Universidade do Minho

Mestrado em Matemática e Computação

Otimização em *Machine Learning*

Classificador logístico multiclasse: OvA vs ECOC

Eduardo Teixeira Dias (PG52249)

Hugo Filipe de Sá Rocha (PG52250)

Simão Pedro Batista Caridade Quintela (PG52257)

Tiago Augusto Lopes Monteiro (PG52258)

Diogo Filipe Santos Lima Rosário (PG49174)

30 de maio de 2024

Conteúdo

Capítulo 1

Introdução

No âmbito da Unidade curricular Otimizacão em *Machine Learning*, do perfil de *Machine Learning* do Mestrado em Matemática e Computação da Universidade do Minho, foi realizado um trabalho prático com o intuito de realizar a análise e implementação do Classificador logístico multiclasse: *OvA vs ECOC* com uma parte de programação executada em linguagem *Python* e de experiência numérica.

Neste projeto comparou-se as duas técnicas *one-vs-all (OvA)* e *Error-Correcting Output Codes (ECOC)* do classificador logístico multi-classe nas versões primal e dual com e sem *kernel* polinomial utilizando métricas de avaliação e ainda *plots* das bases de dados para mais fácil compreensão do problema em questão. No que toca às bases de dados experimentadas, experimentaram-se bases de dados linearmente e não linearmente separáveis, bases de dados simples e uma mais complexa, de forma a poder comparar e avaliar os resultados com a maior fiabilidade possível. Como métricas de avaliação utilizaram-se a *accuracy* e a matriz de confusão. Além disso, é possível solicitar em cada versão dos algoritmos o *plot* de um gráfico do erro ao longo das iterações bem como o *plot* do classificador em questão.

Capítulo 2

Classificadores Logísticos Multiclasse

2.1 Método OvA - One versus All

Em problemas de classificação multiclasse, onde existem mais de duas classes possíveis para os dados, uma abordagem comum é utilizar o método *One versus All* (OvA), também conhecido como *One versus Rest* (OvR). Este método transforma o problema multiclasse em vários problemas binários.

2.1.1 Descrição do Método

No método *One versus All*, para uma tarefa de classificação com K classes, são treinados K classificadores binários. Cada classificador é responsável por distinguir uma classe específica k ($k = 1, 2, \ldots, K$) contra todas as outras classes combinadas. O procedimento pode ser descrito da seguinte maneira:

1. Para cada classe k, criamos um conjunto de dados binário onde os exemplos da classe k são rotulados como positivo ($+1$) e todos os outros exemplos são rotulados como negativo (-1).

2. Treinamos um classificador binário utilizando este conjunto de dados.

3. Repetimos o processo para cada uma das K classes, resultando em K classificadores binários.

Durante as *predicts*, uma nova amostra é passada para cada um dos K classificadores, e a classe com a maior pontuação é selecionada como a classe final para essa amostra.

2.1.2 Vantagens e Desvantagens

Vantagens

- **Simplicidade**: A implementação do método OvA é relativamente simples e direta.
- **Escalabilidade**: Permite o uso de classificadores binários convencionais que podem ser mais bem compreendidos e otimizados.

Desvantagens

- **Desequilíbrio de Classes**: O conjunto de dados binário gerado pode ser altamente desbalanceado, especialmente se uma classe for muito menor ou maior do que as outras.
- **Competição entre Classificadores**: Pode haver conflito quando mais de um classificador binário dá uma pontuação alta para a mesma amostra. A decisão final pode ser complexa, dependendo de como as pontuações são comparadas.

2.1.3 Aplicações e Exemplos

O método *One versus All* tem várias aplicações, como reconhecimento de imagem, processamento de linguagem natural e bioinformática. Por exemplo, num problema de reconhecimento de dígitos, um classificador OvA poderia ser treinado para distinguir cada dígito $(0, 1, \ldots, 9)$ de todos os outros.

