UNIVERSIDADE FEDERAL DE PERNAMBUCO - UFPE
CENTRO DE INFORMÁTICA - CIn
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA: APRENDIZAGEM DE MÁQUINA

PROJETO DA DISCIPLINA APRENDIZAGEM DE MÁQUINA

PROFESSOR: Francisco de Assis Tenório de Carvalho

EQUIPE:
Carlos Antônio Alves Júnior (caaj@cin.ufpe.br)
Marcos de Souza Oliveira (mso2@cin.ufpe.br)
Matheus Johann Araújo (mja@cin.ufpe.br)

RECIFE-PE

Agosto de 2021

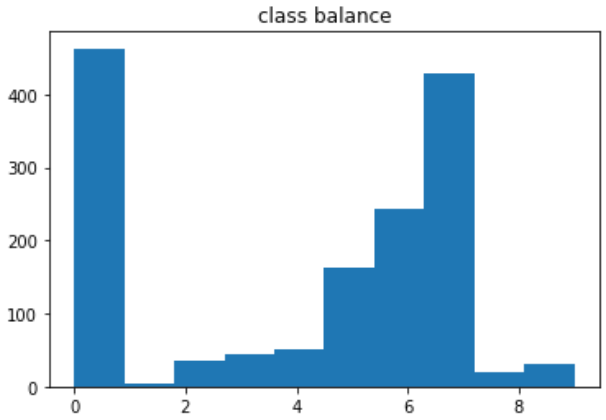
Contextualização

Este projeto consiste em aplicar dois paradigmas de aprendizagem de máquina no conjunto de dados *Yeast Data Set*[1], sendo a primeira etapa a aplicação de uma técnica não-supervisionada [1] e a segunda etapa a aplicação de alguns classificadores para a atividade supervisionada.

O conjunto de dados possui 8 atributos (**mcg, gvh, alm, mit, erl, pox, vac, nuc**), além da variável dependente e 1484 exemplos. Porém antes da aplicação dos métodos é importante realizar uma análise prévia dos dados.

- **Balanceamento das Classes**

  A primeira análise dos dados consiste em verificar a distribuição dos exemplos para cada classe, a figura abaixo ilustra essa proporção.



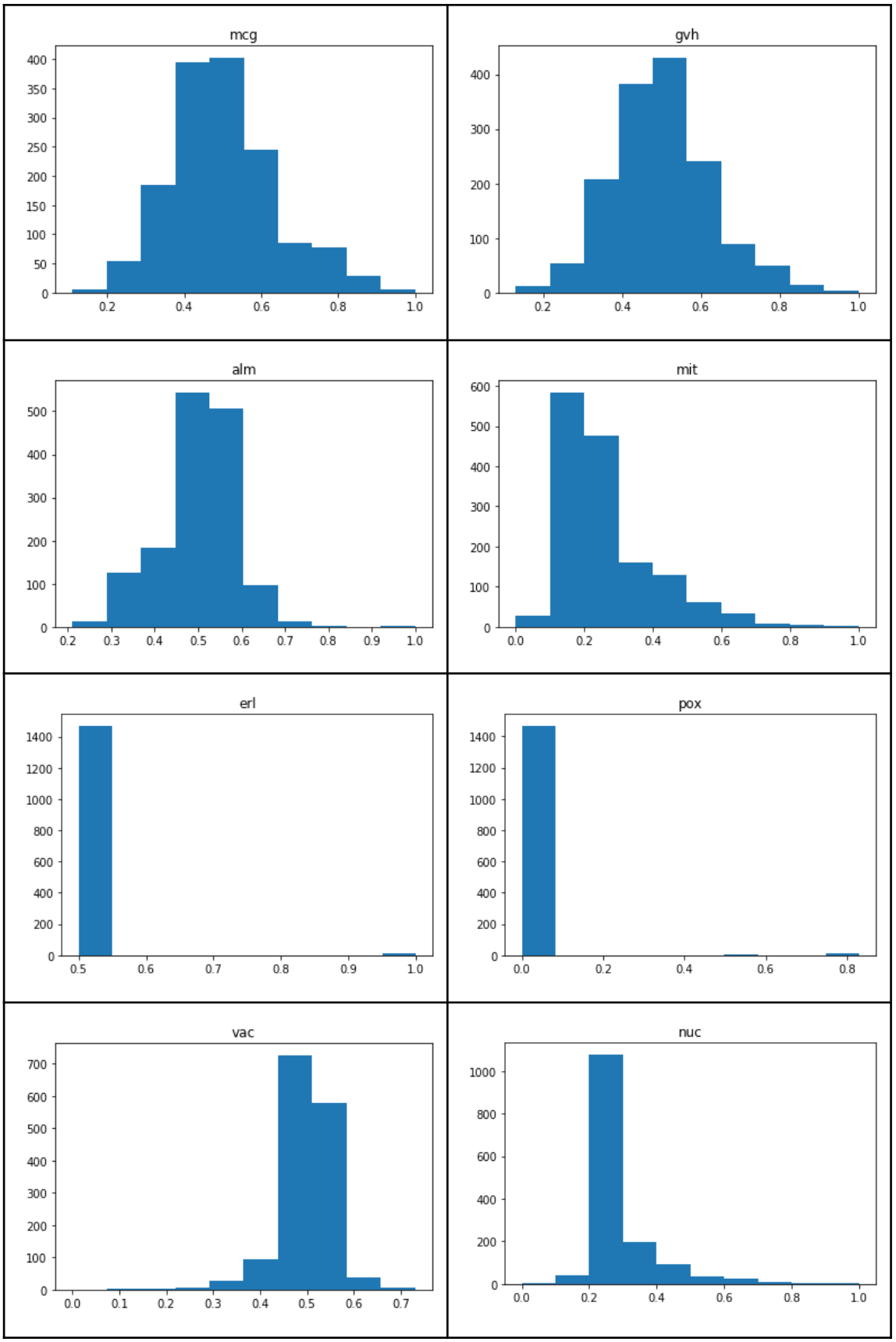
Como observado, a maioria dos exemplos se concentram nas classes 0 e 6, e isto poderá provocar um viés, principalmente, em classificadores que utilizam a probabilidade a priori como um critério de decisão.

