UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

ESTELA DE ANDRADE JOFFILY
GABRIEL GALDINO GADELHA
GABRIEL MIRANDA DE SOUZA
JOÃO PEDRO HENRIQUE SANTOS DUARTE
SÉRGIO DE VASCONCELOS FILHO

PROJETO FINAL
APRENDIZAGEM DE MÁQUINA

Recife-PE
2023

ESTELA DE ANDRADE JOFFILY
GABRIEL GALDINO GADELHA
GABRIEL MIRANDA DE SOUZA
JOÃO PEDRO HENRIQUE SANTOS DUARTE
SÉRGIO DE VASCONCELOS FILHO

PROJETO FINAL
APRENDIZAGEM DE MÁQUINA

Relatório do projeto apresentado como requisito parcial à obtenção de aprovação na disciplina de Aprendizagem de Máquina do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco.

Prof. Dr. Francisco de Assis Tenório de Carvalho
.

Recife-PE
2023

Sumário

1 Resumo

No presente trabalho foram utilizados três conjuntos de dados retiradas da base **Multiple Features** (DUIN, 1998) do repositório UCI *Irvine Machine Learning Repository* (DUA; GRAFF, 2017). Foram implementadas duas atividades para a disciplina Aprendizagem de Máquina, ministrada pelo programa de pós-graduação do Centro de Informática da Universidade Federal de Pernambuco no período de 2023.2.

A seção 2 apresenta uma descrição detalhada das bases de dados.

Na seção 3 é detalhada a primeira atividade realizada, de aprendizagem não-supervisionada, que consiste na implementação do algoritmo Fuzzy C-Means com Kernel Gaussiano e kernelização da métrica, e cálculo dos parâmetros de largura aplicado às bases de dados utilizadas, variando o hiperparâmetro de *fuzzificação*, para obter partições *fuzzy* e *crisp*.

Na seção 4 é descrita a segunda atividade, de aprendizagem supervisionada, em que são treinados quatro classificadores (Bayesiano Gaussiano, Bayesiano baseado em k-Vizinhos, Bayesiano baseado em Janela de Parzen e Regressão Logística) utilizando validação cruzada estratificada. Cada classificador inicial é treinado de forma independente em cada base de dados. Então um novo classificador é criado a partir do voto majoritário de cada um desses três treinamentos, para cada tipo de classificador. Por fim, os quatro classificadores obtidos são combinados num único classificador final.

As questões foram implementadas todas em *Python*, com uso das bibliotecas relevantes, como *NumPy*, *Pandas*, *Matplotlib*, *SciPy* e *Scikit-Learn*.

2 Apresentação do Dataset

O dataset utilizado neste projeto é o **Multiple Features** (DUIN, 1998). Este dataset é composto por *features* de numerais manuscritos (0 a 9, totalizando 10 classes) extraídos de uma coleção de mapas holandeses. Foram digitalizados 200 padrões por classe (totalizando 2000 padrões ou instâncias para cada base de dados) em imagens binárias. Para este projeto, foram utilizados os dados que representam os dígitos em termos de três conjuntos de *features* distintas:

1. **mfeat-fac**: Este conjunto de *features* consiste em 216 correlações de perfil. As correlações de perfil são medidas que descrevem como a intensidade dos pixels em uma imagem varia ao longo de um perfil específico, como uma linha horizontal ou vertical. Essas correlações ajudam a capturar informações detalhadas sobre a estrutura dos caracteres, incluindo como os pixels estão distribuídos e como as *features* estão relacionadas entre si.

2. **mfeat-fou**: Este conjunto de *features* é composto por 76 coeficientes de Fourier que descrevem as formas dos caracteres. Os coeficientes de Fourier são amplamente utilizados em análises de sinais e imagens para representar características periódicas ou oscilatórias.

No contexto de dígitos manuscritos, esses coeficientes capturam informações importantes sobre a forma dos caracteres, como as curvas e oscilações presentes na escrita manual.

3. **mfeat-zer**: Esse conjunto de *features* é composto por 47 momentos de Zernike. Os momentos de Zernike são usados para descrever a forma dos objetos de uma maneira que é robusta à rotação e escala. Eles são especialmente úteis na análise de forma e no reconhecimento de padrões, pois permitem representar as características dos dígitos independentemente de como eles são orientados ou dimensionados.

Cada *dataset* fornece representações diferentes e complementares dos dígitos manuscritos, tornando-os adequados para uma variedade de tarefas de reconhecimento de dígitos e análise de imagem.

As bases não apresentavam dados faltantes nem duplicados. Os dados foram normalizados para manter os valores dos atributos dentro de uma escala comum.

3 Questão 1: KFCM-K-W.2

3.1 Algoritmo

Foi implementado o algoritmo KFCM-K-W.2 descrito em (SIMõES; CARVALHO, 2023), que consiste em um modelo não-supervisionado de aprendizagem de máquina em que os elementos são agrupados em *clusters* de acordo com sua similaridade. Esses *clusters* são do tipo *fuzzy*, ou seja, cada instância possui um grau de pertencimento u para cada *cluster*. Esse modo de particionamento se opõe ao tipo *crisp* ou *hard*, em que cada instância é atribuída a somente um *cluster*, sendo seu grau de pertencimento a este cluster igual a 1, e igual a 0 para os demais.

O algoritmo recebe como entrada:

- O conjunto de dados $\mathscr{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, onde $\mathbf{x}_k$ representa uma instância da base, e x_{kj} é o valor do j-ésimo atributo da k-ésima instância ($1 \leq k \leq n$; $1 \leq j \leq p$). No nosso caso, $k = 2000$ e p varia para cada base.

