Universidade do Minho

Otimização em Machine Learnig - Trabalho Prático (Grupo nº1)

Classificador logístico multiclasse: OvA vs ECOC

Autores:

- Alexandre Abreu (PG55612)
- Ana Oliveira (PG55613)
- André Veiga (PG55614)
- Beatriz Marques (PG57743)

Data:
10 de julho de 2025

Conteúdo

Introdução

A classificação multiclasse representa um dos desafios fundamentais na aprendizagem automática, onde o objetivo consiste em atribuir instâncias a uma de várias classes possíveis (três ou mais). Enquanto muitos algoritmos de classificação foram originalmente concebidos para problemas binários, como a regressão logística, a sua aplicação a cenários multiclasse exige estratégias específicas de decomposição. Entre as abordagens mais comuns destacam-se:

1. **One-vs-All (OvA):** Divide o problema multiclasse em múltiplos problemas binários, onde cada classificador é treinado para distinguir uma classe específica contra todas as outras. Apesar da sua simplicidade conceptual, esta abordagem pode enfrentar desafios quando existe desequilíbrio entre classes.

2. **Error-Correcting Output Codes (ECOC):** Utiliza codificação binária para representar classes, permitindo não só a decomposição do problema multiclasse como também incorporando redundância para correção de erros. Esta abordagem tende a ser mais robusta, especialmente em cenários complexos.

Este trabalho foca-se na implementação comparativa destas duas estratégias, considerando tanto a versão primal (linear) como a versão dual com *kernel* polinomial, que permite lidar com problemas não linearmente separáveis através do mapeamento para espaços de maior dimensão. Adicionalmente, serão exploradas técnicas de otimização computacional, nomeadamente variações do método *mini-batch*, com o objetivo de balancear eficiência e desempenho.

Capítulo 1

Metodologia

1.1 ECOC e OvA

Os métodos **One-vs-All (OvA)** e **Error-Correcting Output Codes (ECOC)** são estratégias para estender classificadores binários a problemas multiclasse.

- **One-vs-All (OvA):**
  - **Funcionamento:** Treina um classificador binário para cada classe, onde uma classe é considerada positiva e todas as outras são tratadas como negativas. Durante a predição, o classificador com a maior probabilidade ou confiança determina a classe final.
  - **Aplicações:** Ideal para problemas com um número moderado de classes, devido à sua simplicidade e eficiência computacional.
- **Error-Correcting Output Codes (ECOC):**
  - **Funcionamento:** Utiliza uma matriz de códigos (codificação binária) para representar cada classe. Cada coluna da matriz corresponde a um classificador binário, treinado para distinguir entre grupos de classes. A predição é feita comparando as saídas dos classificadores com os códigos das classes, escolhendo a classe com menor distância de *Hamming*.
  - **Aplicações:** Adequado para problemas complexos com muitas classes, pois a matriz de códigos pode corrigir erros individuais dos classificadores binários.

Ambos os métodos foram implementados na versão primal, que resolve o problema de classificação diretamente no espaço original dos dados, é eficaz quando os dados são linearmente separáveis ou quando a dimensionalidade não é excessivamente alta. Tal como numa versão dual com kernel, ambas utilizando regressão logística com gradiente descendente.

1.2 ECOC

O primeiro passo na implementação da técnica de **Error-Correcting Output Codes** é avaliar as base de dados com que vamos trabalhar, neste projeto vamos inicialmente abordar algumas bases sintéticas com as mesmas características, ou seja, vamos ter 3 classes diferentes representadas através de dois atributos. Após esta avaliação, podemos começar a criar a nosssa matriz onde cada classe irá ser representada por um código binário.

Seja uma matriz de codificação $M \in \{-1, +1\}^{K \times L}$, onde:

- K é o número de classes no problema;
- L é o número de classificadores binários (colunas da matriz);
- Cada linha M_k representa o código associado à classe k;
- Cada coluna $M_{:,j}$ define um problema binário distinto.

Uma vez que só vamos lidar com 3 classes, o processo para escolher os respetivos códigos para cada classe foi simples, sendo que a regra geral mais importante é não repetir um código para classes diferentes. No nosso caso, a matriz que foi escolhida foi a seguinte:

Classe	**Clf 1**	**Clf 2**	**Clf 3**
C_1	0	1	1
C_2	1	0	1
C_3	1	1	0

O passo seguinte passa por treinar os classificadores binários, neste caso cada classificador vai estar associado a um código, que é representado pelas colunas da matriz, os números desse código vão servir como labels para os elementos de cada classe. No casso do classificador 1, os elementos da primeira classe vão ser representados pela label 0 e os restantes por 1 durante o processo de treino. Este treino vai ser feito através de regressão logistica binária que vamos abordar mais adiante no relatório.