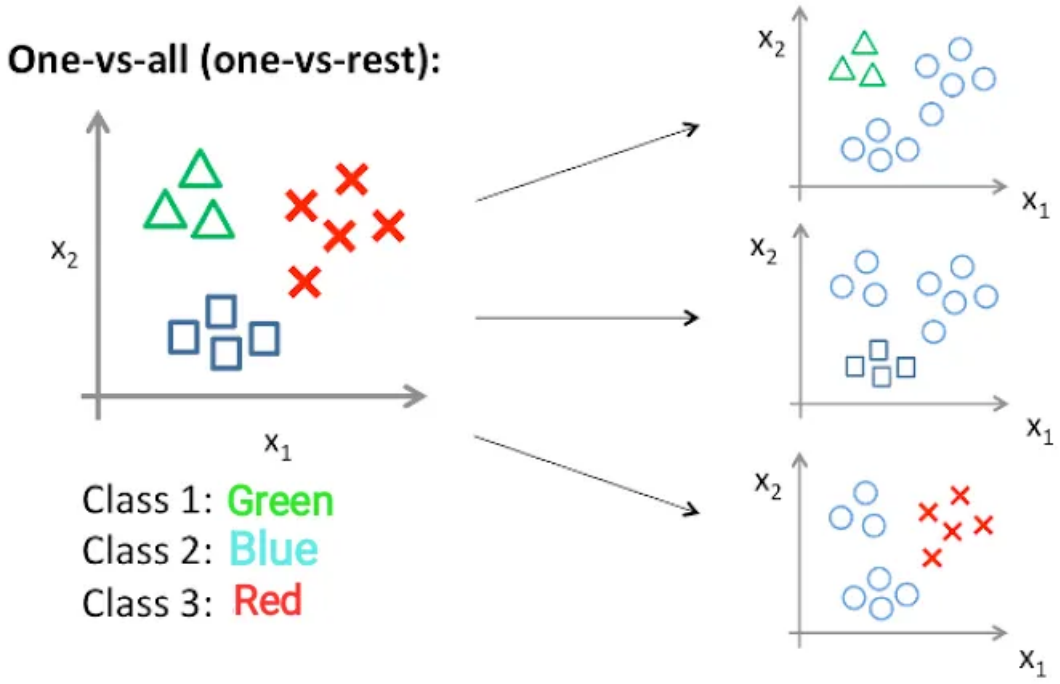

Figura 2.1: Método *One versus All*

Em suma, o método *One versus All* é uma estratégia eficaz para abordar problemas de classificação multiclasse, aproveitando a simplicidade dos classificadores binários para construir modelos capazes de lidar com múltiplas classes.

2.2 Método ECOC - Error Correcting Output Codes

O método Error Correcting Output Codes (ECOC) é uma técnica popular para resolver problemas de classificação multiclasse. Ao contrário do método One versus All, que treina múltiplos classificadores binários independentes, o ECOC cria um conjunto de classificadores binários inter-relacionados.

2.2.1 Descrição do Método

No ECOC, cada classe é representada por um código binário exclusivo. Em vez de treinar um único classificador para cada classe, são treinados vários classificadores binários, cada um projetado para distinguir entre diferentes combinações de classes com base em seus códigos binários.

O processo pode ser descrito da seguinte maneira:

1. **Codificação das Classes**: A cada classe é atribuída a um código binário, geralmente representado por uma matriz onde as linhas correspondem às classes e as colunas correspondem aos bits do código binário.
2. **Treino dos Classificadores**: Para cada coluna do código binário, um classificador binário é treinado para distinguir entre as classes que têm um valor de 0 ou 1 naquela posição.
3. **Descodificação das Classes**: Durante a fase de teste, a saída de todos os classificadores binários é combinada de acordo com o código binário de cada classe. O classificador final é aquele que corresponde ao código binário mais próximo à saída combinada.

2.2.2 Vantagens e Desvantagens

Vantagens

- **Redundância e Tolerância a Erros**: O ECOC pode corrigir erros na classificação através da redundância presente nos códigos binários.
- **Flexibilidade na Representação das Classes**: As classes podem ser representadas de forma flexível por códigos binários, permitindo uma adaptação precisa às características do problema.

Desvantagens

- **Complexidade do Treino**: O treino de múltiplos classificadores binários inter-relacionados pode ser computacionalmente pesado e exigir grandes conjuntos de dados.
- **Sensibilidade à Escolha do Código Binário**: A eficácia do ECOC pode depender significativamente da escolha adequada dos códigos binários para representar as classes.

2.2.3 Aplicações e Exemplos

O método ECOC é amplamente utilizado em vários campos, como reconhecimento de padrões, bioinformática e processamento de sinais. Por exemplo, num problema de classificação de espécies de animais com várias classes, o ECOC pode ser usado para construir um sistema robusto capaz de lidar com erros de classificação e fornecer previsões precisas.

	Code Word														
Class	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
0	1	1	0	0	0	0	1	0	1	0	0	1	1	0	1
1	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0
2	1	0	0	1	0	0	0	1	1	1	1	0	1	0	1
3	0	0	1	1	0	1	1	1	0	0	0	0	1	0	1
4	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1
5	0	1	0	0	1	1	0	1	1	1	0	0	0	0	1
6	1	0	1	1	1	0	0	0	0	1	0	1	0	0	1
7	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1
8	1	1	0	1	0	1	1	0	0	1	0	0	0	1	1
9	0	1	1	1	0	0	0	0	1	0	1	0	0	1	1

Figura 2.2: Códigos gerados no método ECOC

Em resumo, o método ECOC é uma abordagem poderosa e versátil para resolver problemas de classificação multiclasse, oferecendo redundância e tolerância a erros através da combinação de classificadores binários inter-relacionados.

