CHECK-POINT: MODELOS SUPERVISIONADOS

CARLOS FREDERICO CARVALHEIRA MELLO

JOÃO VITOR DEON

1 Objetivos e Introdução

Criar um modelo de classificação de diagnósticos da doença Diabetes Mellitus, usando uma base de dados onde a priori não sabemos o que cada *feature* realmente representa.

Com cada vez mais dados de exames sendo disponíveis, a informação contida nestes pode ajudar em futuros diagnósticos. A automação na classificação de diagnósticos tem como principal objetivo agilizar o processo de diagnosticar um pacientes, padronizar a forma como que os exames são avaliados e auxiliar o médico na hora de tomadas de decisão, resultando em diagnósticos mais precisos e com um custo menor.

2 DataSet

Os dados foram fornecidos pelo Professor Antonio Carlos Gay Thomé do Departamento de Informática e Matemática e foi Aplicado com o intuito de fazer um estudo estatístico sobre cada variável e fazer a diminuição de dimensionalidade do mesmo.

O dataset é formado de um total de dezoito colunas, destas dezessete identificadas sequencialmente pelas letras do alfabeto de “A” até “P” são consideradas como dados de entrada com diferentes tipos e dimensões e ainda uma última com o nome de “Diagnóstico” que contém apenas dois valores representando as duas classes de saída, 2 para positivo e 1 para negativo. Um exemplo a figura 1.0 pode ser visto mostrando um pouco melhor como é estruturado o conjunto de dados, foi usado para essa demonstração a função describe da biblioteca *pandas,* onde podem ser facilmente vistos valores como contagem de ocorrência dos valores, média, desvio padrão, valores mínimos e máximos, etc.

2.1 Redução de dimensão

2.1.1 Análise

Na primeira análise feita, foi importado a tabela de forma crua, ou seja sem nenhuma alteração na tabela. Em seguida foi feito todo um trabalho de limpeza e conversão de alguns valores incorretos.

A primeira tomada de decisão para remoção de algumas colunas do dataset foi a distribuição de seus valores. Como em teoria os valores que estavam no dataset foram obtidos de exames médicos de uma amostra da população, existe uma certa tendência dos valores corretos gerarem uma curva Gaussiana como na Imagem 1.1 e não como a curva representada na imagem 1.2

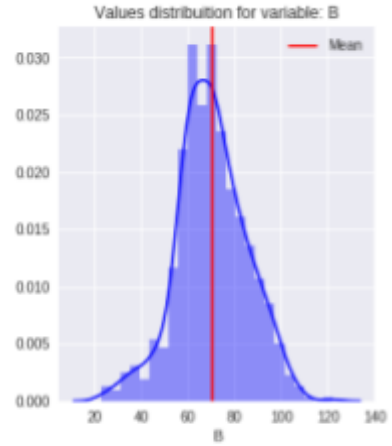


Figura 1.1

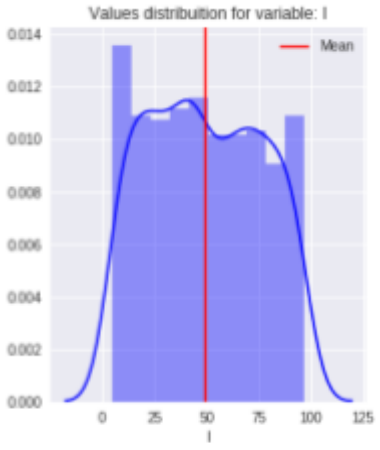


Figura 1.2

Então foram deixadas apenas as colunas que de alguma forma a distribuição dos seus valores lembravam o mínimo possível a de uma curva Gaussiana.

Logo após foram feitas as remoções visando remover a redundância de valores entre colunas, foi feita assim uma análise das correlações lineares entre cada uma das das variáveis do dataset e todas estas que tiveram uma correlação entre si próximo a 1 foram removidas. Como pode ser visto na figura 1.3 um mapa de calor foi feito para ilustrar as correlações entre cada variável do dataset, note que quanto mais escura a cor, maior a correlação da variável.

Ao final foram removidas as seguintes colunas H, I, K, L, N, O e P. Totalizando sete colunas a menos do dataset, o estado desse conjunto de dados foi salvo para ser usado futuramente na análise de cada componente.

2.1.2 Análise de principal componente (PCA)

Utilizando o dataset resultado da última análise, já com menos features o pca foi usado para reduzir ainda mais a dimensão do dataset em apenas alguns componentes.