Um outro problema relacionado pode ocorrer na estratificação dos exemplos de treinamento e teste, nas classes que possuem um baixo número de exemplos. Desta forma uma alternativa viável seria a aplicação de um *oversampling* nas classes minoritárias.

- **Análise dos atributos**

Em um determinado conjunto de dados, realizar a análise dos atributos torna-se uma importante forma de observar as distribuições para cada feature. Ilustração de cada atributo abaixo:

[1] Disponível em: http://archive.ics.uci.edu/ml/datasets/Yeast



Na figura acima, é possível notar que os atributos **erl** e **pox** possuem valores constantes ao longo de praticamente todos os exemplos. Afim de

simplificar o conjunto de dados, uma solução que pode ser adotada envolve a remoção desses atributos.

Neste trabalho não foi empregado um pré-processamento no conjunto de dados, com o objetivo de avaliar a robustez dos métodos ao lidar com os problemas apresentados.

Parte 1 - Aprendizado Não Supervisionado

A primeira etapa consiste em aplicar uma técnica de agrupamento *fuzzy* (FCM-DFCV) e a partir dela gerar uma partição *crisp* e realizar a avaliação através de algumas métricas. O algoritmo FCM-DFCV[1] se trata de uma variação do método *Fuzzy C-Means*. Para a execução do algoritmo realizamos a seguinte sequência de passos: 1) geração dos protótipos; 2) cálculo da matriz **M** para os pesos das variáveis; 3) geração da matriz **U** para o cálculo do grau de pertinência de cada exemplo em cada grupo e por fim 4) o cálculo da função de custo **J**. Esse processo se repetiu por um total de 150 iterações ou quando atingiu os seguintes critérios de parada:

- J obteve um valor igual a zero;
- a diferença entre o J na iteração T-1 e o J na iteração T é menor que o *erro permissível* $\in$*;*
- O J obteve um número inválido[2].

Para a execução do algoritmo FCM-DFCV foram utilizados os seguintes hiperparâmetros:

c = 10; m = {1.1, 1.6, 2.0}; T = 150; = 10−10;

1.1 - Geração dos Protótipos

Na execução do algoritmo FCM-DFCV os primeiros representantes (protótipos) dos **c** grupos foram escolhidos aleatoriamente dentre os exemplos de todo o conjunto de dados, tomando o cuidado de não escolher um mesmo exemplo para múltiplos grupos.

[2] Isso ocorre quando o cálculo das matrizes **M** ou **U** exige um número decimal extremamente baixo ou alto, extrapolando o limite permitido em pontos flutuantes de computadores convencionais.

Após essa escolha aleatória foi construída uma matriz **U** inicial através da seguinte fórmula (Equação 03 [1]):

$$u_{ik} = \left[\sum_{h=1}^{c} \left\{ \frac{\sum_{j=1}^{p} (x_k^j - g_i^j)^2}{\sum_{j=1}^{p} (x_k^j - g_h^j)^2} \right\}^{1/(m-1)} \right]^{-1} \quad \text{for } i = 1, \ldots, c.$$

Onde g_i se refere ao **i**-ésimo protótipo, x_k o **k**-ésimo exemplo e j se refere o **j**-ésimo atributo. Após essa matriz inicial serão gerados os protótipos através da seguinte fórmula (Equação 03 [1]):

$$\mathbf{g}_i = \frac{\sum_{k=1}^{n} (u_{ik})^m \mathbf{x}_k}{\sum_{k=1}^{n} (u_{ik})^m}.$$

Esse passo a passo foi seguido devido a dependência da matriz **U**. A próxima etapa se refere a geração da matriz **M** que tem dependência tanto dos **G** protótipos quanto da matriz inicial **U**.