- O número de *clusters* c a que as instâncias serão atribuídas. Foi utilizada a quantidade de clusters igual à quantidade de classes. Para a base *multiple features*, $c = 10$.

- O número máximo de iterações (épocas) T que o algoritmo irá rodar a cada execução. Foi estabelecido $T = 100$.

- A constante de parada (limiar) ε. Foi determinado o valor de $\varepsilon = 10^{-6}$.

- O parâmetro de *fuzzificação* m. Foram testados diferentes valores para m, tem-se que $m = \{1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9\}$. Este hiperparâmetro determina o quão *fuzzy* será o particionamento das instâncias em *clusters*; quanto mais próximo de 1, mais *crisps* serão as partições.

- Um valor de *seed* (opcional), para inicialização aleatória dos protótipos $\mathbf{g}$. Com a utilização de um *seed*, é possível a replicação de resultados para fins de documentação, *debugging* e transparência.

A cada iteração, são redefinidos:

- Os vetores locais de parâmetro de janela: $\mathbf{s}_1, \ldots, \mathbf{s}_c$, onde $s_{ij} \equiv \frac{1}{s_{ij}^2}$ representa o valor do j-ésimo parâmetro de janela do vetor $\mathbf{s}_i$ ($1 \leq i \leq c$; $1 \leq j \leq p$). Na inicialização, cada s_{ij} recebe o valor 1, e após isso, vale que $\prod_{j=1}^{p} \frac{1}{s_{ij}^2} = 1$.
- Os representantes dos *clusters* (protótipos): $\mathbf{g}_1, \ldots, \mathbf{g}_c$, onde g_{ij} representa o valor do j-ésimo atributo do protótipo $\mathbf{g}_i$ ($1 \leq i \leq c$; $1 \leq j \leq p$). Na inicialização, são escolhidos de forma aleatória dentre as instâncias da base.
- A matriz $\mathbf{U} = [u_{ki}]$ de graus de pertencimento da instância $\mathbf{x}_k$ ao *cluster* representado pelo protótipo $\mathbf{g}_i$ ($1 \leq k \leq n$; $1 \leq i \leq c$). Vale que $u_{ki} \geq 0$ e $\sum_{i=1}^{c} u_{ki} = 1$.
- A função objetivo, dada pela Eq. 1, que deve ser minimizada a cada nova atualização.

$$J_{KFCM-K-W.2} = \sum_{i=1}^{c} \sum_{k=1}^{n} (u_{ki})^m (2 - 2\mathscr{K}^{(\mathbf{s}_i)}(\mathbf{x}_k, \mathbf{g}_i)) \tag{1}$$

O kernel Gaussiano é um dos mais utilizados na literatura. No algoritmo KFCM-K-W.2 ele possui o seu parâmetro de largura s variável para cada *cluster* e para cada parâmetro p do conjunto de dados. O kernel Gaussiano do algoritmo KFCM-K-W.2 é definido na Eq. 2 a seguir:

$$\mathscr{K}^{(\mathbf{s}_i)}(\mathbf{x}_k, \mathbf{g}_h) = \exp\left\{ -\frac{1}{2} \sum_{j=1}^{p} \frac{1}{s_{ij}^2} (x_{kj} - g_{hj})^2 \right\} \tag{2}$$

O algoritmo itera até que o número máximo de épocas seja atingido ($t > T$) ou a diferença entre duas funções objetivo consecutivas atinja o limiar ($|J_{NEW} - J_{OLD}| < \varepsilon$). Para um maior detalhamento do funcionamento e implementação do algoritmo KFCM-K-W.2, consultar o trabalho realizado por (SIMõES; CARVALHO, 2023).

O algoritmo foi executado 50 vezes para cada valor de m em cada base. Selecionamos o melhor resultado (aquele que forneceu o menor valor para a função objetivo) para cada base e armazenamos os protótipos, os parâmetros de janela e as métricas. Produzimos também um particionamento *crisp* a partir dos graus de pertencimento *fuzzy*, onde o valor 1 é atribuído ao *cluster* com o maior valor dentre os demais, e o valor 0 atribuído ao restante. Construímos uma matriz de confusão relacionando as classes *a priori* às partições *crisp*. Esses resultados são discutidos na seção 3.3.

3.2 Métricas

Foram definidas três métricas diferentes para avaliar a performance do algoritmo nas bases: acurácia, índice de Rand corrigido e coeficiente de partição modificado. Uma breve discussão sobre essas métricas e suas respectivas definições é feita nas seções seguintes.

3.2.1 Acurácia

A acurácia é uma medida estatística do quão bem um teste de classificação identifica ou não as classes corretamente. Nesta atividade foi utilizada a acurácia como definida na Eq. 3:

$$\text{Acc} = \sum_{i=1}^{K} \frac{n_i}{n} P_i \tag{3}$$

Em que n_i é o número de elementos do *cluster* i, n é o número total de elementos, K é o número de classes *a priori* e P_i é definido a seguir:

$$P_i = \max_{1 \leq j \leq C} \left(\frac{n_{ij}}{n_i} \right) \tag{4}$$

Em que n_{ij} é o número de elementos atribuídos ao cluster i pertencentes à classe j, e C é o número de *clusters*. P_i representa a máxima probabilidade de que os membros do *cluster* i pertençam à classe j.