Capítulo 3

Metodologia

3.1 Formulação Primal com o método OvA

3.1.1 Fórmulas

1.
$$\hat{y} = \begin{cases} 1, & \text{se } \hat{p} > \epsilon_{CLog} \\ 0, & \text{se } \hat{p} \leq \epsilon_{CLog} \end{cases}$$

2.
$$L(\bar{w}; D) = \prod_{n=1}^{N} \begin{cases} \hat{p}^n, & \text{se } y^n = 1 \\ (1 - \hat{p}^n), & \text{se } y^n = 0 \end{cases} = \prod_{n=1}^{N} (\hat{p}^n)^{y^n} (1 - \hat{p}^n)^{1-y^n}$$

3.1.2 Função de Custo

1.
$$E(\bar{w}; D) = \frac{1}{N} \sum_{n=1}^{N} E_n(\bar{w}; x^n, y^n),$$

2.
$$E_n(\bar{w}; x^n, y^n) = -y^n \ln(\hat{p}^n) - (1 - y^n) \ln(1 - \hat{p}^n)$$

3.2 Formulação Dual com o método OvA

3.2.1 Introdução

Na formulação primal, os parâmetros w e b são diretamente otimizados. Na formulação dual, utilizamos os multiplicadores de Lagrange para transformar o problema de otimização, permitindo a aplicação eficiente de kernels para problemas não linearmente separáveis.

3.2.2 Fórmulas

$$\hat{p} \equiv h(x; \alpha) = \sigma \left(\sum_{n=1}^{N} \alpha_n (\bar{x}^n \cdot \bar{x}) \right), \alpha \in \mathcal{H}.$$

3.2.3 Função de Custo

1.
$$E(\alpha; D) = \frac{1}{N} \sum_{n=1}^{N} E_n(\alpha; x^n, y^n)$$

2.
$$E_n(\alpha; x^n, y^n) = -y^n \ln(\hat{p}^n) - (1 - y^n) \ln(1 - \hat{p}^n)$$

3.3 Formulação Dual com *kernel* polinomial

3.3.1 Introdução

Para capturar relações não lineares nos dados, utilizamos a função kernel polinomial. Este kernel permite mapear os dados para um espaço de características de dimensão superior, onde é mais provável que sejam linearmente separáveis.

3.3.2 Fórmulas

1.
$$\hat{p} \equiv h(x; \alpha) = \sigma \left(\sum_{n=1}^{N} \alpha_n K(\bar{x}^n, \bar{x}) \right), \alpha \in \mathcal{H}.$$

2.
$$K(\bar{x}^n, \bar{x}) = (\bar{x}^n \cdot \bar{x})^d$$

Capítulo 4

Aplicação dos diferentes métodos

Depois de terem sido criados os modelos previamente apresentados, recorremos a nova bases de dados para averiguar a sua qualidade e eficácia de classificação:

4.1 Bases de dados

As bases de dados utilizadas foram as bases de dados que continham as tabelas de verdade dos operadores AND, OR e XOR, as bases de dados fornecidas pelo docente do ex1 ao ex6 (utilizadas para validação de cada classificador logístico binário implementado) e, 2 bases de dados criadas por nós, as quais contêm 16 observações de treino e 15 de teste e que apelidamos de Multiclass1 e Multiclass2, respetivamente.

4.1.1 Multiclass1 e Multiclass2

Estas duas bases de dados foram criadas no intuito de testar os modelos multi-classe desenvolvidos, tanto o OvA como o ECOC. No caso da Multiclass1, é uma base de dados que diz respeito aos quadrantes de um referencial ortonormado, isto é, tem 4 classes onde a primeira, segunda, terceira e quarta classe contém pontos no primeiro, segundo, terceiro e quarto quadrante respetivamente, no caso, 4 pontos em cada um deles para treino. Trata-se de uma base de dados simples onde cada uma das classes (individualmente) é linearmente separável das outras.

Relativamente à base de dados Multiclass2, é uma extensão da base de dados Multiclass1 onde é introduzida uma quinta classe que se situa por volta da origem de um referencial ortonormado, também com 4 pontos para treino. Trata-se de uma base de dados simples, mas ligeiramente mais desafiadora devido ao facto de essa quinta classe adicionada (individualmente) não ser linearmente separável das restantes.