1.2 - Geração da Matriz de pesos M

A matriz **M** trata-se de uma matriz diagonal onde armazena uma espécie de peso de relevância para cada atributo, o peso para o atributo **J** é calculado de acordo com a seguinte fórmula (Equação 19 [1])

$$\lambda^j = \frac{\{\prod_{h=1}^{p} [\sum_{i=1}^{c} \sum_{k=1}^{n} (u_{ik})^m (x_k^h - g_i^h)^2]\}^{1/p}}{\sum_{i=1}^{c} \sum_{k=1}^{n} (u_{ik})^m (x_k^j - g_i^j)^2} \quad (j = 1, \ldots, p).$$

O próximo passo será a geração da matriz **U,** agora utilizando a matriz **M**.

1.3 - Geração da Matriz U

Uma vez computado os protótipos **G** e a matriz **M** a matriz **U** é recalculada, agora através da seguinte fórmula (Equação 20 [1])

$$u_{ik} = \left[\sum_{h=1}^{c} \left(\frac{d^2_{\mathbf{M}_i}(\mathbf{x}_k, \mathbf{g}_i)}{d^2_{\mathbf{M}_h}(\mathbf{x}_k, \mathbf{g}_h)} \right)^{1/(m-1)} \right]^{-1} = \left[\sum_{h=1}^{c} \left(\frac{\sum_{j=1}^{p} \lambda_i^j (x_k^j - g_i^j)^2}{\sum_{j=1}^{p} \lambda_h^j (x_k^j - g_h^j)^2} \right)^{1/(m-1)} \right]^{-1}.$$

Na qual $\sum_{i=1}^{c} u_{ik} = 1$ e o determinante da matriz **M** no grupo **i** é igual a 1.

1.4 - Cálculo da Função Objetivo J5

Após calcularmos as matrizes **G**, **M** e **U** é possível calcularmos a função de custo **J5** que guiará a execução do algoritmo FCM-DFCV. O cálculo do custo é realizado através da seguinte fórmula (Equação 17 [1]).

$$J5 = \sum_{k=1}^{n}\sum_{i=1}^{c}(u_{ik})^m d^2_{\mathbf{M}_i}(\mathbf{x}_k, \mathbf{g}_i) = \sum_{k=1}^{n}\sum_{i=1}^{c}(u_{ik})^m (\mathbf{x}_k - \mathbf{g}_i)^{\mathrm{T}}\mathbf{M}_i(\mathbf{x}_k - \mathbf{g}_i)$$

A implementação do algoritmo FCM-DFCV é ilustrada abaixo.

Algorithm 1 Algoritmo FCM-DFCV

Input O conjunto de dados X
Inicialização T=150, $e = 10^{-10}$, m=[1.1, 1.6, 2.0], J_{old}=99999, J=99999.

Construa a matriz **G** aleatoriamente;
Construa a matriz **U** a partir de **G** utilizando a Equação 4;
while $t < T$ and J_{old} - $J > e$ **do**
    $J_{old} = J$
    Gere novos protótipos **G** a partir da matriz **U** utilizando a Equação 3.
    Construa a matriz **M** a partir de **G** e da matriz **U** utilizando a Equação 19.
    Construa a matriz **U** a partir de **G** e da matriz **M** utilizando a Equação 20.
    Atualize **J** utilizando a Equação 17.
    $t = t + 1$

return J, U, G.

2.1 - Resultados do FCM-DFCV

Foram realizadas 3 avaliações das partições *fuzzy*, uma vez que cada avaliação corresponde a um valor diferente para o hiperparâmetro **m**. Dessa forma, ao executar o algoritmo 100 vezes para cada variação do **m** foi possível obter os seguintes resultados.

2.2 - *Modified Partition Coefficient*

O Coeficiente de Partição Modificado (MPC) é usado para medir a quantidade de sobreposição entre os grupos. Os valores devem estar dentro do intervalo 0 e 1, sendo que quanto mais próximo de 1 melhor o resultado para as classes. Os resultados para o MPC são apresentados na Tabela 1.

	MPC (m=1.1)	MPC (m=1.6)	MPC (m=2.0)
count	100	100	100
mean	0.993981	0.957169	0.951405
std	0.017853	0.115885	0.117852
min	0.900054	0.339813	0.511110
25%	0.999472	0.997256	0.990008
50%	1.000000	0.999804	0.998209
75%	1.000000	1.000000	0.999956
max	1.000000	1.000000	1.000000

Tabela 1: *Modified Partition Coefficient* em cada partição *fuzzy*.

Como é possível observar, os melhores resultados foram obtidos quando o **m** foi configurado como **1.1**, tendo inclusive uma baixa dispersão (0.018) quando comparado com as demais configurações.