3.2.2 Índice de Rand Corrigido

O Índice de Rand corrigido é uma métrica utilizada para calcular a similaridade entre dois *clusters crisps*. O valor do índice de Rand é corrigido para levar em consideração a aleatoriedade na predição dos *clusters*. O índice de Rand corrigido possui valor próximo a 0 quando a predição é feita de forma totalmente aleatória independentemente da quantidade de *clusters*. O índice de Rand corrigido é definido a seguir na Eq. 5:

$$\text{Rand} = \frac{TP + TN - \text{corr}}{TP + TN + FN + FP - \text{corr}} \tag{5}$$

Em que TP, TN, FN e FP são a quantidade de verdadeiros positivos, verdadeiros negativos, falsos negativos e falsos positivos, respectivamente. O fator de correção corr é definido como:

$$\text{corr} = \frac{(TP + FN)(TP + FP) + (TN + FN)(TN + FP)}{TP + TN + FN + FP} \tag{6}$$

3.2.3 Modified Partition Coefficient

O coeficiente de partição avalia a separação entre as partições *fuzzy*, assumindo o valor $\frac{1}{c}$ no caso em que os graus de pertencimento u dos *clusters* forem iguais, e valor 1 quando os graus

de pertencimento tendem a uma partição *crisp* (SIMõES; CARVALHO, 2023). Já o coeficiente de partição modificado é um ajuste para que o valor da métrica fique no intervalo (0,1). O coeficiente de partição modificado é definido como:

$$\text{MPC} = 1 - \frac{c}{c-1}\left[1 - \left(\frac{1}{n}\sum_{k=1}^{n}\sum_{i=1}^{c} u_{ki}^2\right)\right] \tag{7}$$

3.3 Resultados

Os resultados das métricas obtidas com as bases mfeat-fac, mfeat-fou e mfeat-zer são mostrados nas Tabelas 1, 2 e 3, respectivamente.

Tabela 1. *Tabela com valores das métricas referentes à base* mfeat-fac *para a Questão 1*

	Métricas		
Hiperparâmetro "m"	**Acurácia**	**Índice Rand Corrigido**	**MPC**
1.3	0.6724 ± 0.0636	0.4845 ± 0.0780	0.1020 ± 0.0140
1.4	0.6520 ± 0.0659	0.4595 ± 0.0776	0.0505 ± 0.0087
1.5	0.6371 ± 0.0684	0.4530 ± 0.0763	0.0282 ± 0.0054
1.6	0.6190 ± 0.0686	0.4398 ± 0.0769	0.0169 ± 0.0031
1.7	0.6077 ± 0.0680	0.4302 ± 0.0757	0.0109 ± 0.0020
1.8	0.5930 ± 0.0644	0.4148 ± 0.0709	0.0074 ± 0.0014
1.9	0.5695 ± 0.0560	0.3915 ± 0.0629	0.0054 ± 0.0009

Tabela 2. *Tabela com valores das métricas referentes à base* mfeat-fou *para a Questão 1*

	Métricas		
Hiperparâmetro "m"	**Acurácia**	**Índice Rand Corrigido**	**MPC**
1.3	0.3547 ± 0.0061	0.1800 ± 0.0038	0.0743 ± 0.0040
1.4	0.2861 ± 0.0340	0.1471 ± 0.0134	0.0265 ± 0.0063
1.5	0.1985 ± 0.0000	0.1114 ± 0.0000	0.0000 ± 0.0000
1.6	0.1985 ± 0.0000	0.1114 ± 0.0000	0.0000 ± 0.0000
1.7	0.1985 ± 0.0000	0.1114 ± 0.0000	0.0000 ± 0.0000
1.8	0.1980 ± 0.0000	0.1112 ± 0.0000	0.0000 ± 0.0000
1.9	0.1980 ± 0.0000	0.1112 ± 0.0000	0.0000 ± 0.0000

Nas Tabelas 4, 5 e 6 temos as métricas obtidas na iteração cuja função objetivo obteve o menor valor, ou seja, o melhor resultado segundo a função objetivo dentre as 50 execuções de cada base.

As Tabelas 7, 8 e 9 apresentam as matrizes de confusão para cada base, mostrando o número de instâncias de cada classe que foi atribuído a cada *cluster*. As matrizes de confusão são referentes à iteração com menor valor da função objetivo quando $m = 1.3$, que foi o m que obteve as melhores métricas.

Tabela 3. *Tabela com valores das métricas referentes à base* mfeat-zer *para a Questão 1*

	Métricas		
Hiperparâmetro "m"	**Acurácia**	**Índice Rand Corrigido**	**MPC**
1.3	0.4891 ± 0.0092	0.3066 ± 0.0075	0.3487 ± 0.0072
1.4	0.4506 ± 0.0046	0.2972 ± 0.0029	0.1677 ± 0.0026
1.5	0.4196 ± 0.0158	0.2769 ± 0.0229	0.0870 ± 0.0071
1.6	0.3582 ± 0.0273	0.2165 ± 0.0249	0.0388 ± 0.0016
1.7	0.2879 ± 0.0176	0.1701 ± 0.0112	0.0210 ± 0.0002
1.8	0.2841 ± 0.0132	0.1613 ± 0.0092	0.0138 ± 0.0000
1.9	0.3028 ± 0.0190	0.1612 ± 0.0145	0.0093 ± 0.0000

Tabela 4. *Tabela com valores das métricas referentes ao melhor resultado da base* mfeat-fac *segundo a função objetivo para a Questão 1*

	Métricas			
Hiperparâmetro "m"	**Função objetivo**	**Acurácia**	**Índice Rand Corrigido**	**MPC**
1.3	1789.31881	0.7230	0.5520	0.1060
1.4	1443.5908	0.7165	0.5445	0.0535
1.5	1155.3738	0.5350	0.3324	0.0212
1.6	921.6805	0.5315	0.3325	0.0137
1.7	734.0058	0.5180	0.3271	0.0092
1.8	584.3493	0.5205	0.3303	0.0065
1.9	464.9451	0.5175	0.3272	0.0048

