**POS IESB 2020**

**Disciplina: Fundamentos de IA**

**Alunos: Eduardo Gomes Piza**

**Valter Takechi Hada**

**Data: 27/08/2020**

**Trabalho 01 – Previsão de Risco de Diabetes em Estágio Inicial**

A partir do dataset disponibilizado vamos treinar uma rede neural PMC visando classificar a partir de um grupo de dados informados por um paciente se o mesmo encontra-se em estágio inicial de diabetes.

O dataset possui 519 registros, cada um com 17 atributos. Após o pré processamento (os procedimentos foram realizados pelo código jupyter de entrega deste trabalho) os dados ficaram com o seguinte formato:

. Idade;

. Sexo (0=Feminino e 1=Masculino);

. Poliuria (0=Não e 1=Sim);

. Polidipsia (0=Não e 1=Sim);

. Perda súbita de peso (0=Não e 1=Sim);

. Fraqueza (0=Não e 1=Sim);

. Polifagia (0=Não e 1=Sim);

. Herpes genital (0=Não e 1=Sim);

. Visão borrada (0=Não e 1=Sim);

. Coceira (0=Não e 1=Sim);

. Irritabilidade (0=Não e 1=Sim);

. Cicatrização lenta (0=Não e 1=Sim);

. Paralisia parcial (0=Não e 1=Sim);

. Rigidez muscular (0=Não e 1=Sim);

. Calvicie (0=Não e 1=Sim);

. Obesidade (0=Não e 1=Sim);

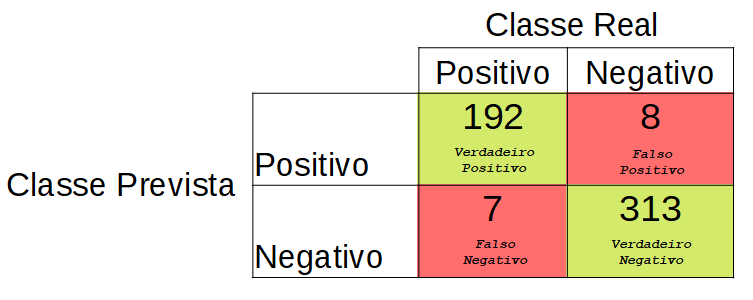
. Diagnóstico (0=Negativo e 1=Positivo).

Todos os dados foram usados para treinamento e testes, mas separados em grupos de 90% e 10% (respectivamente) em cada um dos ciclos de treino e predição.

A arquitetura de rede utilizada na execução que apresentou o melhor resultado de acurácia e precisão (como consta no código jupyter) possuía 2 camadas com 128 neurônios cada. Utilizamos o algoritmo otimizador ‘*lbfgs*’ (método quasi-Newton) em virtude da quantidade de dados de input ser pequena (apenas algumas centenas de registros) e, conforme documentação do scikit-learn, este é o algoritmo que pode apresentar melhor performance e convergir mais rapidamente. O mesmo critério foi usado para a utilização de algoritmo de função hiperbólica na função de ativação para a camada oculta(*activation='tanh'*)visto que estas funções comumente performam melhor que as funções sigmoides.

Diversos outros parâmetros foram testados mas como decidimos utilizar o ‘*lbfgs*’ como algoritmo otimizador (em nossa versão final) alguns dos outros parâmetros não se aplicavam.

Segue a matriz de confusão resultante dos testes:



A partir deste ponto do documento vamos registrar o resultado de **algumas** das execuções treino/teste realizadas, indicando os parâmetros usados e a acurácia/precisão obtidos até chegarmos na melhor configuração encontrada (penúltima linha da tabela abaixo, ***“Configuração 09”,*** que contém justamente os parâmetros encontrados no código jupyter):