2.3 - *Partition Entropy*

O *Partition Entropy* (PE) mede a imprecisão de uma partição da classe em uma determinada matriz do grau de pertinência de um objeto **i** ao *cluster* **k**. O intervalo de valores do índice PE está entre [0, log_a (C)]. Quanto mais próximo o valor da PE estiver de 0, mais nítido é o agrupamento. Logo, o valor do índice próximo ao limite superior indica a ausência de qualquer estrutura de agrupamento nos conjuntos de dados ou a incapacidade do algoritmo de extraí-la. A seguir na Tabela 2, seguem os respectivos valores do PE para cada partição.

	PE (m=1.1)	PE (m=1.6)	PE (m=2.0)
count	100	100	100
mean	9.388324e-03	6.428793e-02	7.631151e-02
std	2.774241e-02	1.696796e-01	1.719051e-01
min	1.236375e-08	1.407474e-14	2.651796e-11
25%	1.611522e-05	1.069152e-06	1.041061e-04
50%	4.150377e-05	4.053279e-04	2.805545e-03
75%	8.261557e-04	5.679193e-03	2.295916e-02
max	1.479648e-01	1.035130e+00	6.945560e-01

Tabela 2: *Partition Entropy* em cada partição *fuzzy*.

Conforme apresentado na Tabela 2, os melhores resultados foram alcançados quando o **m** foi configurado com o valor de 1.1, sugerindo assim essa parametrização para as partições *fuzzy*, isso quando observado os índices MPC e PE. A seguir serão realizadas algumas análises quando avaliamos as partições *crisp*.

2.4 - Partições *Crisp*

Em seguida, definiu-se a Partição *Crisp* (partição exclusiva) a partir da matriz **U** atribuindo para cada objeto i o *cluster* k que obteve o maior grau de pertinência u_{ik}. Utilizando a partição *crisp* e as classes previamente preenchidas do conjunto de dados foi possível calcular as métricas para o Índice de Rand corrigido e a *F-Measure.*

2.4.1 - Índice de Rand Corrigido

Foi utilizado o pacote “adjusted_rand_score” da biblioteca Scikit-learn para cálculo da métrica Índice de Rand Corrigido (IRC).

O Índice Rand Corrigido é usado para medir a similaridade dos pontos de dados presentes nos grupos, ou seja, quão semelhantes são as instâncias que estão presentes em cada grupo.

O intervalo de valores para o índice varia de -1 a 1, sendo quanto mais próximo de 1 melhor é a partição gerada. Os resultados obtidos foram:

	IRC (m=1.1)	IRC (m=1.6)	IRC (m=2.0)
count	100	100	100
mean	0.008086	0.011955	0.010284
std	0.010038	0.014384	0.007058
min	-0.005273	-0.002199	-0.004375
25%	0.002980	0.003713	0.008557
50%	0.008545	0.011632	0.011615
75%	0.011632	0.011632	0.011632
max	0.075881	0.113643	0.057885

Tabela 3: Índice de Rand Corrigido em cada partição *crisp*.

Como é possível notar, ao utilizarmos as partições geradas com o **m** configurado para **1.6**, obtivemos os melhores desempenhos quando observamos o IRC médio (0.012) e o valor máximo obtido (0.11).

2.4.2 - *F-Measure*

Para a determinação da métrica *F-measure* utilizou-se a funcionalidade “f1_score” da biblioteca Scikit-learn. O *F-measure* (FM) pode ser interpretada como uma média ponderada dos valores de duas métricas: a Precisão e o *Recall*.

O *F-measure* atinge seu melhor valor em 1 e o pior valor em 0, logo quanto mais próximo de 1 mais corretivo está a classificação. Além disso, esta métrica é mais útil quando se têm classes com tamanhos diferentes, o que é o nosso caso. Os resultados para cada partição são apresentados na Tabela 4.

	FM (m=1.1)	FM (m=1.6)	FM (m=2.0)
count	100	100	100
mean	0.110195	0.098726	0.091678
std	0.108505	0.108695	0.103957
min	0.000000	0.000000	0.000000
25%	0.024090	0.020889	0.021395
50%	0.040094	0.032345	0.034030
75%	0.166611	0.164420	0.129043
max	0.312668	0.311321	0.318059

Tabela 4: F-Measure em cada partição *crisp*.

De acordo com os resultados, pode-se observar que, em média, as melhores partições quando observamos o *F-measure* são aquelas quando geradas a partir do **m**=1.1, com isso podemos sugerir que o hiperparâmetro tenha um valor inferior a 2.0 para gerar um bom desempenho tanto em uma partição *fuzzy* quanto em uma partição *crisp*.

A seguir iremos selecionar, entre todos os resultados, aquele que obteve o menor custo (**J5**) e realizaremos algumas análises tanto na partição *fuzzy* quanto na partição *crisp*.