Para um elemento x da base de dados, obtemos o vetor de saída dos classificadores:

$$s(x) = [clf_1(x), clf_2(x), \ldots, clf_L(x)]$$

A classe escolhida é aquela cuja linha da matriz M_k está mais próxima de $s(x)$, com base numa medida de distância, como:

$$\hat{y} = \arg\min_k \; d(s(x), M_k)$$

Onde d vai ser a distância de Hamming.

1.3 OvA

Na técnica **One vs All** o processo é mais simples, vamos apenas criar um classificador binário para cada classe. A principal diferença destes classificadores para os do ECOC vai estar na forma como atribuímos uma label a cada elemento no processo de treino.

Antes de começar o treino de um classificador k, vamos atribuir a label 1 aos elementos da classe k, e aos restantes o valor 0, dai a origem do nome **One vs All**. No final do treino, um classificador vai apenas tentar prever se um elemento é da classe k ou não, não o vai tentar atribuir a mais nenhuma outra classe, simplesmente avalia se esse elemento se enquadra na classe k, e com que confiança fez essa previsão. O processo de treino vai ser o mesmo usado para os classificadores do **ECOC**.

1.4 Método de Treino

O método utilizado ao longo deste projeto para treinar os classificadores foi o de Regressão Logistica. O nosso objetivo é que cada classificador estime uma probabilidade para cada elemento x, que vai ser utilizada para escolher a classe desse elemento, no caso do **ECOC** esse valor estimado vai ser transformado numa das labels 0 ou 1 conforme o seu valor, de forma a obtermos um código final, já no **OvsA** vamos manter o valor da probabilidade obtido e escolher o mais alto entre os classificadores.

1.4.1 Regressão Logística Binária

É um método de classificação supervisionada utilizado para prever labels binárias (por exemplo 0,1). O valor de saida vai ser uma probabilidade que é mapeada para as classes através de uma função de ativação chamada sigmoide. Este modelo é treinado otimizando uma função de custo que mede o erro entre as previsões e as labels reais, utilizando em maior parte dos exemplos que vamos demonstrar a técnica de otimização do gradiente descendente.

A função de decisão da regressão logística binária é linear:

$$f(x) = w^T x + b$$

- w vetor dos pesos.
- b representa o bias.
- x elemento da base de dados.

A saída de f(x) é transformada numa probabilidade pela função sigmoide

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

A função de custo que mede o erro entre as previsões e os rótulos reais:

$$J(w,b) = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\log(\sigma(f(x^{(i)}))) + (1-y^{(i)})\log(1-\sigma(f(x^{(i)})))\right]$$

Vai penalizar previsões incorretas mais severamente quando o modelo está muito confiante e errado, o objetivo é minimizar J.Para isso, usamos gradiente descendente, que requer o cálculo dos gradientes da função de custo em relação aos parâmetros w e b respetivamente.

$$\frac{\partial J}{\partial w} = \frac{1}{n}\sum_{i=1}^{n}\left(\sigma(f(x^{(i)})) - y^{(i)}\right)x^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{1}{n}\sum_{i=1}^{n}\left(\sigma(f(x^{(i)})) - y^{(i)}\right)$$

Que depois vão ser atualizados.

$$w := w - \eta \cdot \frac{\partial J}{\partial w}$$

$$b := b - \eta \cdot \frac{\partial J}{\partial b}$$

O simbolo η representa a taxa de aprendizagem, parâmetro que vai controlar o tamanho/rapidez com que o modelo atualiza os restantes parâmetros.

Após o treino, o modelo pode prever a classe do elemento x.

$$\hat{y} = \begin{cases} 1 & \text{se } \sigma(f(x)) \geq 0.5 \\ 0 & \text{caso contrário} \end{cases}$$

1.4.2 Versão Dual com Kernel

Na versão dual com kernel da regressão logística binária, a ideia central é não trabalhar diretamente no espaço original das características, mas sim num espaço de dimensão superior onde os dados possam ser mais separáveis. Para isso, utilizamos a função kernel, que calcula a similaridade entre pares de amostras sem precisar transformar explicitamente os dados.