Tabela 4.1: Resultados obtidos organizados por accuracy decrescente

Modelo OVA	**d**	**Base de Dados**	**accuracy**
CLogD_MGB	-	Multiclass1	1
CLogDKP_MGB	2	Multiclass1	0.6
CLogDKP_MGB	3	Multiclass1	0.6
CLogD_MGB	-	Multiclass2	0.8
CLogDKP_MGB	2	Multiclass2	0.466
CLogDKP_MGB	3	Multiclass2	0.466

Tabela 4.2: Resultados obtidos organizados por accuracy decrescente

Modelo ECOC	**d**	**Base de Dados**	**accuracy**
CLogD_MGB	-	Multiclass1	1
CLogDKP_MGB	2	Multiclass1	0.933
CLogDKP_MGB	3	Multiclass1	0.933
CLogDKP_MGB	2	Multiclass2	0.8
CLogDKP_MGB	3	Multiclass2	0.8
CLogD_MGB	-	Multiclass2	0.666

Como podemos observar nos resultados, em ambos os classificadores OvA e ECOC, foi possível obter bons resultados de *accuracy* em ambas as bases de dados. Como ambas as bases de dados eram relativamente simples optámos por testar nas mesmas as versões *dual* dos algoritmos que, em geral, são mais dispendiosas em termos de tempo em relação à versão primal, problema que não se coloca neste tipo de base de dados. Conclui-se de forma clara que os modelos que utilizam *kernel* saíram-se bem melhor no classificador ECOC do que no OvA, onde parecem estar *overfitted*. De notar que, na grande maioria dos classificadores binários intermédios tanto no OvA como no ECOC, apresentaram uma *accuracy* de 100% nos dados de treino.

4.1.2 MNIST

De forma a verificar se os classificadores conseguiriam manter as suas performances caso fosse utilizada uma base de dados mais complexa decidimos utilizar o dataset MNIST, o qual é uma base de dados de dígitos escritos à mão. Este é composto por 60 000 observações para treino e 10 000 para teste, sendo que cada uma delas representa um dígito de 0 a 9 em Grayscale (escala de cinza).

Por questões de tempo e eficiência reduzimos o conjunto de treino para 540 observações (54 de cada dígito) e o conjunto de teste para 80 observações (8 de cada dígito).

4.2 Resultados do *dataset* MNIST com o classificador OvA

Tabela 4.3: Resultados obtidos organizados por accuracy decrescente

Modelo	**d**	**accuracy**	**tempo de execução**
CLog-MGmB (tamanho da *batch* fixo a 64)	-	0.75	2min49s
CLog-MGB	-	0.725	2min31s
CLog-MGmB (tamanho da *batch* aleatório)	-	0.7125	2min52s
CLogD-MGB	-	0.675	64min10s
CLog-MGE	-	0.6375	22min24s
CLog-MGE-sequencial	-	0.6125	26min50s
CLogDKd-MGB	3	0.4	33min17s
CLogDKd-MGE	2	0.25	24min44s

Analisando os resultados, o modelo que obteve a melhor *accuracy* foi o modelo *gradiente mini batch* com uma *batch* de tamanho fixo (64), com uma *accuracy* de 0.75. O resultado parece ser bom devido ao facto de se tratar de um modelo primal e com tempo de execução rápido. De seguida, podemos ver novamente 2 modelos primais desta vez o modelo *gradiente batch* e o *gradiente mini batch* mas de tamanho aleatório. Conclui-se portanto que uma *batch* de tamanho fixo apresenta uma ligeira vantagem em relação a um tamanho de *batch* aleatório no método CLog-MGmB. Além disso, o método do gradiente estocástico também parece trazer uma ligeira vantagem em relação ao método do gradiente estocástico mas sequencial, onde a base de dados é percorrida sequencialmente. Por outro lado, os modelos que utilizam *kernel*, além de serem mais dispendiosos em termos de tempo computacional, não trouxeram vantagens em termos de *accuracy* e parecem estar *overfitted*, isto porque, após fornecer os dados de teste ao classificador, a maior parte das previsões erradas o modelo não conseguia atribuir uma probabilidade alta a nenhuma das classes.

4.3 Resultados do *dataset* MNIST com o classificador ECOC

Tabela 4.4: Resultados obtidos organizados por accuracy decrescente

Modelo	**d**	**accuracy**	**tempo de execução**
CLog-MGB	-	0.6875	31min47s
CLog-MGmB (tamanho da *batch* fixo a 64)	-	0.6625	19min23s
CLog-MGE-sequencial	-	0.65	34min24s
CLog-MGE	-	0.625	33min12s

Nota: Notar que, a tabela de códigos utilizada para a base de dados MNIST, foi a tabela presente na Figura 2.2.