O Número final de componentes foi escolhido com base na verificação explanatória de um somatório da representatividade de variância de cada componente. Podemos observar na imagem 1.4 que com apenas 4 componentes finais usando a transformação ortogonal podemos praticamente representar cerca de 99% dos dados do dataset.

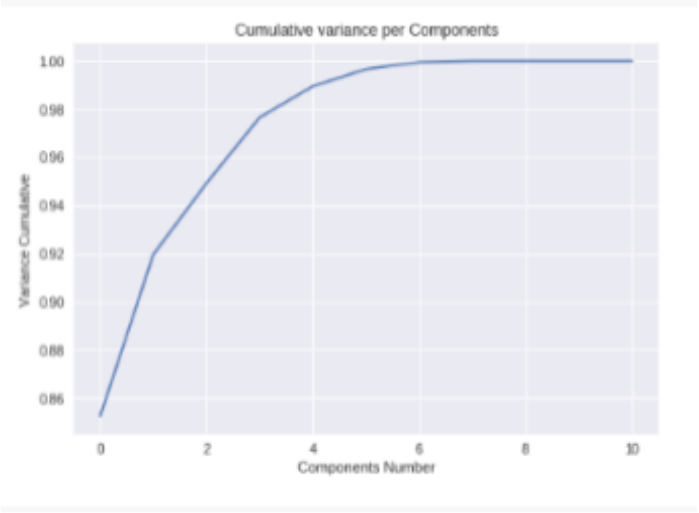


Figura 1.4

Então ao final da transformação feita usando o pca com o número de componentes igual a 4, acabamos com um dataset com apenas 4 features que representam cerca de 99% de representatividade dos dados finais, mais ainda a coluna de diagnóstico, totalizando cerca de 5 colunas finais.

3 Experimentos dos Modelos

3.1 K-NN

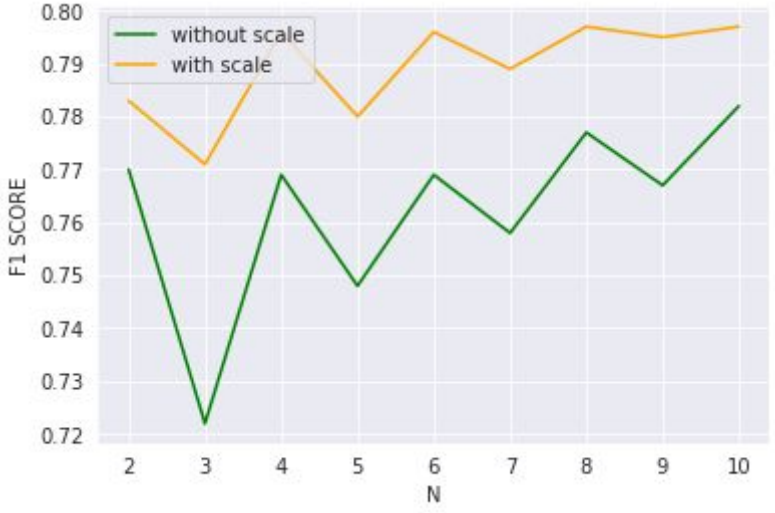
3.1.1 Metodologia dos testes

O modelo foi implementado usando o algoritmo ‘KNeighborsClassifier’ da biblioteca de machine learning ‘sklearn’.

Nesse modelo foi utilizado um método de força bruta, onde foram testados diversos parâmetros k diferentes para saber qual o que obtém melhores métricas de desempenho.