Diferente da versão primal onde treinamos diretamente os pesos w, na versão dual usamos um vetor α, com um coeficiente por amostra de treino.

Em vez de usar diretamente os vetores x, usamos uma função kernel, que calcula uma medida de similaridade entre pares de amostras, neste projeto a função mais utilizada foi a do Kernel Polinomial, porém foram testadas outras variantes.

$$K(x_i, x_j) = (\gamma \cdot x_i^T x_j + c)^d$$

- γ controla a influência da magnitude dos produtos internos.
- c coeficiente livre.
- d grau do polinómio.

Função sigmoide:

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

A função de decisão vai passar a ser representada por:

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$$

Função de custo:

$$J(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \log\left(1 + \exp(-y_i \cdot f(x_i))\right) + \frac{\lambda}{2}\|\boldsymbol{\alpha}\|^2$$

- λ parâmetro de regularização para evitar overfitting

Gradiente e atualização de α

$$\nabla_\alpha J = -K^T(y \cdot p) + \lambda\boldsymbol{\alpha}$$

$$\boldsymbol{\alpha} := \boldsymbol{\alpha} - \eta \cdot \nabla_\alpha J$$

- η taxa de aprendizagem.

Após o processo de treino o resultado obtido durante a previsão de um elemento é na mesma um valor probabilístico, que vai ser usado da mesma forma para fazer uma escolha.

$$\hat{y} = \begin{cases} 1 & \text{se } \sigma(f(x)) \geq 0.5 \\ 0 & \text{caso contrário} \end{cases}$$

1.5 Implementações Complementares / Optimizações

Após a implementação dos métodos apresentados anteriormente, foram exploradas quatro estratégias complementares de maneira a otimizar ambas as versões dos classificadores. Essas alternativas incluem:

- Utilização do Mini-Batch Gradient Descent, que permite configurar o tamanho do batch para equilibrar eficiência e estabilidade durante o treino;
- Utilização do Stochastic Gradient Descent, com um batch size igual a 1, na qual a base de dados é percorrida de forma sequencial;
- Exploração de diferentes métodos de otimização avançados, como Adam e Adagrad, que ajustam adaptativamente a taxa de aprendizagem;
- A exploração de diferentes kernels como o Kernel Linear ou RBF, para melhorar a capacidade dos modelos em capturar padrões nos dados (lineares e não lineares).

1.5.1 Mini-Batch Gradient Descent

O Mini-Batch (MBGD) trata-se de uma variação que combina os benefícios de outros gradientes. como o Gradiente Descendente Completo (que utiliza a totalidade dos dados de treino) e o Gradiente Descendente Estocástico (que processa apenas uma amostra de cada vez). Este divide os dados em pequenos subconjuntos (batches) de tamanho configurável, processando cada batch separadamente para atualizar os parâmetros do modelo.
As suas etapas principais consistem em:

1. *Shuffle* da base de dados para garantir que cada batch é representativo da distribuição dos dados.
2. Divisão dos dados em mini-batches de tamanho especificado.
3. Cálculo do gradiente para cada batch, utilizando a função de custo e sua derivada.
4. Atualização dos parâmetros do modelo (vetor de pesos, w e bias, b) com base no gradiente calculado.
5. Repetição do passo anterior até alcançar convergência ou a satisfação de um critério de paragem

Entre as principais vantagens deste gradiente, destacam-se a sua convergência mais rápida do que outras técnicas, especialmente em grandes conjuntos de dados devido à maior frequência de atualização dos parâmetros. Para além disso, os pequenos batches utilizados reduzem a memória necessária e o seu treino é mais estável do que noutros gradientes como o Gradiente Descendente Estocástico, proporcionando um caminho de convergência mais suave.

1.5.2 Stochastic Gradient Descent

Como referido brevemente ao falar do MBGD, o Stochastic Gradient descent (equivalente a batch size=1) atualiza os parâmetros após cada amostra de treino, percorrendo a base de dados de forma sequencial, tornando-se particularmente útil em cenários onde a eficiência computacional e a capacidade de escapar de mínimos locais são prioridades.
As suas etapas principais consistem em:

1. Em cada iteração, é selecionada apenas uma única amostra do conjunto de dados ($(x^{(i)}, y^{(i)})$).

2. O gradiente da função de custo (J) é calculado com base na amostra atual.
3. Os parâmetros do modelo (como o vetor de pesos, w e o bias, b) são atualizados na direção oposta ao gradiente.
4. Repetição do processo para todas as amostras de um conjunto de dados até que um critério de paragem seja satisfeito.