Ao analisar os resultados percebemos que, no ECOC, os modelos primais não se saíram tão bem e o tempo de execução aumentou significativamente em relação ao OvA, devido, presumivelmente, ao aumento do número de classificadores (10 para 15) e devido, a isso, decidimos testar todos os modelos primais mas tornou-se inpraticável em termos de tempo testar as restantes versões dos algoritmos (*dual* com e sem *kernel*).

Bibliografia

[1] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[2] H. Daumé. *A Course in Machine Learning*. Hal Daumé III, 2017.

[3] Thomas G Dietterich and Ghulum Bakiri. Error-correcting output codes: A general method for improving multiclass inductive learning programs. In *The Mathematics of Generalization*, pages 395–407. CRC Press, 2018.

[4] Sarit Khirirat, Hamid Reza Feyzmahdavian, and Mikael Johansson. Mini-batch gradient descent: Faster convergence under data sparsity. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2880–2887. IEEE, 2017.

[5] Stephen Marsland. *Machine Learning - An Algorithmic Perspective.* Chapman and Hall / CRC machine learning and pattern recognition series. CRC Press, 2009.

[6] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms.* Cambridge University Press, 2014.

[7] Masashi Sugiyama. *Introduction to Statistical Machine Learning*. Morgan Kaufmann, Amsterdam, 2016.