Tabela 5. *Tabela com valores das métricas referentes ao melhor resultado da base* mfeat-fou *segundo a função objetivo para a Questão 1*

	Métricas			
Hiperparâmetro "m"	**Função objetivo**	**Acurácia**	**Índice Rand Corrigido**	**MPC**
1.3	1284.6935	0.3555	0.1808	0.0747
1.4	1032.0511	0.2750	0.1427	0.0260
1.5	822.0517	0.1985	0.1114	0.0000
1.6	652.9789	0.1985	0.1114	0.0000
1.7	518.6795	0.1985	0.1114	0.0000
1.8	412.0018	0.1980	0.1111	0.0000
1.9	327.2647	0.1980	0.1111	0.0000

A partir das tabelas é possível observar que as métricas obtiveram os melhores valores a partir da base *mfeat-fac*. Isso indica que no uso do algoritmo KFCM-K-W.2 a extração de atributos da base *mfeat* que obteve melhor desempenho foi a partir das correlações de perfil (*mfeat-fac*). Em seguida, a extração de atributos a partir dos momentos de Zernike *mfeat-zer* gerou as melhores métricas após a *mfeat-fac*. Por último, a base *mfeat-fou* a partir de coeficientes de Fourier.

Tabela 6. *Tabela com valores das métricas referentes ao melhor resultado da base* mfeat-zer *segundo a função objetivo para a Questão 1*

	Métricas			
Hiperparâmetro "m"	**Função Objetivo**	**Acurácia**	**Índice Rand Corrigido**	**MPC**
1.3	846.6011	0.4920	0.3052	0.3528
1.4	710.8758	0.4560	0.2970	0.1704
1.5	581.9679	0.4340	0.2963	0.0943
1.6	469.3876	0.3970	0.2458	0.0411
1.7	375.3939	0.2730	0.1604	0.0208
1.8	299.4428	0.2795	0.1559	0.0138
1.9	238.5819	0.3365	0.1798	0.0093

Tabela 7. *Matriz de confusão da melhor configuração, quando a função objetivo J assume o menor valor com* $m = 1.3$ *referente à base* mfeat-fac *para a Questão 1*

	Clusters									
Classes	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	0	6	0	0	0	1	106	1	86
1	10	3	4	1	1	166	10	0	5	0
2	0	4	0	2	3	0	190	0	1	0
3	0	9	0	5	9	2	5	0	170	0
4	156	0	0	0	0	25	0	6	13	0
5	4	45	47	0	1	0	22	4	77	0
6	189	0	8	0	0	0	0	1	2	0
7	0	19	0	84	95	0	2	0	0	0
8	2	0	177	0	0	0	10	4	6	1
9	0	183	1	0	1	0	10	0	5	0

Tabela 8. *Matriz de confusão da melhor configuração, quando a função objetivo J assume o menor valor com* $m = 1.3$ *referente à base* mfeat-fou *para a Questão 1*

	Clusters									
Classes	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	18	0	2	0	0	0	0	180	0
1	0	13	0	148	0	0	39	0	0	0
2	0	14	0	156	0	0	30	0	0	0
3	0	1	0	72	0	0	127	0	0	0
4	0	1	0	26	0	0	173	0	0	0
5	0	0	0	70	0	0	130	0	0	0
6	0	4	0	28	0	0	168	0	0	0
7	0	0	0	21	0	0	179	0	0	0
8	0	196	0	0	0	0	4	0	0	0
9	0	6	0	27	0	0	167	0	0	0

Tabela 9. *Matriz de confusão da melhor configuração, quando a função objetivo J assume o menor valor com* $m = 1.3$ *referente à base* mfeat-zer *para a Questão 1*

	Clusters									
Classes	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	0	95	0	2	0	0	98	1	3
1	0	80	2	0	26	84	0	5	1	2
2	6	0	7	1	7	3	67	2	3	104
3	8	0	2	7	28	7	106	0	10	32
4	5	49	0	7	79	56	0	1	3	0
5	7	2	4	1	15	8	35	28	37	63
6	107	0	5	4	11	10	5	3	50	5
7	1	0	0	161	21	0	1	0	16	0
8	9	0	105	0	2	0	4	64	8	8
9	117	1	3	1	14	8	4	1	46	5

Das tabelas, observa-se também que as métricas do modelo estão intimamente relacionadas ao hiperparâmetro de *fuzzificação m*. À medida que *m* foi aumentando, os valores das métricas diminuíram. Os melhores valores das métricas foram obtidos quando $m = 1.3$, tanto para acurácia quanto para o índice Rand corrigido, atingindo valores de 0.7230 e 0.5520, respectivamente, para a base *mfeat-fac*. Já para base *mfeat-fou* a acurácia e o índice Rand corrigido obtiveram valores de até 0.3555 e 0.1808, respectivamente. E para a base *mfeat-zer*, valores de 0.4920 e 0.3052, respectivamente. Além disso, quanto mais próximo de 1 está o valor de *m*, mais próximos os *clusters* estão de partições *crisp*, como pode-se observar através dos valores do coeficiente de partição modificado, que diminuíam à medida que *m* aumentava. As partições mais *crisp* ao final dos testes foram da base *mfeat-zer*, que teve valores de *m* de até 0.3528. As outras duas bases tiveram valores menores de *m*, ou seja, partições mais *fuzzy*, com alguns valores sendo até zerados (graus de pertencimento iguais ou praticamente iguais).