|  |  |
| --- | --- |
| **Parâmetros** | **Resultado e Gráficos** |
| **Configuração 01:**  **epocas=**100  **hidden\_layer\_sizes=**(128,128,)  **activation=**'relu'  **alpha=**1e-4  **solver=**'sgd'  **tol=**1e-5  **learning\_rate\_init=**.001  **k =** 10 #folds  Observação: tempo de treinamento da MLP bastante curto (inferior a 10 segundos) | Tempo médio de treinamento MLP com 10 kfold 5.2203125  Médias das Validações com 10 folds  Acurácia\_mlp: 84.61538461538461  Precisão\_mlp: 89.5  **gráfico loss\_value:** |
| **Configuração 02:**  **epocas=**100  **hidden\_layer\_sizes=**(256,)  **activation=**'relu'  **alpha=**1e-4  **solver=**'sgd'  **tol=**1e-5  **learning\_rate\_init=**.001  **k =** 10 #folds  Observação: tempo de treinamento da MLP bastante curto (inferior a 10 segundos) | Tempo médio de treinamento MLP com 10 kfold 4.503125  Médias das Validações com 10 folds  Acurácia\_mlp: 81.34615384615387  Precisão\_mlp: 55.0  **gráfico loss\_value:** |
| **Configuração 03:**  **epocas=**100  **hidden\_layer\_sizes=**(128,128,)  **activation=**'relu'  **alpha=**1e-4  **solver=**'lbfgs'  **tol=**1e-5  **learning\_rate\_init=**.001  **k =** 10 #folds  Observação: tempo de treinamento da MLP bastante curto (inferior a 10 segundos) | Tempo médio de treinamento MLP com 10 kfold 7.4328125  Médias das Validações com 10 folds  Acurácia\_mlp: 92.49999999999999  Precisão\_mlp: 91.5  **gráfico loss\_value:**  apenas o otimizador sgd possui função automática para loss function |
| **Configuração 04:**  **epocas=**100  **hidden\_layer\_sizes=**(256,)  **activation=**'relu'  **alpha=**1e-4  **solver=**'lbfgs'  **tol=**1e-5  **learning\_rate\_init=**.001  **k =** 10 #folds  Observação: tempo de treinamento da MLP bastante curto (inferior a 10 segundos) | Tempo médio de treinamento MLP com 10 kfold 4.85625  Médias das Validações com 10 folds  Acurácia\_mlp: 92.11538461538461  Precisão\_mlp: 91.0  **gráfico loss\_value:**  apenas o otimizador sgd possui função automática para loss function |
| **Configuração 05:**  **epocas=**100  **hidden\_layer\_sizes=**(128,128,)  **activation=**'relu'  **alpha=**1e-4  **solver=**'lbfgs'  **tol=**1e-5  **learning\_rate\_init=**.0001  **k =** 10 #folds  Observação: tempo de treinamento da MLP bastante curto (inferior a 10 segundos) | Tempo médio de treinamento MLP com 10 kfold 7.678125  Médias das Validações com 10 folds  Acurácia\_mlp: 92.49999999999999  Precisão\_mlp: 91.5  **gráfico loss\_value:**  apenas o otimizador sgd possui função automática para loss function |
| **Configuração 06:**  **epocas=**100  **hidden\_layer\_sizes=**(128,128,)  **activation=**'relu'  **alpha=**1e-4  **solver=**'sgd'  **tol=**1e-5  **learning\_rate\_init=**.0001  **k =** 10 #folds  Observação: tempo de treinamento da MLP bastante curto (inferior a 10 segundos). Estes parâmetros apresentaram o **pior resultado** dentre todas as 42 configurações testadas, sendo a principal di-ferença entre os parâmetros a redução da “learning\_rate”. | Tempo médio de treinamento MLP com 10 kfold 5.9421875  Médias das Validações com 10 folds  **Acurácia\_mlp: 62.6923076923077**  **Precisão\_mlp: 3.5**  **gráfico loss\_value:** |
| **Configuração 07:**  **epocas=**100  **hidden\_layer\_sizes=**(128,128,)  **activation=**'tanh'  **alpha=**1e-4  **solver=**'lbfgs'  **tol=**1e-5  **learning\_rate\_init=**.001  **k =** 10 #folds  Observação: tempo de treinamento da MLP bastante curto (inferior a 10 segundos), e obteve o melhor dos resultados com treinamentos de curta duração. Na fase de cross validation era recebido um warning: *ConvergenceWarning: lbfgs failed to converge (status=1)* | Tempo médio de treinamento MLP com 10 kfold 5.9515625  Médias das Validações com 10 folds  **Acurácia\_mlp: 92.88461538461539**  **Precisão\_mlp: 94.0**  **gráfico loss\_value:**  apenas o otimizador sgd possui função automática para loss function |
| **Configuração 08:**  **epocas=**100  **hidden\_layer\_sizes=**(128,128,128,)  **activation=**'tanh'  **alpha=**1e-4  **solver=**'lbfgs'  **tol=**1e-5  **learning\_rate\_init=**.001  **k =** 10 #folds  Observação: tempo de treinamento da MLP bastante curto (inferior a 10 segundos). Na fase de cross validation era recebido um warning: *ConvergenceWarning: lbfgs failed to converge (status=1)* | Tempo médio de treinamento MLP com 10 kfold 9.0328125  Médias das Validações com 10 folds  Acurácia\_mlp: 92.88461538461539  Precisão\_mlp: 93.5  **gráfico loss\_value:**  apenas o otimizador sgd possui a loss function |
| **Configuração 09:**  **epocas=5000**  **hidden\_layer\_sizes=**(128,128,128,)  **activation=**'tanh'  **alpha=**1e-4  **solver=**'lbfgs'  **tol=**1e-5  **learning\_rate\_init=**.001  **k =** 10 #folds  Observação: tempo de treinamento da MLP extremamente lento quando comparado com os demais (177 segundos, cerca de 20 vezes mais “lento” que os demais treina-mentos). Este aumento expressivo no número do *epocas* usadas (principal motivo do aumento do tempo de treinamento) foi neces-sário para solucionar os proble-mas de warning descrito nas con-figurações 07 e 08. A duração da cross validation também foi ampliada na mesma proporção (levando pouco mais de 20 minu- tos).  **Esta execução foi a que apresen-tou a melhor combinação de acu-rácia e precisão!** | Tempo médio de treinamento MLP com 10 kfolds - Tempo: 177.5453125 segundos.  Médias das Validações com 10 folds  **Acurácia\_mlp: 97.11538461538461**  **Precisão\_mlp: 96.0**  **gráfico loss\_value:**  apenas o otimizador sgd possui a loss function |
| **Configuração 10:**  **epocas=8000**  **hidden\_layer\_sizes=**(128,128,128,)  **activation=**'tanh'  **alpha=**1e-4  **solver=**'lbfgs'  **tol=**1e-5  **learning\_rate\_init=**.001  **k =** 10 #folds  Observação: tempo de treinamento da MLP extremamente lento quando comparado com os demais (176 segundos, cerca de 20 vezes mais “lento” que os demais treina-mentos). Este aumento expressivo no número do *epocas* usadas (principal motivo do aumento do tempo de treinamento) foi neces-sário para solucionar os proble-mas de warning descrito nas con-figurações 07 e 08. A duração da cross validation também foi ampliada na mesma proporção (levando pouco mais de 20 minu- tos). Apesar do aumento do núme-ro de épocas (de 60%, para 8000) vemos que resultado da acurácia e precisão foram praticamente os mesmos da execução anterior. | Tempo médio de treinamento MLP com 10 kfolds - Tempo: 176.125 segundos.  Médias das Validações com 10 folds  **Acurácia\_mlp: 97.11538461538461**  **Precisão\_mlp: 95.5**  **gráfico loss\_value:**  apenas o otimizador sgd possui a loss function |