2.5 - Análise do Melhor Resultado

Após computarmos as 100 execuções para cada possível valor para o hiperparâmetro **m**, armazenamos os protótipos partir da menor função de custo **J5**, conforme listado abaixo:

Menor *J5*: 8.90926821735237e-30

Protótipos:

0	0.498955	0.498834	0.499937	0.259248	0.5	2.089737e-145	0.499494	0.276900
1	0.776150	0.733249	0.411142	0.273819	0.5	2.833927e-107	0.522439	0.220020
2	0.440003	0.529996	0.520002	0.229994	0.5	8.300000e-01	0.509999	0.220000
3	0.502000	0.496667	0.511333	0.253333	0.5	7.420000e-01	0.508667	0.237333
4	0.573723	0.593120	0.376120	0.224092	0.5	1.457722e-56	0.518729	0.227973
5	0.591429	0.577143	0.492857	0.252857	1.0	2.386426e-118	0.525714	0.279286
6	0.523599	0.547891	0.507444	0.561874	0.5	1.579188e-108	0.509171	0.230861
7	0.611038	0.584205	0.493854	0.218669	0.5	4.014276e-102	0.510490	0.228726
8	0.475059	0.490149	0.488930	0.226901	0.5	1.646073e-119	0.522499	0.695924
9	0.649557	0.612392	0.493696	0.214594	0.5	9.386093e-12	0.513643	0.223253

2.5.1 - Comparativo entre a Partição Crisp com as Classes a Priori

Com base no menor valor de custo **J5** geramos uma partição *fuzzy* e a partir dela uma partição *crisp*. Com essa partição calculamos a matriz de confusão comparando com as classes previamente definidas no conjunto de dados. Foi utilizado o pacote “confusion_matrix” da biblioteca Scikit-learn para cálculo da métrica Matriz de Confusão.

A Matriz de Confusão é útil para avaliar a precisão de uma classificação. A partir desta, é possível verificar o quanto a partição *crisp* gerada acertou/errou ao atribuir cada classe aos exemplos. A Figura 1 apresenta a matriz gerada.

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	459	0	4	0	0	0	0
1	0	0	0	0	0	5	0	0	0	0
2	0	0	0	35	0	0	0	0	0	0
3	0	0	0	44	0	0	0	0	0	0
4	0	0	0	50	0	1	0	0	0	0
5	0	0	0	162	0	1	0	0	0	0
6	0	0	0	244	0	0	0	0	0	0
7	0	0	0	426	0	3	0	0	0	0
8	0	0	0	20	0	0	0	0	0	0
9	0	0	0	30	0	0	0	0	0	0

Figura 1: Matriz de Confusão.

Conforme a Figura 1 a matriz de confusão gerada demonstra que a maioria dos objetos foram classificados na classe 3 e alguns poucos na classe

5, porém não houve a partir da partição *crisp* a classificação nas demais classes. Uma outra análise sobre a partição *crisp* é através dos índices ilustrados na Tabela 5.

Métrica	Valor
Acurácia[3]	0.3032
F-Measure	0.0069
IRC	0.0029
MPC	1
PE	0

Tabela 4: Índices gerados com a comparação das partições *crisp* e a priori.

Podemos observar que o algoritmo FCM-DFCV conseguiu convergir em um valor muito baixo para a função objetivo **J5**, porém a maioria dos exemplos tiveram um alto grau de pertinência em um só grupo (classe 3), ocasionando com isso um alto desempenho no quesito de agrupamento e um baixo desempenho quando comparado com a partição a priori. A seguir iremos abordar os experimentos realizados na etapa supervisionada do projeto.

Parte 2 - Análise de classificadores

Foram considerados quatro classificadores de aprendizagem supervisionada, e um quinto que representa a combinação dos 4 anteriores.

O objetivo consistiu-se na realização de uma análise dos 5 classificadores através do uso de quatro métricas diferentes (*Recall*, *F-Measure*, Precisão e Erro) além do teste de Friedman para comparação. Os classificadores consistiam em: Regressão Logística, um classificador Bayesiano baseado em K-vizinhos (KNN), um classificador Bayesiano baseado em Janela de Parzen (*Kernel Density Estimator*) e um classificador Bayesiano

[3] Foi utilizada a métrica: *accuracy_score*, Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score

Gaussiano. Os quatro classificadores foram treinados e testados com a base de dados "*Yeast Data Set*", através do uso de 5-fold estratificado.

Os resultados de cada classificador, para cada métrica, foram obtidos considerando um intervalo de confiança de 95%.

3.1 - Classificador Bayesiano Gaussiano

Foram geradas as médias μ_i e as matrizes de variância covariância (Σ_i) para cada classe a partir da Estimativa de Máxima Verossimilhança considerando a normal multivariada para a amostra de dados no treinamento.

A partir destes parâmetros computados, o objetivo do classificador é calcular as probabilidades a posteriori de cada exemplo do conjunto de teste para cada classe a partir da maior probabilidade. Como o classificador Bayesiano Gaussiano não possui nenhum hiperparâmetro que possa ser estimado para ajustar de uma melhor forma o modelo, então a performance do modelo ficou a mercê (depender) dos dados.