Na matriz de confusão da base *mfeat-fac* (Tabela 7) nota-se que, no geral, houve uma divisão relativamente balanceada das classes nos diferentes *clusters*. As classes '4' e '6' foram as duas classes que mais se confundiram entre si, ambas ficando no mesmo *cluster* 1 com 156 e 189 instâncias cada, respectivamente. Já as classes '0' e '7' se dividiram aproximadamente em metade das suas instâncias em um *cluster* e a outra metade em outro. Já na matriz de confusão da base *mfeat-fou* (Tabela 8), observa-se que houve um aglomerado bem maior de classes diferentes no mesmo *cluster*, por exemplo as classes '3', '4', '5', '6', '7' e '9' todas com mais de 100 instâncias cada no mesmo *cluster* 7. O *cluster* 4 também com várias instâncias de classes diferentes. Em contrapartida, o *cluster* 9 reuniu instâncias apenas da classe '0', o que sugere que os coeficientes de Fourier podem ser uma boa representação para a classe 0. A matriz de confusão da base *mfeat-fou* foi a única a possuir *clusters* sem nenhum elemento de nenhuma classe. Como apresentado na Tabela 8, os *clusters* 1, 3, 5, 6, 8 e 10 não possuem nenhum elemento. Por fim, a matriz de confusão da base *mfeat-zer* (Tabela 9) não possui aglomerações

de diversas classes no mesmo *cluster* de maneira tão evidenciada como na da base *mfeat-fou*, porém não tem uma representação tão boa quanto a da base *mfeat-fac*.

As matrizes de protótipos e parâmetros de largura, e as matrizes de confusão de todas as bases para cada *m* no melhor valor da função objetivo estão disponíveis nos materiais complementares.

4 Questão 2: Comitê de Classificadores

Essa questão envolve realizar o treinamento de cada dataset em quatro classificadores diferentes: o Bayesiano Gaussiano, o K-Vizinhos, a Janela de Parzen e a Regressão Logística. Posteriormente, cada modelo foi combinado utilizando a regra do voto majoritário. A estratégia de treinamento e validação utilizada para avaliar o desempenho dos modelos foi a de validação cruzada **30**$\times$**10-*folds*** com amostragem estratificada.

4.1 Implementação, Treinamento e Tuning de Hiperparâmetros

Os classificadores utilizados tiveram como base a implementação disponível na biblioteca Scikit Learn (COURNAPEAU, 2023). Em alguns cenários, a implementação padrão da biblioteca precisou ser ajustada para corresponder aos requisitos do projeto. Nos classificadores onde foi necessário realizar *tuning* de hiper-parâmetros, o seguinte procedimento foi adotado para cada uma das três bases de dados (fac, fou e zer):

1. Para cada uma das 30 iterações do treinamento, escolhemos uma das permutações de *folds*.
2. Da permutação escolhida, selecionamos aleatoriamente uma amostra estratificada de 20% do conjunto de treinamento.
3. Executamos uma busca com valores pré-definidos para cada hiper-parâmetro. No fim, a combinação de hiper-parâmetros que maximiza a acurácia foi salva.
4. Ao final das 30 iterações, os valores dos hiper-parâmetros foram definidos utilizando os seguintes critérios:
   (a) Moda para valores discretos;
   (b) Média aritmética para valores contínuos.

A seguir, discutiremos o processo de implementação, treinamento e escolha de hiperparâmetros para cada um dos classificadores utilizados.

4.1.1 Bayesiano Gaussiano

O modelo Bayesiano Gaussiano foi implementado através da classe *QuadraticDiscriminantAnalysis* disponibilizado pela biblioteca Scikit-learn. Este classificador possui um limite de decisão quadrático, gerado ao ajustar densidades condicionais de classe aos dados e aplicar a regra de Bayes.

A fórmula de Bayes é usada para calcular a probabilidade condicional de uma classe, maximizando essa probabilidade posterior. Matematicamente, para análise discriminante linear e quadrática, $P(x|y)$ é modelado como uma distribuição Gaussiana multivariada com densidade:

$$P(x|y=k) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^t \Sigma_k^{-1}(x-\mu_k)\right) \tag{8}$$

Onde d é o número de características e k indica a classe.

No contexto deste projeto, a estimativa de máxima verossimilhança para $P(\omega_i)$ e $p(x_k|\omega_i,\theta_i)$, assumindo uma distribuição normal multivariada para cada classe $\omega_i (1 \leq i \leq 10)$, é fundamentada conforme a formulação acima. Além disso, realizamos a regularização das estimativas de covariância de cada classe com o parâmetro `reg_param`, variando entre os valores [0.001, 0.01, 0.1, 1, 10, 100, 1000], para evitar o sobreajuste e melhorar a robustez do modelo.

Os resultados do ajuste fino dos hiperparâmetros são detalhados na Tabela 10, onde a escolha do valor ótimo de `reg_param` para cada base de dados é apresentada.

Tabela 10. *Fatores de regularização adotados para cada dataset*

Dataset	FAC	FOU	ZER
Fator de regularização	1	0.001	1

4.1.2 K-Vizinhos

Tomamos como base o classificador *KNeighborsClassifier* disponível na biblioteca Scikit Learn. O classificador foi configurado para utilizar a distância euclidiana e o único hiperparâmetro que foi customizado foi a quantidade de vizinhos k.

Para escolher o valor de k, realizamos uma busca variando os valores de k no intervalo $[1, 10]$ conforme o processo descrito anteriormente. Por fim, os valores de k escolhidos foram os que mais se repetiram na busca. A Tabela 11 mostra os valores finais adotados para cada dataset.