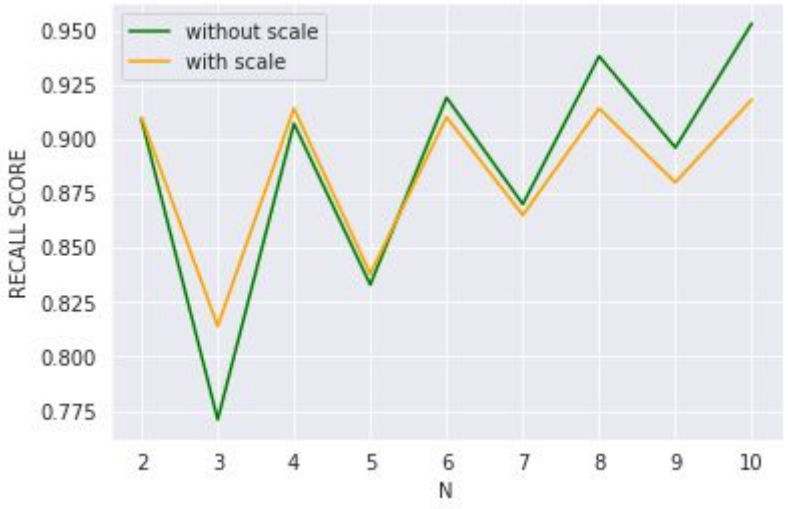
Os testes foram feitos usando a função ‘StratifiedShuffleSplit’, que além de usar validação cruzada, faz o embaralhamento dos conjuntos de treino e teste em cada cruzamento, o que permite testar cada modelo sempre de diversas maneiras e diminuir drasticamente as chances de enviesamento do classificador.

3.1.2 Resultados



Como vemos na figura acima, testamos tanto os dados crus, como os dados escalonados usando a função ‘MinMaxScaler’(que deixa os valores entre 0 e 1).

Percebemos que com a utilização do escalonamento de valores nosso modelo se saiu melhor, considerando a métrica F1 SCORE. Também podemos perceber que a utilização de valores ímpares para k(os vizinhos mais próximos) têm uma tendência de redução de métrica.



Se observarmos a métrica recall, o escalonamento foi pouco eficiente, porém conseguimos um ótimo valor para esta métrica, que para nós é a mais importante.

De forma geral, o modelo obteve estes resultados:

Accuracy: 0.654
Precision: 0.663
Recall: 0.953
F1-measure: 0.782

3.2 Decision Tree

3.2.1 Metodologia dos testes

Aqui também usaremos a biblioteca ‘sklearn’, e mais especificamente o método ‘DecisionTreeClassifier’.

Os testes foram feitos usando a função ‘GridSearchCV’ que promove validação cruzada e aqui foram usados 10 k-folds. Essa função nos permite testar diversas combinações de parâmetros e podemos chamar um método para nos retornar nossos melhores parâmetros, que vamos usar futuramente para fazer tunning e tentar melhorar o desempenho de nosso modelo.

3.2.2 Resultados

Antes de fazermos o tunning dos parâmetros, nos obtivemos os seguintes resultados:

Accuracy: 0.697
Precision: 0.762
Recall: 0.778
F1-measure: 0.77

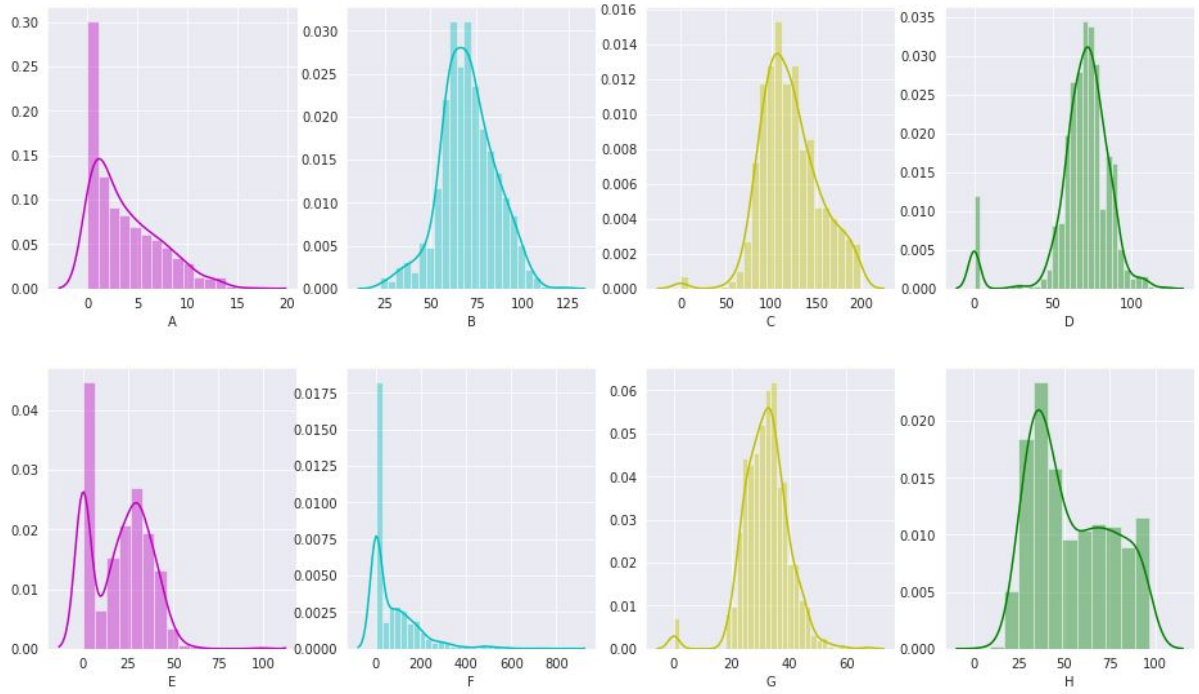
Temos valores medianos que podem ser considerados bons mas se olharmos para o recall, nosso modelo de KNN teve uma performance bem melhor.