A Estimativa de Máxima Verossimilhança utilizada, é dada por:

$$p(x_k|\omega_i) = p(x_k|\omega_i, \theta_i) = (2\pi)^{\frac{-d}{2}}(|\Sigma_i^{-1}|)^{\frac{1}{2}} exp\{-\frac{1}{2}(x_k - \mu_i)^{tr}\Sigma_i^{-1}(x_k - \mu_i)\}$$

Os resultados podem ser vistos na tabela 5.

Bayes Gaussiano		
z = 1.96, K-Folds = 5	Média	Desvio Padrão
Erro	43.82%	2.314%
Cobertura (Recall)	43.7%	2.1819%
Precisão	43.82%	6.8210%
F-Score (F-Measure)	43.70%	3.8437%

Tabela 5: Resultado da aplicação das métricas no classificador Bayesiano Gaussiano.

Pelos resultados, é possível observar que as métricas estão alinhadas, com valores no mesmo intervalo. A média de erro ficou em 43%, um valor bastante alto. Com isso, pode-se dizer que o classificador acerta pouco mais da metade dos exemplos.

É importante ressaltar que os resultados apresentados dizem respeito ao modelo treinado e testado no conjunto de dados sem pré-processamento e com todas as *features* incluídas. Nesse caso, algumas classes possuem matrizes de covariância cujo determinante é 0, assim como a média de algumas classes também, o que “condena” a classificação.

3.2 - Classificador Bayesiano baseado em K-Vizinhos Mais Próximos

O classificador tem como objetivo atribuir um label a um objeto a partir do teorema de bayes ao considerar sua vizinhança pré-determinada. As distâncias foram calculadas através da distância euclidiana entre cada exemplo do conjunto de teste e os K vizinhos mais próximos do conjunto de treino. O K-NN possui um único hiperparâmetro (o próprio **K**), onde, através de ajuste utilizando parte do conjunto de validação, constatou-se que o melhor valor de **K** seria 4.

Na aplicação do K-NN, foi utilizada a biblioteca *Scikit-Learn*, mais especificamente o *sklearn.neighbors.KNeighborsClassifier*, com o **K** (testado) e definido com valor 4.

Os resultados do classificador podem ser vistos na tabela 6.

K-NN		
z = 1.96, K-folds = 5, k = 4	Média	Desvio Padrão
Erro	40.61%	0.1862%
Cobertura (Recall)	26.29%	2.1819%
Precisão	15.23%	0.0%
F-Score (F-Measure)	18.65%	0.02165%

Tabela 6: Resultado da aplicação das métricas no classificador K-NN.

No KNN temos resultados bastante diferentes do que foi apresentado no Bayes Gaussiano. Aqui, apesar da média de erro estar também acima de 40%. as outras três métricas obtiveram uma piora significativa, com a precisão sendo a mais afetada (apenas 15% de precisão). Novamente, assim como no Bayes Gaussiano, pode-se dizer que o classificador acerta pouco mais da metade dos exemplos.

3.3 - Classificador Bayesiano baseado em Janela de Parzen

A Janela de Parzen é uma abordagem não paramétrica amplamente utilizada para estimar uma função de densidade de probabilidade $p(x)$ para um ponto específico x a partir de uma amostra x_n que não requer qualquer conhecimento ou suposição sobre a distribuição subjacente.

Uma aplicação popular da Janela de Parzen é estimar as densidades condicionais de classe (ou também frequentemente chamadas 'probabilidades') $p(x \mid \omega_i)$ num problema de classificação de padrões supervisionado do conjunto de dados de formação (onde x é uma amostra multidimensional que pertence a uma determinada classe ω_i).

$$P(i \mid x) = p(x|i) \cdot P(i) \,/\, p(x) \Rightarrow probabilidade \;\; a \; posteriori$$

Para aplicar o método da Janela de Parzen foi utilizado o pacote *Scikit-Learn*, mais especificamente o *sklearn.neighbors.KernelDensity*[4] com um kernel gaussiano e o parâmetro *bandwidth* definido em 0.5

Na etapa de treinamento um KDE será treinado com exemplos de cada classe dos dados, então na etapa de predição esses KDEs por meio da função *score_samples* irão calcular o log das probabilidades para cada classe. Após isso será aplicada a função exponencial nas probabilidades e retornado um *array* com a classe mais provável para cada linha e a sua respectiva probabilidade. Os resultados do classificador podem ser vistos na tabela 7.

Janela de Parzen		
z = 1.96, K-Folds = 5, Bandwidth = 0.5	Média	Desvio Padrão
Erro	43.60%	2.387%
Cobertura (Recall)	41.60%	1.654%
Precisão	43.48%	4.378%
F-Score (F-Measure)	42.01%	1.334%

[4] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KernelDensity.html#sklearn.neighbors.KernelDensity

Tabela 7: Resultado da aplicação das métricas no classificador baseado em Janela de Parzen.