Tabela 11. *Melhores valores de k para cada dataset*

Dataset	FAC	FOU	ZER
k	4	5	5

4.1.3 Janela de Parzen

O classificador baseado na janela de Parzen, é um algoritmo de classificação não paramétrico que utiliza a estimativa de densidade de Parzen para classificação dos dados (PARZEN, 1962; ROSENBLATT, 1956). A fórmula de Bayes é aplicada para calcular a probabilidade condicional de uma classe usando as estimativas de densidade de probabilidade obtidas pela estimativa de densidade de Parzen (RASCHKA, 2014).

Para a implementação do classificador de Janela de Parzen, empregamos o classificador *KernelDensity* da biblioteca Scikit-learn (COURNAPEAU, 2023) com a função de kernel gaussiano. Ajustamos o hiperparâmetro de largura de banda por meio de uma busca logarítmica, testando 50 valores entre 0.01 e 100. Para cada conjunto de dados, calculamos a média dos melhores valores de largura de banda e os detalhamos na Tabela 12.

Tabela 12. *Valores adotados para a largura de banda para cada dataset*

Dataset	FAC	FOU	ZER
Largura de banda	26.060	0.035	20.833

Para cada um dos três datasets, estimamos a densidade de probabilidade de cada uma das dez classes possíveis através de estimativa de densidade de Parzen. O modelo *KernelDensity* estima a distribuição de apenas uma classe fornecida como entrada por vez. Para cada classe, o modelo foi ajustado individualmente, fornecendo estimativas probabilísticas numéricas para a pertinência de um item à classe correspondente. Durante o processo de validação cruzada, segmentamos o conjunto de treinamento em dez partes, criando um modelo específico para cada classe. A atribuição final da classe aos elementos de teste foi baseada no modelo que atribuiu a maior probabilidade a cada item.

4.1.4 Regressão Logística

Para a implementação do classificador de Regressão Logística, foi utilizado como base o classificador disponível na biblioteca Scikit Learn. O classificador foi ajustado para utilizar a estratégia um contra todos bem como teve a quantidade máxima de iterações escolhida a fim de que o algoritmo sempre convergisse.

A fim de evitar *overfitting* no modelo, introduzimos uma regularização usando funções de penalidade e um fator de regularização. Estes dois hiperparâmetros atuam penalizando os coeficientes dos atributos, diminuindo a complexidade do modelo e reduzindo a chance de *overfitting*.

No Scikit Learn, temos duas funções de penalidade disponíveis (Lasso e Ridge) e é possível informar um fator de regularização a ser usado durante o treinamento. Realizamos buscas desses dois hiperparâmetros conforme o procedimento descrito acima. Para cada *dataset*, as combinações de função penalidade e fator de regularização adotadas estão representadas na tabela 13.

Tabela 13. *Função de penalidade e fator de regularização adotados*

Dataset	**FAC**	**FOU**	**ZER**
Função de penalidade	Lasso	Ridge	Ridge
Fator de regularização	0.1	0.1	1000

4.2 Composição dos Classificadores

Para cada classificador base introduzido anteriormente, construímos um classificador composto utilizando a regra do voto majoritário. Este classificador recebe o exemplo com os atributos de cada *dataset* e repassa a cada classificador que o compõe apenas os atributos referentes ao seu *dataset* para que seja realizada a predição da classe ao qual o exemplo pertence.

Essas predições são então avaliadas utilizando a regra do voto majoritário, sendo a classe que mais se repete escolhida pelo classificador composto. Na ocasião de ocorrer um empate entre as classes, o classificador escolhe aleatoriamente uma das classes como resposta. A Figura 1 ilustra o processo adotado.

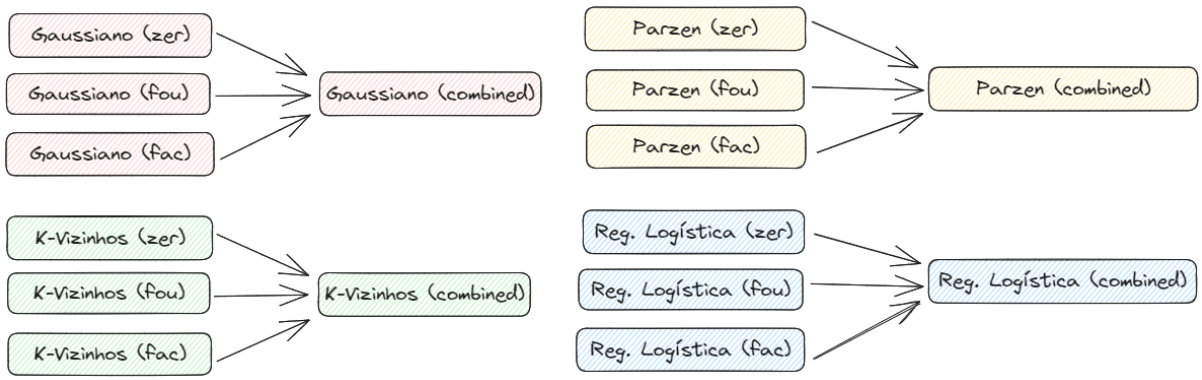

Figura 1. *Representação de como os datasets e classificadores foram combinados*

4.3 Estimativa pontual e intervalo de confiança

Para a realização do experimento, utilizamos validação estratificada cruzada **30**$\times$**10-*folds***, totalizando 300 iterações dos classificadores em diferentes permutações dos *datasets*. Para cada iteração, coletamos as métricas de taxa de erro, precisão, cobertura e Score F1.

De posse destas métricas, realizamos o cálculo do valor médio e do desvio padrão amostral, e construímos os intervalos de confiança de 95%. Para a estimativa pontual, utilizamos a média amostral do valor da métrica. Os resultados estão apresentados nas Tabelas 14 e 15. A Tabela 14 apresenta o resultado por *dataset* para cada tipo de classificador treinado, enquanto a Tabela 15 apresenta o resultados dos modelos resultantes da regra do voto majoritário.