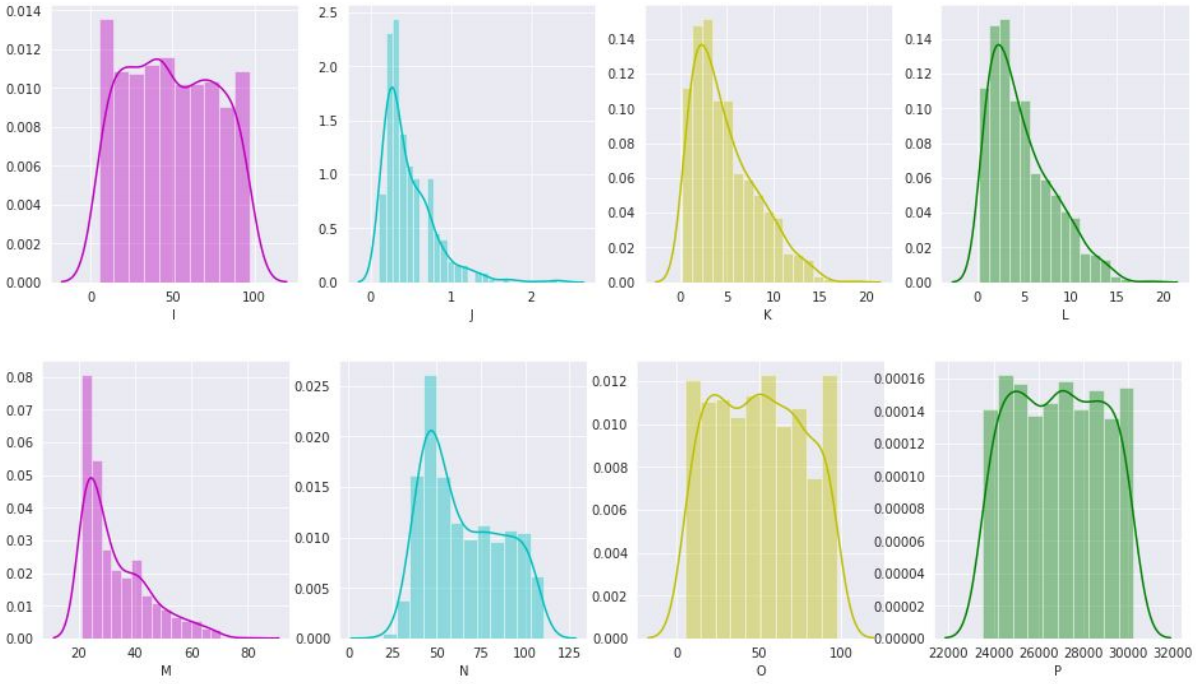
3.3 Gaussian Naive Bayes

3.3.1 Metodologia dos testes

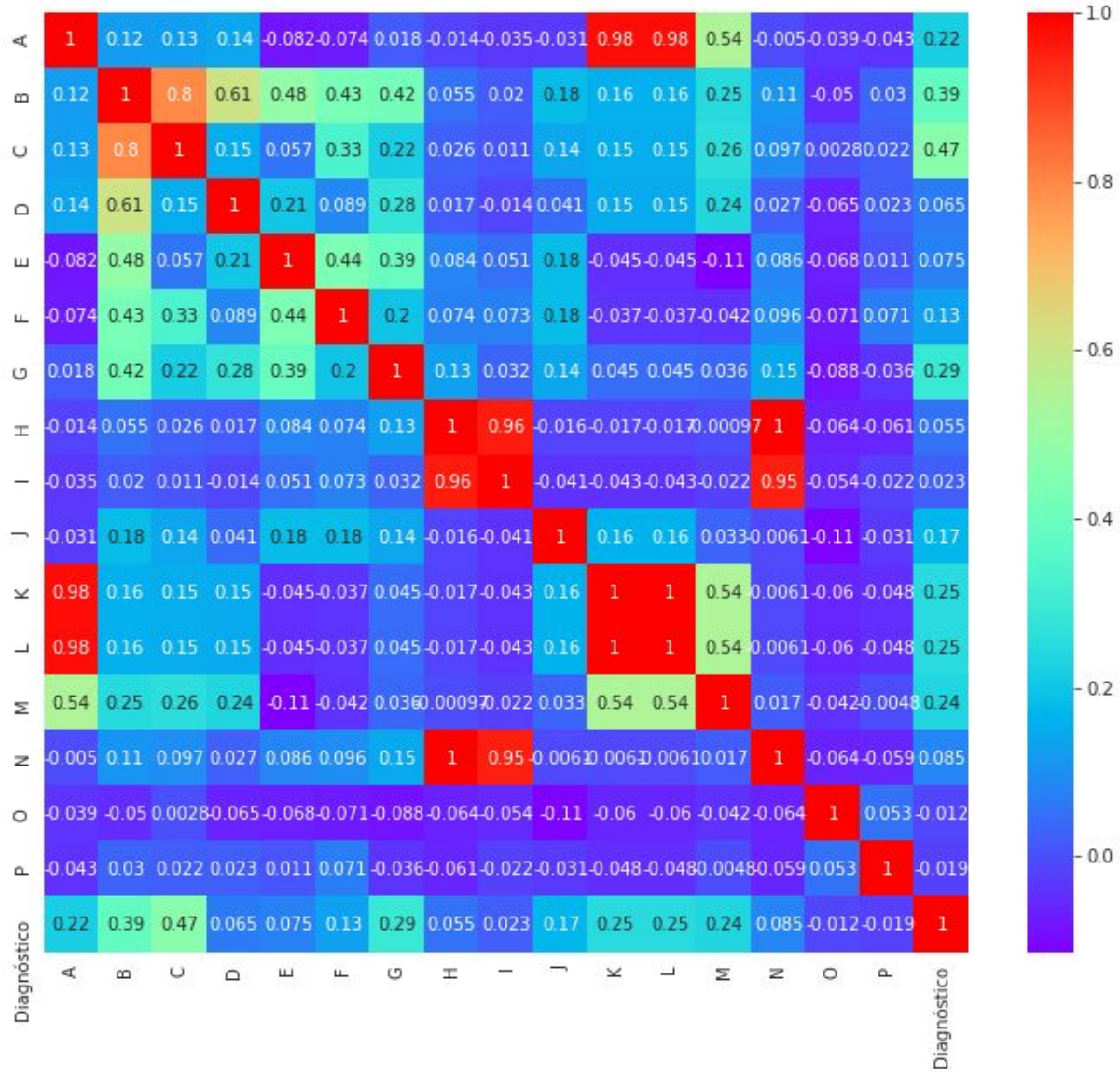
Para este modelo, foi utilizado o algoritmo ‘GaussianNB’ da biblioteca ‘sklearn’. Aqui, iremos focar na análise visual das features do nosso problema para tentar entendê-las melhor. Esse algoritmo não permite a mudança de parâmetros e para melhorar seu desempenho devemos modificar diretamente as variáveis,se for pertinente para o problema.

3.3.2 Resultados





Olhando para as distribuições das colunas acima, percebemos que a única coluna que parece seguir uma distribuição normal é a coluna B.



Este gráfico acima representa as correlações de todas as colunas do nosso dataset. De cara, percebemos que não há nenhuma coluna que explique de forma clara o diagnóstico da doença de diabetes. Depois percebemos uma colinearidade de algumas features, como as colunas K e L com a coluna A; H e I com a coluna N; a coluna C com a B; e a coluna L com a K que possuem o maior valor(1) de correlação de pearson.

3.4 MLP

3.4.1 Metodologia dos testes

A rede neural foi toda implementada em Python 3, usando a biblioteca 'Tensorflow' para fazer os cálculos da rede e a Keras para importar já alguns modelos de rede neural . A rede foi montada em uma arquitetura sequencial, contendo duas camadas internas com um tamanho inicial igual a três segundo a fórmula de ((números de entradas + número de labels)/ 2). Os pesos de cada camada foram inicializados seguindo uma distribuição randômica normal. E por fim os bias foram inicializados com valores iguais a um.