Os resultados do classificador baseado na Janela de Parzen são comparáveis aos resultados do Bayes Gaussiano. Todas as métricas obtiveram valores no mesmo intervalo. O erro ficou novamente acima de 40%.

3.4 - Regressão Logística

No caso da Regressão Logística, um classificador foi treinado para cada classe do conjunto de dados e foi utilizada a abordagem “Um contra todos (OvA)” para fazer a classificação dos exemplos da base de dados.

Assim como o classificador Bayesiano Gaussiano, o classificador da Regressão Logística também não possui hiperparâmetros ajustáveis, e seu treinamento ocorre através do treinamento de pesos e vieses, os quais, na hora da predição, serão aplicados à entrada e posteriormente à função de ativação (no nosso caso, Sigmoid). A função sigmoid é definida como:

$$f(x) = \frac{1}{1+e^{-(x)}}$$

Novamente foi utilizada a biblioteca *Scikit-Learn* (*sklearn.linear_model.LogisticRegression*) com o parâmetro “OvR” (One vs rest). Os resultados do classificador podem ser vistos na tabela 8.

Regressão Logística		
z = 1.96, K-Folds = 5	Média	Desvio Padrão
Erro	43.69%	4.296%
Cobertura (Recall)	43.69%	4.023%
Precisão	43.62%	4.33%
F-Score (F-Measure)	43.76%	3.646%

Tabela 8: Resultado da aplicação das métricas no classificador de Regressão Logística.

Mais uma vez podemos comparar os resultados do classificador de Regressão logística com o classificador Bayes Gaussiano e o baseado em

Janela de Parzen. As métricas dos três ficaram na mesma faixa de valores, e a média do erro novamente ficou acima de 40%.

3.5 - Ensemble de classificadores

O quinto classificador, nada mais era que a união dos outros quatro explicados anteriormente. Nesse caso, os quatro classificadores já treinados, faziam a predição dos dados do conjunto de testes e, através do voto majoritário (moda) era decidido qual a verdadeira classe de cada exemplo. Os resultados desse classificador podem ser vistos na tabela 9.

Ensemble		
z = 1.96, K-Folds = 5	Média	Desvio Padrão
Erro	15.03%	0.638%
Cobertura (Recall)	28.2%	2.694%
Precisão	8.19%	0.205%
F-Score (F-Measure)	11.11%	0.3801%

Tabela 9: Resultado da aplicação das métricas no Ensemble de classificadores.

O Ensemble de classificadores obteve a melhor média de erro entre todos os 5 classificadores avaliados. Com uma média de 15% de erro, o classificador possui uma taxa de acerto de pouco menos que 85%. Estranhamente, todas as outras métricas também obtiveram valores baixos, com destaque para a precisão, que ficou com pouco mais de 8% na média. Ainda sim, esse foi o classificador com maior variação nos resultados.

3.6 - Teste de Friedman para comparar os classificadores

Com o objetivo de comparar se os classificadores, anteriormente detalhados, será aplicado o teste de Friedman. O teste se refere a uma alternativa ao teste de ANOVA, uma vez que no teste de Friedman não necessitamos de assumir a normalidade dos dados. O objetivo do teste é o de avaliar a hipótese nula, na qual propõe que os desempenhos obtidos pelos

classificadores em cada métrica possuam um resultado estatisticamente similar. A estatística do teste (F_F) é calculada através da seguinte fórmula [6]

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2}$$

Onde N se refere ao número de experimentos, neste trabalho o número de folds, $k$se refere ao número de classificadores e χ^2_Fé definido como

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right]$$

O R se refere ao ranking médio do classificador $j$nos 5-folds e o Fcrítico pode ser facilmente obtido a partir da Tabela da distribuição F. Com 5−1 = 4 e (5−1)×(5−1) = 16 graus de liberdade.

O valor crítico de F(4,16) para α = 0.05 é 3.238872, dessa forma o resultado do teste de Friedman é ilustrado a seguir:

	metric	F_f	F_critical	result
0	error_rate	21.316456	3.238872	reject H0
1	f_measure	80.000000	3.238872	reject H0
2	precision	79.333333	3.238872	reject H0
3	recall	37.666667	3.238872	reject H0

Após rejeitar a hipótese nula (H0) é possível aplicar um teste post-hoc para complementar a análise.

3.7 - Teste Post-Hoc

O teste de Nemenyi se refere a um pós-teste ao resultado obtido pelo teste de Friedman. O objetivo do teste consiste em identificar uma diferença crítica (CD), de forma pareada, entre os classificadores, ao nível de significância de 5% e 10%. O valor do CD é calculado de acordo com a seguinte equação.