Para os classificadores combinados, a diferença entre as estimativas pontuais entre os classificadores superaram consistentemente os respectivos intervalos de confiança, indicando diferenças significativas entre os classificadores para todas as métricas. Sob a premissa de

Tabela 14. *Estimativa pontual e intervalo de confiança por dataset, para cada classificador*

Classificador	Dataset	Taxa de Erro	Precisão	Cobertura	Score F1
Bayesiano Gaussiano	FAC	0.175 ± 0.003	0.836 ± 0.003	0.825 ± 0.003	0.825 ± 0.003
	FOU	0.160 ± 0.003	0.844 ± 0.003	0.840 ± 0.003	0.839 ± 0.003
	ZER	0.285 ± 0.003	0.717 ± 0.003	0.715 ± 0.003	0.711 ± 0.003
K-Vizinhos	FAC	0.044 ± 0.002	0.958 ± 0.002	0.956 ± 0.002	0.956 ± 0.002
	FOU	0.159 ± 0.002	0.845 ± 0.002	0.841 ± 0.002	0.841 ± 0.002
	ZER	0.197 ± 0.002	0.804 ± 0.002	0.903 ± 0.002	0.802 ± 0.002
Janela de Parzen	FAC	0.343 ± 0.004	0.688 ± 0.004	0.657 ± 0.004	0.663 ± 0.004
	FOU	0.444 ± 0.004	0.593 ± 0.003	0.556 ± 0.004	0.566 ± 0.003
	ZER	0.262 ± 0.003	0.752 ± 0.002	0.738 ± 0.003	0.742 ± 0.003
Regressão Logística	FAC	0.022 ± 0.001	0.979 ± 0.001	0.978 ± 0.001	0.978 ± 0.001
	FOU	0.176 ± 0.003	0.826 ± 0.003	0.824 ± 0.003	0.822 ± 0.003
	ZER	0.170 ± 0.002	0.824 ± 0.002	0.830 ± 0.002	0.825 ± 0.002

Tabela 15. *Estimativa pontual e intervalo de confiança para os classificadores combinados*

Classificador	Taxa de Erro	Precisão	Cobertura	Score F1
Bayesiano Gaussiano	0.138 ± 0.002	0.867 ± 0.002	0.862 ± 0.002	0.861 ± 0.002
K-Vizinhos	0.100 ± 0.002	0.902 ± 0.002	0.900 ± 0.002	0.900 ± 0.002
Parzen	0.249 ± 0.003	0.759 ± 0.003	0.751 ± 0.003	0.751 ± 0.003
Regressão Logística	0.062 ± 0.002	0.939 ± 0.002	0.938 ± 0.002	0.938 ± 0.002

distribuição normal das medidas, o classificador de regressão logística destacou-se com o desempenho mais elevado, seguido pelo método dos K-vizinhos, pelo Bayesiano Gaussiano e, por fim, pela Janela de Parzen. Contudo, apenas os classificadores bayesiano gaussiano e janela de parzen obtiveram métricas melhores para o classificador combinado quando comparado a seus modelos individuais treinados com apenas uma das bases de dados. O classificador com a melhor métrica foi o de regressão logística treinado na base de dados fac.

4.4 Testes de Significância Estatística

A utilização de testes estatísticos visa comparar os classificadores para verificar a presença de diferenças significativas entre eles. Neste trabalho, aplicamos o teste de Friedman e o subsequente pós-teste de Nemenyi em quatro grupos de classificadores. Cada grupo corresponde a um tipo de classificador — Bayesiano Gaussiano, K-Vizinhos, Janela de Parzen e Regressão Logística — com quatro membros: um treinado em cada uma das bases de dados (fac, fou e zer) e um quarto membro representando o resultado da regra do voto majoritário. Um quinto grupo é composto somente pelos classificadores de voto majoritário. A Tabela 16 apresenta de maneira visual todos os modelos e quais compõem cada um dos grupos.

As hipóteses estatísticas foram estabelecidas da seguinte forma:

- H_0: Não existe diferença estatística significativa entre os grupos de classificadores.

Tabela 16. *Representação dos 5 grupos utilizados para os testes de significância estatística*

Group 1	Group 2	Group 3	Group 4	Combined group
bayesian gaussian (fac)	bayesian knn (fac)	bayesian parzen (fac)	logistic regression (fac)	combined bayesian gaussian
bayesian gaussian (fou)	bayesian knn (fou)	bayesian parzen (fou)	logistic regression (fou)	combined bayesian knn
bayesian gaussian (zer)	bayesian knn (zer)	bayesian parzen (zer)	logistic regression (zer)	combined bayesian parzen
combined bayesian gaussian	combined bayesian knn	combined bayesian parzen	combined logistic regression	combined logistic regression

- H_1: Existe uma diferença estatística significativa entre os grupos de classificadores.

O valor com significância 5% foi definido de modo que resultados com $p\ value \leq 0.05$ tinham a hipótese nula (H_0) rejeitada.

4.4.1 Teste de Friedman

O teste de Friedman é uma ferramenta estatística não paramétrica utilizada para detectar diferenças nas distribuições de amostras repetidas. Implementado na biblioteca SciPy através da função `friedmanchisquare`, este teste compara a consistência entre múltiplos classificadores aplicados a diferentes conjuntos de dados e foi aplicado para todas as métricas de análise (taxa de erro, precisão, cobertura, F-measure). O teste verifica a hipótese nula (H_0) de que não há diferença significativa entre os classificadores do grupo analisado, fornecendo um *p value* que, sob a premissa de uma distribuição qui-quadrado, é considerado confiável para amostras com $n > 10$ e mais de seis amostras repetidas.