Apêndice A

Classificador binário base de dados XOR

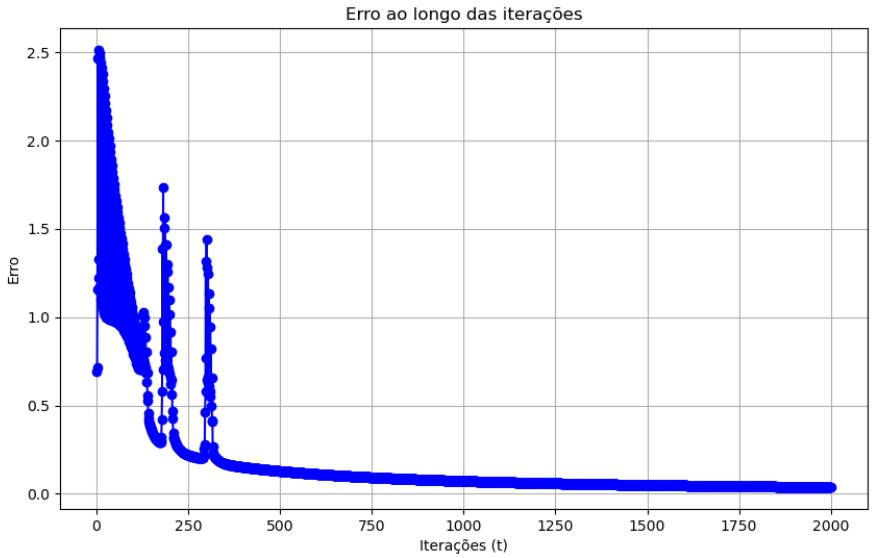

Figura A.1: Exemplo de um gráfico de erro: CLogDKP2_MGB, XOR

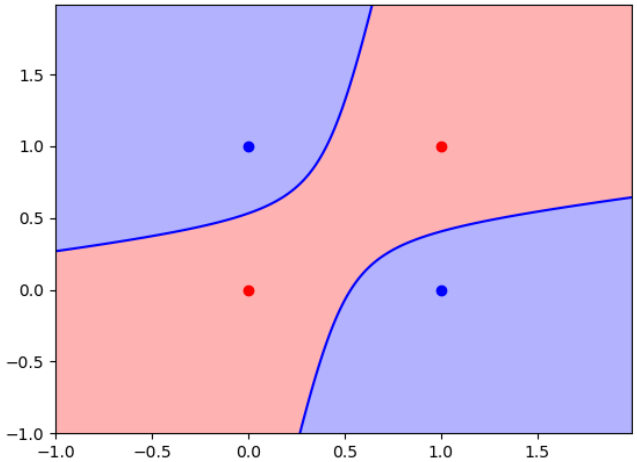

Figura A.2: Exemplo de um classificador binário: CLogDKP2_MGB, XOR

Apêndice B

Bases de dados Multiclasse 1 e 2

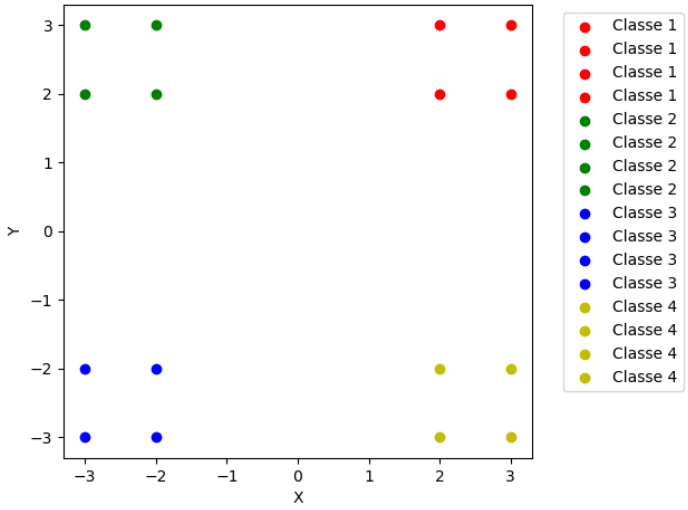

Figura B.1: Conjunto de treino da base de dados Multiclass1

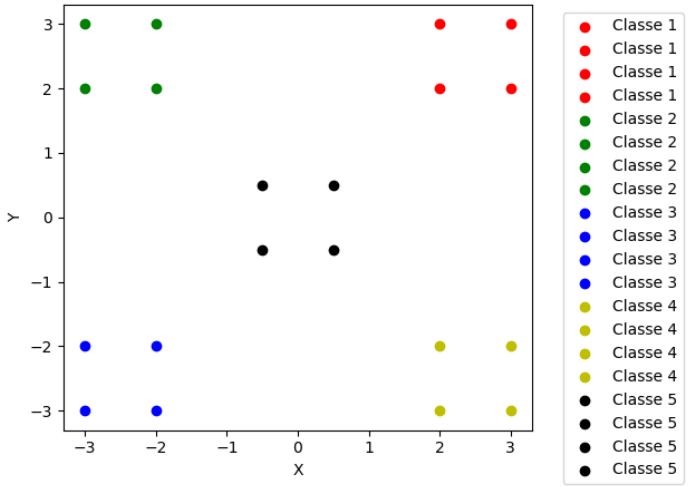

Figura B.2: Conjunto de treino da base de dados Multiclass2

Apêndice C

ECOC base de dados Multiclass1 e 2

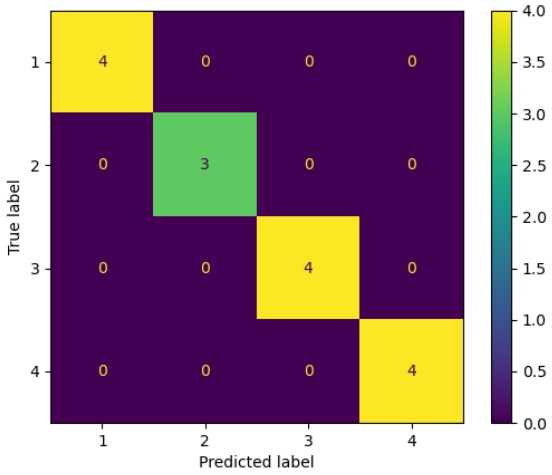

Figura C.1: Matriz de confusão ECOC com CLogD_MGB com a BD Multiclass1