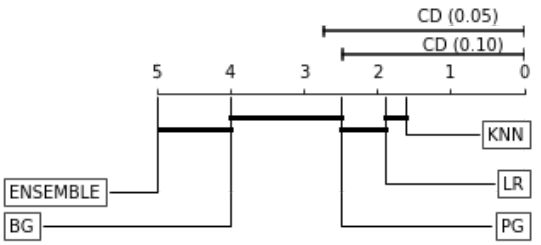
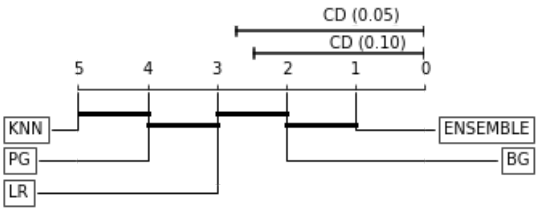
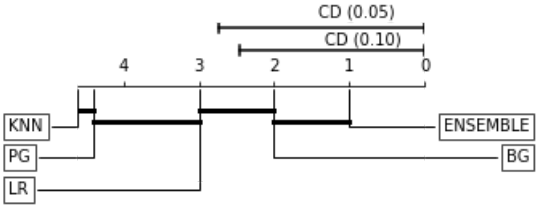
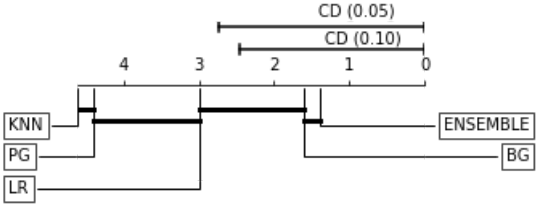
$$CD = q_{\alpha}\sqrt{\frac{k(k+1)}{6N}}$$

Onde q_{α} é obtido através da Tabela de nemenyi. Neste trabalho os valores para o CD foram:

CD (5%) = 2.728

CD (10%) = 2.459

A Tabela 10 apresenta os resultados do pós-teste dos classificadores para cada métrica.

Métrica	CD em 5% e 10%
Taxa de erro	
F-Measure	
Precisão	
Cobertura	

Conforme a Tabela acima, houve uma forte similaridade entre os desempenhos obtidos dos classificadores KNN e PG, ENSEMBLE e BG, sendo o classificador ENSEMBLE responsável pelo melhor ranking médio em 3 das 4 métricas avaliadas.

3.8 - Considerações Finais

Foi verificado através dos experimentos que a utilização de classificadores combinados pela regra do voto majoritário, resulta em uma melhor classificação de novos exemplos (melhor generalização por parte do classificador). Três dos cinco classificadores obtiveram performances bem parecidas, com as métricas atingindo resultados na mesma faixa de valores.

É importante notar que, com uma análise mais aprofundada dos dados, e um melhor pré-processamento, alguns dos classificadores podem obter uma performance ainda melhor, principalmente aqueles que são altamente dependentes dos dados de entrada (treino), como o Bayes Gaussiano. Uma outra sugestão, é o uso de outras métricas, uma vez que para muitos dos classificadores, as métricas atingiram resultados muito parecidos, tornando difícil fazer uma distinção de performance apenas por elas.

Por fim, mostrou-se que a união de classificadores não tão bons, atinge resultados aceitáveis e sem muito custo computacional adicional. Ademais, para poder comprovar com certeza, seria interessante a aplicação em outras bases de dados. O código utilizado no projeto está disponível no GitHub.

REFERÊNCIAS

[1] DE CARVALHO, Francisco de AT; TENÓRIO, Camilo P.; JUNIOR, Nicomedes L. Cavalcanti. **Partitional fuzzy clustering methods based on adaptive quadratic distances. Fuzzy Sets and Systems,** v. 157, n. 21, p. 2833-2857, 2006.

[2] FAN, J.-L.; WU, C.-M.; MA, Y.-L. **A modified partition coefficient. In: *5th International Conference on Signal Processing Proceedings*. 16th World Computer Congress 2000.** p. 1496-1499, 2000.

[3] CHENG, H. D.; CHEN, J.-R.; LI, J. **Threshold selection based on fuzzy c-partition entropy approach. *Pattern recognition*,** v. 31, n. 7, p. 857-870, 1998.

[4] TERPILOWSKI, M. Scikit-posthocs: **Pairwise multiple comparison tests in Python. Journal of Open Source Software,** v. 4, n. 36, 1169, 2019.

[5] VANDERPLAS, J. **Python Data Science Handbook.** 1. ed. [S.l.]: O'Reilly Media, Inc., 2016.

[6] DEMŠAR, Janez. Statistical comparisons of classifiers over multiple data sets. **The Journal of Machine Learning Research,** v. 7, p. 1-30, 2006.

[7] PEDREGOSA, Fabian et al. **Scikit-learn: Machine Learning in Python,** JMLR 12, pp. 2825-2830, 2011. Disponível em: <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html>. Acesso em: 28 de julho de 2021.