Por processar apenas uma amostra de cada vez, este gradiente é altamente eficiente em termos computacionais, requerendo menos memória, o que o torna ideal para conjuntos de dados grandes que não cabem na memória. Além disso, as atualizações frequentes dos parâmetros, realizadas após cada amostra, permitem uma adaptação rápida do modelo, sendo particularmente útil em cenários de aprendizagem online, onde os dados chegam em tempo real. A alta variância nas atualizações também é uma vantagem, pois ajuda o modelo a escapar de mínimos locais em funções de custo não convexas, favorecendo a busca por soluções mais robustas. Por fim, em conjuntos de dados mais pequenos, o SGD destaca-se pela rapidez, visto que não precisa de processar grandes quantidades de dados em cada iteração, tornando o seu treino mais ágil.

1.5.3 Exploração de Diferentes Kernels

Na versão dual da regressão logística, foram avaliados dois kernels principais: o kernel linear e o kernel de base radial (RBF), com o objetivo de capturar diferentes tipos de separabilidade nos dados. Estes kernels transformam o espaço de entrada para facilitar a classificação, sendo particularmente úteis nas abordagens One-vs-All (OvA) e Error-Correcting Output Codes (ECOC).

Kernel Linear

O kernel linear é definido como:

$$K(x_i, x_j) = x_i^T x_j,$$

onde x_i e x_j são vetores de características. Este kernel calcula o produto interno entre os vetores, mantendo os dados no espaço original sem transformações não lineares.

Este kernel destaca-se pela sua eficiência computacional, especialmente em conjuntos de dados onde as classes são linearmente separáveis, permitindo modelar relações entre características e classes através de uma fronteira linear simples. A sua simplicidade reduz significativamente o custo computacional, tornando-o adequado para datasets de grande escala com separabilidade clara. Além disso, minimiza o risco de *overfitting* em cenários onde a separação linear é suficiente.

Kernel RBF

O kernel de base radial (RBF) é definido como:

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right),$$

onde γ é um hiperparâmetro que controla a largura do kernel, influenciando a sensibilidade às distâncias entre os pontos.
A função de decisão na forma dual é dada por:

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x),$$

e a função de custo é:

$$J(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i \cdot f(x_i)\right)\right) + \frac{\lambda}{2}\|\boldsymbol{\alpha}\|^2,$$

com o gradiente:

$$\nabla_\alpha J = -K^T(y \cdot p) + \lambda \boldsymbol{\alpha},$$

onde K é a matriz de kernel, p é o vetor de probabilidades, e λ é o parâmetro de regularização.

Este kernel é altamente eficaz para capturar padrões não lineares, permitindo modelar relações complexas entre características e classes em datasets onde as fronteiras de decisão não são lineares. A sua flexibilidade torna-o adequado para uma ampla gama de problemas de classificação, especialmente em cenários onde os dados apresentam uma separabilidade não linear.
Esta capacidade de adaptação a padrões complexos melhora significativamente o desempenho em tarefas desafiadoras.

Capítulo 2

Dataset

2.1 Descrição do *Dataset*

Os *datasets* usados dividem-se em dois tipos, dados sintéticos, criados especificamente para o projeto, e um *dataset* real, para uma aplicação mais realista das técnicas desenvolvidas. Os dados sintéticos permitem um maior controlo e estão limitados a 2 atributos, para se visualizar graficamente os resultados. Foram criados vários *datasets* sintéticos, todos com três classes, de forma a testar a *performance* dos modelos com diferentes distribuições de dados. A base de dados real escolhida é mais complexa, com um número maior de atributos e classes, que permite uma comparação de tempo computacional e qualidade dos métodos usados num cenário menos controlado.

2.2 Leitura e Análise dos Dados

Os três *datasets* sintéticos têm distribuições de dados diferentes, cada ponto é caracterizado por duas coordenadas e um valor inteiro, que indica a sua classe com base nas coordenadas. Os dados reais são baseados num conjunto de medições físicas de 4177 indivíduos da espécie abalone, um animal cuja idade é indicada pelo número de anéis presentes na sua concha. As medições incluem 9 atributos, entre estes diferentes pesos, tamanhos, o sexo e o número de anéis, um valor inteiro entre 1 e 29. O problema de classificação será então a análise das medições e previsão da idade com base nessa informação. Como o cálculo de idade obtém-se adicionando 1.5 ao número de anéis, não é relevante para o problema, sendo então os resultados obtidos relativos apenas ao valor do campo *Rings*.