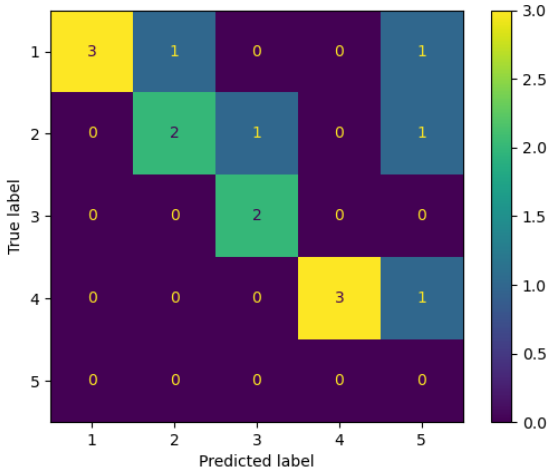

Figura C.2: Matriz de confusão ECOC com CLogD_MGB com a BD Multiclass2

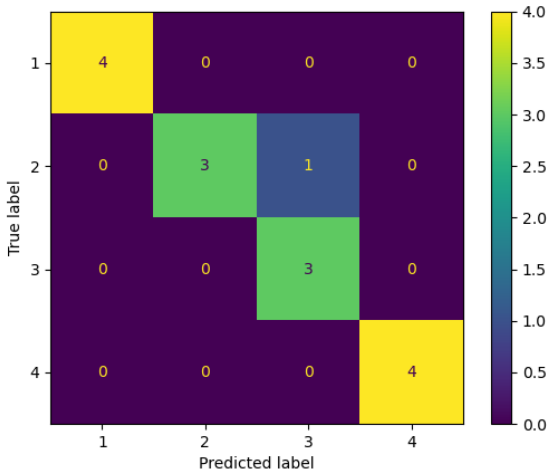

Figura C.3: Matriz de confusão ECOC com CLogDKP2_MGB com a BD Multiclass1

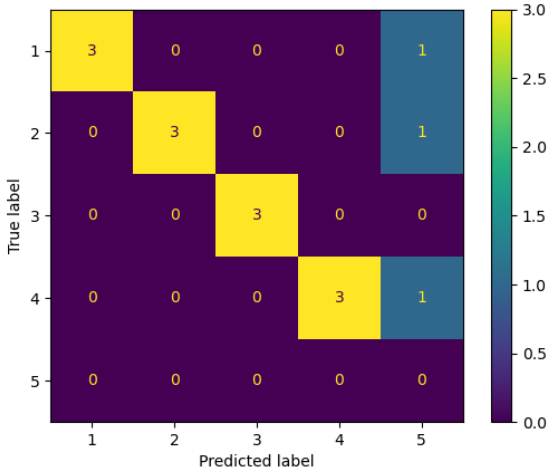

Figura C.4: Matriz de confusão ECOC com CLogDKP2_MGB com a BD Multiclass2

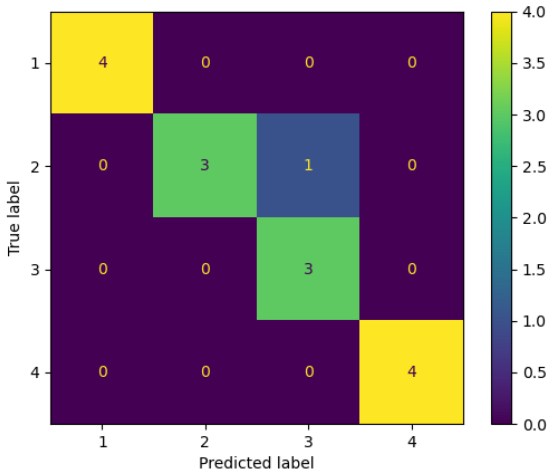

Figura C.5: Matriz de confusão ECOC com CLogDKP3_MGB com a BD Multiclass1

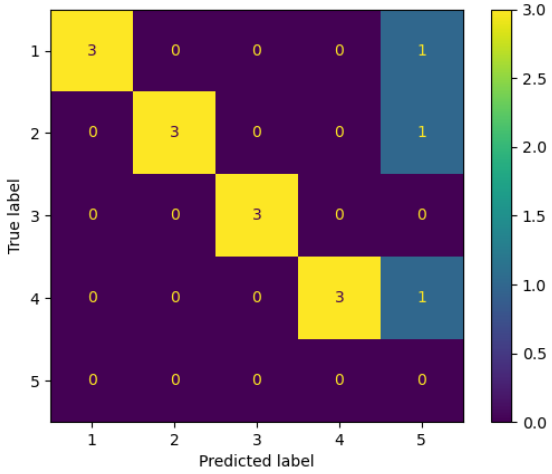

Figura C.6: Matriz de confusão ECOC com CLogDKP3_MGB com a BD Multiclass2

Apêndice D

OVA base de dados Multiclass1 e 2

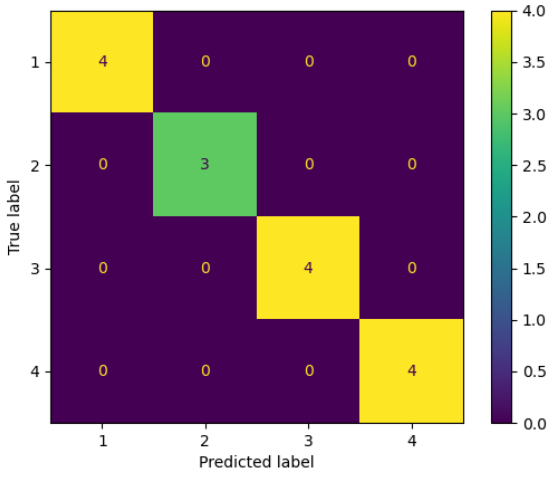