Os conjuntos de treino e teste foram divididos com tamanhos iguais e com representatividades iguais das classes e para os testes de iteração eu optei por não usar nenhum tipo de k-fold.

As duas camadas tiveram a mesma função de ativação 'relu' e a saída possui uma camada com a ativação Sigmóide, onde essa responde com um valor contínuo entre 0 e 1 e dependendo da escolha de um threshold que as classes serão atingidas com esses valores.

Quanto as iterações dos testes todos foram feitas em EXCEÇÃO as 10 000. Foram feitas no total, Taxa de aprendizado: 0.1, 0.001, 0.0001; Número de iterações: 100, 1000; e por fim tamanho das camadas: 3, 12, 24.

O otimizador usado foi o 'Stochastic Gradient Descendent', como o momentum tinha que ser fixo em 0.8, eu supus que esse seria a função de otimização mais apropriada e a função de perda foi Mean Square Error.

3.4.2 Resultados

Foi colocado em um vetor com uma tupla, contendo respectivamente o conjunto de parâmetros usados(Iterações, taxa de aprendizado e multiplicador do tamanho da camada) e a acurácia do modelo. Idealmente eu gostaria de ter colocado o tempo de execução para cada conjunto de parâmetros, infelizmente por erro de implementação acabei por não fazer isso. O resultado foi o seguinte:

```
[([1000, 0.1, 1], 75.98425149917603),
 ([100, 0.1, 1], 74.40944910049438),
 ([1000, 0.1, 4], 74.40944910049438),
 ([1000, 0.001, 1], 74.40944910049438),
```

```
([100, 0.001, 4], 74.01574850082397),
([100, 0.001, 8], 74.01574850082397),
([1000, 0.0001, 8], 74.01574850082397),
([100, 0.001, 1], 73.62204790115356),
([100, 0.0001, 8], 73.62204790115356),
([1000, 0.0001, 1], 73.62204790115356),
([1000, 0.0001, 4], 73.62204790115356),
([100, 0.0001, 1], 72.83464670181274),
([100, 0.0001, 4], 72.83464670181274),
([1000, 0.001, 8], 72.83464670181274),
([1000, 0.001, 4], 72.44094610214233),
([100, 0.1, 4], 71.2598443031311),
([100, 0.1, 8], 70.07874250411987),
([1000, 0.1, 8], 66.92913174629211)]
```

Podemos observar com clareza que com mais iterações temos uma acurácia um pouco melhor, mas executar as 1000 iterações de uma rede neural demora excessivamente e por isso optei no modelo final por usar apenas 100. Quanto a taxa de aprendizado usada optei por usar a 0.01 que era um valor intermediário entre os dois primeiros e por fim usei o valor de 12 como tamanho que também é um intermediário.

No teste final usei uma distribuição diferente dos conjuntos de teste e treino com uma divisão desses usando um 'StratifiedKFold' com um tamanho de dobra de 10. E por fim usando os seguintes parâmetros:

Otimizador: SGD, learning rate = 0.01 e momentum 0.8.
Loss: Mean Square Error.
Model: Sequential, 2 hidden layers com 'relu' e saída com 'sigmoid'.
k-fold: 10 folds.
Epochs: 100
Ao final da execução desse teste tive uma acurácia de 73,4%.

4 Melhorias aos modelos

4.1 Melhorias do KNN

Observando os gráficos do KNN percebemos que o melhor valor de K foi 10, ou seja, o modelo teve melhor desempenho analisando os 10 vizinhos mais próximos de cada ponto de estudo. Quando olhamos para o seu desempenho , sua acurácia deixou a desejar mas o que importa para o nosso caso é a métrica recall. Essa métrica nos diz a porcentagem de uma pessoa ser diagnosticada com diabetes dentre as pessoas que realmente tem a doença. Usando K=10 obtivemos um recall de aproximadamente 95%, o que nos indica que entre as

pessoas que realmente tem a doença, apenas 5% não foram diagnosticadas com a doença.

4.2 Melhorias da Decision Tree

Como discutido no tópico de testes, usaremos uma função para tunar nossos parâmetros da nossa árvore.