As alterações nos parâmetros que permitiram atingir este alto nível de acurácia e precisão foram guiadas a partir da leitura da documentação encontrada no site [*https://scikit-learn.org/*](https://scikit-learn.org/) (para o MLPClassifier) e na comunidade [*https://stackoverflow.com/*](https://stackoverflow.com/) (para verificar sugestões de solução para os warnings encontrados durante as execuções).

Ficou claro que não necessariamente o aumento de determinados parâmetros trarão um resultado superior (em nossas simulações chegamos a usar redes com 3 camadas de 512 neurônios e os resultados foram inferiores aos obtidos com apenas 2 camadas de 128).

Teste similar foi feito com a redução do parâmetro “learning\_rate\_init” de .001 para .0001 (este foi um testes escolhidos como evidência e destacado na tabela acima, e nesta simulação tivemos o pior resultados dentre as 42 configurações testadas).

Brasília, 27/08/2020.

O arquivo “***Trabalho\_01\_Eduardo\_Valter.ipynb***” e o dataset utilizado no estudo foram disponibilizados no GitHub, conforme links abaixo:

**Código Jupyter:**

*https://github.com/EduPiza/POS\_IESB\_FUND\_IA/blob/master/Trabalho\_01\_Eduardo\_Valter.ipynb*

**Dataset:** *https://github.com/EduPiza/POS\_IESB\_FUND\_IA/blob/master/diabetes\_data\_upload.csv*