Figura D.1: Matriz de confusão OVA com CLogD_MGB com a BD Multiclass1

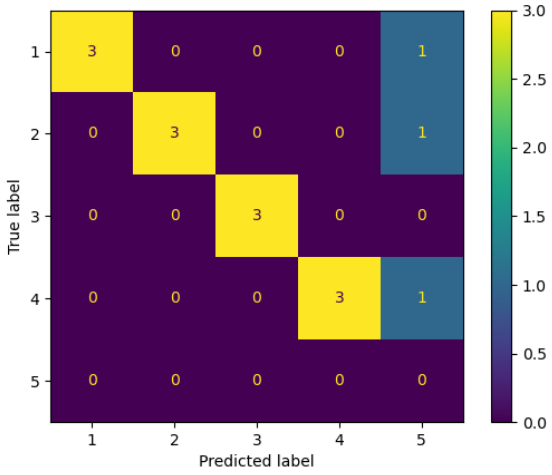

Figura D.2: Matriz de confusão OVA com CLogD_MGB com a BD Multiclass2

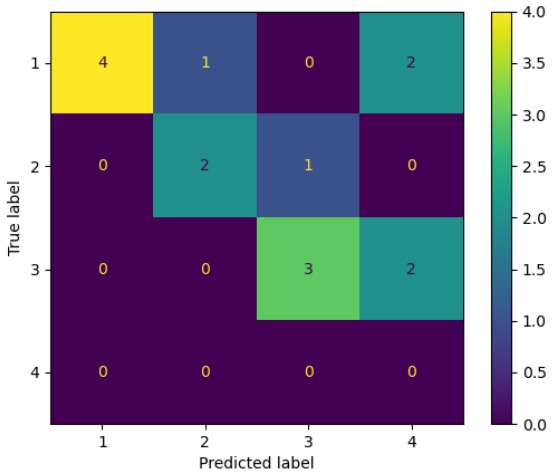

Figura D.3: Matriz de confusão OVA com CLogDKP2_MGB com a BD Multiclass1

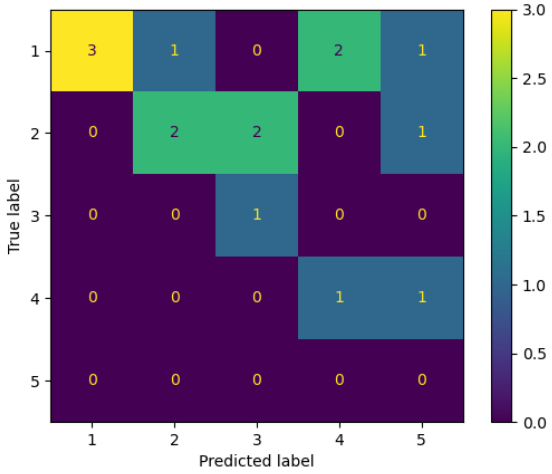

Figura D.4: Matriz de confusão OVA com CLogDKP2_MGB com a BD Multiclass2

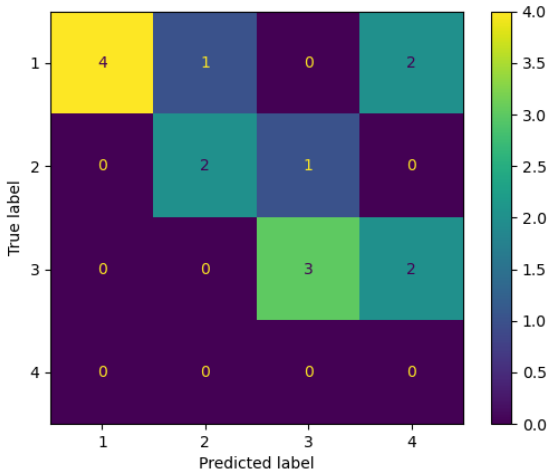

Figura D.5: Matriz de confusão OVA com CLogDKP3_MGB com a BD Multiclass1

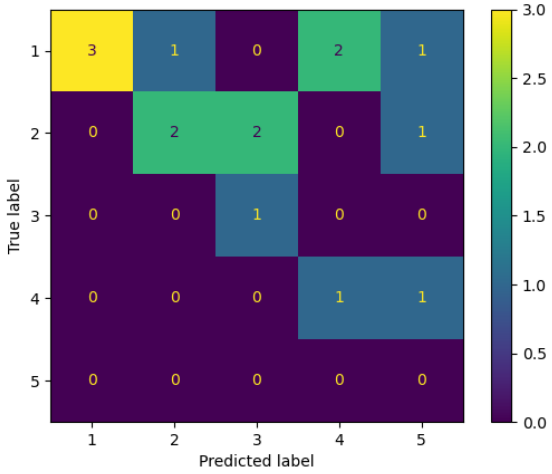

Figura D.6: Matriz de confusão OVA com CLogDKP3_MGB com a BD Multiclass2