Utilizamos as seguintes combinações de parâmetros:

```
{
        'criterion': ['gini', 'entropy'],
        'max_depth' : [2,4,6,8,10] ,
        'min_samples_split': [2,4,6,8,10],
        'min_samples_leaf': [2,4,6,8,10]
                                                }
```

Os melhores parâmetros encontrados foram :

```
{
        'criterion':'entropy',
        'max_depth': 6,
        'min_samples_leaf': 10,
        'min_samples_split': 2
                                        }
```

Após aplicar esses novos parâmetros no classificador obtivemos os seguintes resultados :

Accuracy: 0.722
Precision: 0.791
Recall: 0.787
F1-measure: 0.786

Comparando estas métricas ao resultado sem tunning, tivemos uma pequena melhoria em todas as métricas, porém estamos em busca de uma melhor recall.

4.3 Melhorias da Gaussian Naive Bayes

Como não podemos tunar os parâmetros desse classificador, visto que ele trabalha apenas com as probabilidades iniciais e a posteriori do nosso dataset, iremos apresentar o seu desempenho na linha abaixo:

Accuracy 0.74
Precision 0.799
Recall 0.803
F1-measure 0.801

Novamente, temos bons valores, inclusives muito próximos ao Decision Tree já com tunning de parâmetros, porém estamos a procura de melhor recall, com resultados próximos ou superiores ao do nosso modelo KNN.

4.4 Melhorias da MLP

Para melhorar a rede o primeiro passo que eu fiz foi trocar a função de otimização. Existe hoje um '*hype*' quando ao otimizador chamado por Adaptive Momentum(Adam), que nada mais é do que uma atualização da Stochastic Gradient apresentados por Diederik Kingma e o Jimmy Ba. O algoritmo além de taxa de aprendizado convencional que agora é conhecida como alpha, possui também dois parâmetros de decaimento exponencial o beta 1 e o beta 2.

Por ser um método adaptativo o Adam altera a taxa de aprendizado para cada parâmetro de entrada da rede, além disso ele usa estimadores betas para poder adaptar o momentum para cada peso da rede, gerando uma taxa de aprendizado mais balanceada e adaptativa para cada entrada da rede, ao final pode-se gerar um resultado de acurácia bem mais satisfatório do que o convencional Stochastic Gradient.

Além disso, valores um pouco maiores para as camadas cerca de 32 neurônios por layer, um número menor ainda de iterações cerca de 50 epochs já é o suficiente para o projeto, e por fim por conta do banco de dados ser muito pequeno eu adicionaria ainda mais dobras ao K-fold para obter um conjunto de treino e teste ainda maior.

5 Conclusão

Os métodos de aprendizado supervisionados oferecem uma boa solução quando temos as respostas(targets) do conjunto de dados, ao contrário de um método não supervisionado que o próprio algoritmo tenta agrupar os valores e chegar a um padrão desconhecido. É importante frisar que nenhum dos 4

classificadores testados sofreram overffiting, pois utilizamos as técnicas ‘StratifiedShuffleSplit’, ‘StratifiedKFold’ e ‘GridSearchCV’, que além de usarem validação cruzada, fazem o embaralhamento dos conjuntos de treino e teste em cada cruzamento, o que permite testar cada modelo sempre de diversas maneiras e diminuir drasticamente as chances de enviesamento do classificador.

Nesse projeto, ficou claro a nossa preocupação com a métrica recall em específico, isso se dá a sua importância para nosso problema em específico de corretamente termos que classificar uma pessoa diabética com esta doença.

Quanto menor o recall, maior a proporção de pessoas que realmente tem a doença mas não foram classificadas como tendo a mesma, o que é bastante grave.

Tendo isso em mente, mesmo com o KNN não apresentando boas métricas em geral, seu recall foi de aproximadamente 95%, o que indica que apenas 5% das pessoas que realmente tinham diabetes deixaram de ser diagnosticadas positivamente com a doença. É um ótimo valor, considerando o tamanho pequeno do nosso conjunto de dados. Logo, dentre os 4 classificadores testados ao longo deste projeto, o que melhor desempenha a nossa principal preocupação foi o KNN.

6 Referências

Amato, F., López, A., Peña-Méndez, E.M., Vaňhara, P., Hampl, A. and Havel, J., 2013. Artificial neural networks in medical diagnosis.

Šter, B. and Dobnikar, A., 1996. Neural networks in medical diagnosis: Comparison with other methods. In International conference on engineering applications of neural networks (pp. 427-30).

Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.