracy RandomForestClassifier: %.3f' % (accuracy2*100))
53
54  accuracy3 = accuracy_score(y_test, yhat3)
55  print('Accuracy KNeighborsClassifie: %.3f' % (accuracy3*100))
56
57  accuracy4 = accuracy_score(y_test, yhat4)
58  print('Accuracy SVM: %.3f' % (accuracy4*100))
```

Os resultados obtidos através desse algoritmo, utilizando o banco sem nenhum tratamento prévio dos dados foram em média:

```
Accuracy LogisticRegression: 77.92
Accuracy RandomForestClassifier: 75.76
Accuracy KNeighborsClassifier: 70.13
```

5.2. Implementação em Python e resultados com tratamento de dados

Conforme citado no ítem 4, as técnicas de aperfeiçoamento de dados escolhidas foram exclusão de atributos e exclusão de dados faltantes.

5.2.1. Implementação e resultado do tratamento de exclusão de atributos

A exclusão do atributo escolhido, que foi o *SkinThicknes* foi feita com a função data.pop('SkinThickness').

Na imagem abaixo, mostra num primeiro momento a leitura das 5 primeiras linhas do arquivo, sem a exclusão do atributo *SkinThickness*. Logo abaixo, a leitura do arquivo, já excluindo a coluna *SkinThickness*.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

	Pregnancies	Glucose	BloodPressure	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	0	33.6	0.627	50	1
1	1	85	66	0	26.6	0.351	31	0
2	8	183	64	0	23.3	0.672	32	1
3	1	89	66	94	28.1	0.167	21	0
4	0	137	40	168	43.1	2.288	33	1

Ao fazer essa modificação, os resultados em média foram os seguintes:

```
Accuracy LogisticRegression: 78.35
Accuracy RandomForestClassifier: 76.62
Accuracy KNeighborsClassifier: 72.29
```

Já apresentando um aumento em torno de 2% na acurácia de cada modelo.

5.2.2. Implementação e resultado do tratamento de dados faltantes

A coluna que mais apresentou dados com valor 0, foi a *Insulin,* por isso a escolhida para ser parâmetro e excluir as linhas onde, nessa coluna, o valor fosse 0 a fim de melhorar acurácia. Essa operação foi feita da seguinte forma:

```
18    indexNames = data[data['Insulin'] == 0].index
19    data.drop(indexNames , inplace=True)
```

A imagem abaixo, mostra num primeiro momento a leitura das 5 primeiras linhas do arquivo, sem o tratamento dos dados faltantes. Logo abaixo, a leitura do arquivo, já excluindo as linhas onde na coluna *Insulin,* o valor era 0.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
6	3	78	50	32	88	31.0	0.248	26	1
8	2	197	70	45	543	30.5	0.158	53	1
13	1	189	60	23	846	30.1	0.398	59	1

Para esse tratamento de dados, os resultados foram em média:

```
Accuracy LogisticRegression: 78.15
Accuracy RandomForestClassifier: 76.47
Accuracy KNeighborsClassifier: 74.79
```

Apresentando um comportamento semelhante ao anterior, aumentando em torno de 2% a acurácia.

5.2.3. Implementação e resultado da combinação do tratamento de dados faltantes e da exclusão de atributos

A implementação se deu conforme apresentado nos itens 5.2.1 e 5.2.2. Na imagem abaixo pode-se observar as 5 primeiras linhas do arquivo sem tratamento algum, e depois, as 5 primeiras linhas com os dois tratamentos aplicados:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

	Pregnancies	Glucose	BloodPressure	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
3	1	89	66	94	28.1	0.167	21	0
4	0	137	40	168	43.1	2.288	33	1
6	3	78	50	88	31.0	0.248	26	1
8	2	197	70	543	30.5	0.158	53	1
13	1	189	60	846	30.1	0.398	59	1

Os resultados do algoritmo, com os dois tratamentos aplicados foram em média os seguintes:

```
Accuracy LogisticRegression: 79.83
Accuracy RandomForestClassifier: 78.99
Accuracy KNeighborsClassifier: 78.15
```

Apresentando um resultado ainda melhor que os dados sem tratamento, assim como os tratamentos aplicados individualmente.

6. Avaliação dos resultados e respostas às perguntas propostas

Os dados apresentados como resultados são sempre referidos como “resultados em média” porque a cada execução do código, o algoritmo determina um grupo de dados distinto e isso influencia no resultado final. Mesmo com esse porém, é possível tirar conclusões claras de qual algoritmo apresentou melhor desempenho, e que o tratamento de dados influencia numa acurácia maior.

De forma geral, o algoritmo que apresentou melhor desempenho foi o *LogisticRegression,* mesmo quando sem tratamento de dados, quanto nos tratamentos individuais e na combinação dos dois aperfeiçoamentos.

O pré-processamento ajudou, porque ao excluir a coluna menos relevante, seriam menos dados analisados e ao remover as linhas em que um atributo importante estava referido como 0, se tem um resultado mais fiel ao esperado.

7. Conclusão

A acurácia pode variar conforme as execuções, assim como varia de acordo a porcentagem de dados de treinamento e teste, que podem ser arbitradas conforme interesse do usuário.

O *LogisticRegression* e o *Random Forest Classifier* apresentaram um desempenho melhor que o KNN em mesmas condições e especificamente nesse tipo de dados tratados, o que não exclui a possibilidade desses algoritmos se comportarem de forma diferente quanto as acurácias, se aplicados a dados binários, por exemplo.

A escolha do algoritmo mais adequado depende da aplicação específica, de um modo geral todos os 3 apresentados entregam resultados satisfatórios para problemas o problema proposto.