O teste foi aplicado aos 5 grupos, e todos os 5 grupos tiveram resultados com $p\ value \ll 0.00001$ para todas as métricas. Ou seja, todos os 5 grupos rejeitaram H_0, o que indica que existe uma diferença estatística significativa entre os grupos de classificadores.

4.4.2 Pós-teste de Nemenyi

Após a aplicação do teste de Friedman, utilizamos o pós-teste de Nemenyi para realizar comparações emparelhadas entre classificadores de cada grupo. Este teste estatístico é conduzido quando resultados significativos são obtidos pelo teste de Friedman. O teste é fundamental para avaliar dois a dois as diferenças entre os classificadores de cada grupo. A implementação do pós-teste foi feita a partir da função `posthoc_nemenyi_friedman` da biblioteca Scikit-Posthocs. Dentre todos os pares analisados apenas o par formado pela regressão logística treinada nas bases 'fou' e 'zer' não demonstrou diferença significativa (p-$value > 0.05$). Todos os demais pares de todos os grupos tiveram resultados com $p\ value \ll 0.05$, rejeitando H_0.

4.4.3 Resultados da análise de significância estatística

Já que as 3 bases de dados representam as mesmas 10 classes através de um conjunto de *features* diferentes, os resultados para os grupos de 1 a 4 indicam que diferentes estratégias de extração de *features* resultarão em modelos diferentes (com exceção de um único par de modelos do grupo de regressão logística). Além disso, também indicaram que o metamodelo construído a

partir da regra do voto majoritário resultará em um modelo com comportamento diferente dos demais, contudo, esse modelo resultante não é necessariamente melhor do que o melhor dos modelos individuais. Já o resultado do 5º grupo indica que os modelos finais resultantes do mesmo *pipeline* de treinamento e regra do voto majoritário possuíram resultados diferentes para cada tipo de classificador.

5 Conclusão

Na seção 3 foi implementado o algoritmo não supervisionado KFCM-K-W.2. O hiperparâmetro m de *fuzzificação* foi variado a fim de encontrar o valor que geraria melhores métricas. Foi visto que $m = 1.3$ obteve os melhores resultados com valores de acurácia de 0.7230 e índice Rand corrigido de 0.5520 na iteração com a menor função objetivo J para a base *mfeat-fac*. A base *mfeat-fac* obteve resultados significativamente melhores que as bases *mfeat-fou* e *mfeat-zer*, indicando que a extração de atributos de correlações de perfil seria mais adequada na representação dos numerais da base *multiple features* em relação aos outros métodos no caso da aplicação do algoritmo KFCM-K-W.2.

Na seção 4 vários métodos supervisionados foram utilizados e comparados. Os algoritmos combinados pela regra do voto majoritário foram melhores do que os individuais para os classificadores bayesiano gaussiano e janela de Parzen, já para os classificadores de regressão logística e K-vizinhos, os modelos combinados tiveram resultados piores do que a melhor versão individual treinada em apenas uma das bases. Os melhores resultados foram obtidos com o algoritmo de regressão logística, que foi o melhor modelo combinado resultante do voto majoritário para todas as 4 métricas. Já o melhor modelo geral em relação às 4 métricas foi o modelo de regressão logística treinado na base de dados *mfeat-fac*.

Apesar de treinados e avaliados com conjuntos diferentes, completo para o *fuzzy* e diversos 10-*folds* para o KNN, consideramos que ambos obtiveram bons resultados, o que sugere competitividade entre as abordagens supervisionadas e não-supervisionadas para problemas multiclasses e precisaríamos de mais testes para definir a melhor para nosso *dataset*.

No geral, os métodos supervisionados obtiveram melhores resultados com as três bases utilizadas: *mfeat-fac*, *mfeat-fou* e *mfeat-zer*. Porém, os métodos não-supervisionados, como o KFCM-K-W.2 no contexto desse trabalho, possuem a vantagem de não necessitar das classes *a priori* das instâncias. A não necessidade das classes *a priori* possui implicações importantes em diversas aplicações, como sistemas de recomendação, busca, e detecção de padrões. Ambos os métodos supervisionado e não-supervisionado possuem vantagens e desvantagens em suas respectivas áreas de aplicação.

Referências

COURNAPEAU, D. **Scikit-learn: Machine Learning in Python**. 2023. Versão 1.3.1.

Disponível em: <http://scikit-learn.org/>.

DUA, D.; GRAFF, C. **UCI Machine Learning Repository**. 2017. Disponível em: <http://archive.ics.uci.edu/ml>.

DUIN, R. **Multiple Features**. 1998. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5HC70.

PARZEN, E. On estimation of a probability density function and mode. **The Annals of Mathematical Statistics**, Institute of Mathematical Statistics, v. 33, n. 3, p. 1065–1076, 1962.

RASCHKA, S. **Kernel density estimation via the Parzen-Rosenblatt window method**. 2014. Acesso em: 2023-11-06. Disponível em: <https://sebastianraschka.com/Articles/2014_kernel_density_est.html>.

ROSENBLATT, M. Remarks on some nonparametric estimates of a density function. **The Annals of Mathematical Statistics**, Institute of Mathematical Statistics, v. 27, n. 3, p. 832–837, 1956.

SIMõES, E. C.; CARVALHO, F. de A. T. de. Gaussian kernel fuzzy c-means with width parameter computation and regularization. **Pattern Recognition**, Elsevier, v. 143, n. 109749, 2023.