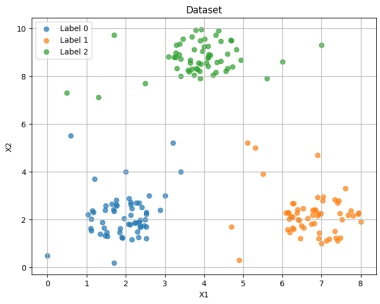

Figura 2.1:

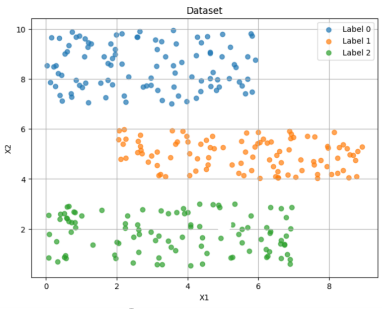

Figura 2.2:

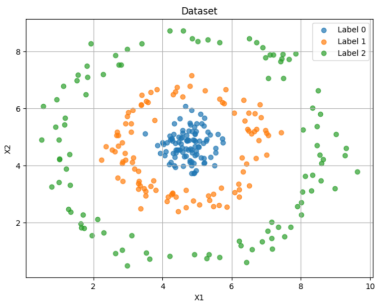

Figura 2.3:

2.3 Tratamento de Dados

Sendo um conjunto de dados para teste, os *datasets* sintéticos não necessitam de tratamento, todos os valores são numéricos e os dados estão no formato necessário para a aplicação dos classificadores. Quanto aos dados sobre abalones, apesar de ser um *dataset* pré-tratado, foi necessário remover alguns valores em falta e alterar um atributo para valores numéricos. Para a classificação, agruparam-se os valores de anéis em 4 *bins*, de forma a reduzir o número de classes. Foi importante manter as amostras de cada *bin* próximas, de modo a não manipular os resultados.

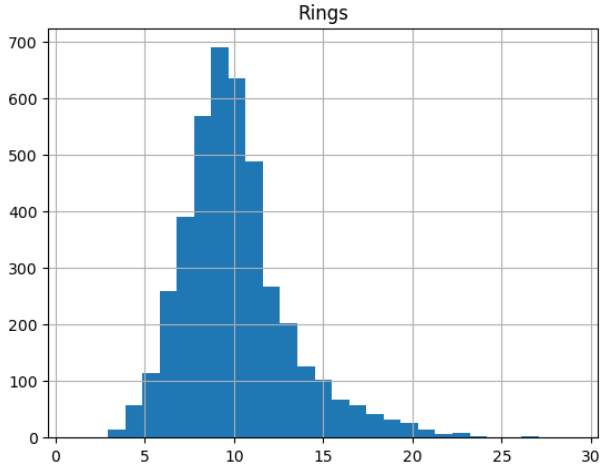

Figura 2.4: Distribuição de elementos por anéis

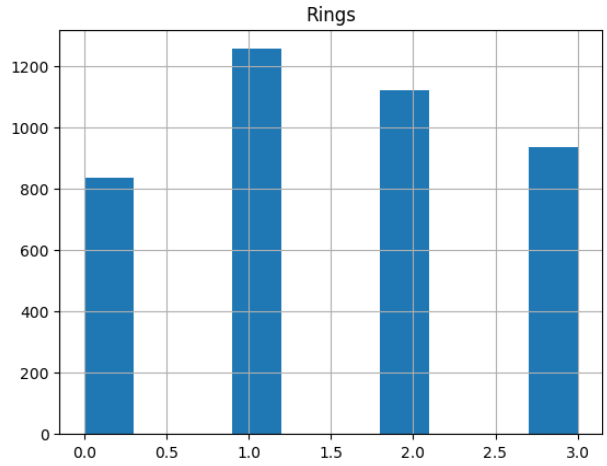

Figura 2.5: Distribuição de elementos por *bin* de anéis

Para se obter dados para teste, o *dataset* foi dividido em 75% dos dados para treino e aos restantes 25% separou-se o atributo dos anéis para teste. Esta divisão de treino e teste foi escolhida por obter os melhores resultados em termos de precisão.

Capítulo 3

Resultados Obtidos e Observações

3.1 Dados Sintéticos

Nesta secção vamos explorar alguns dos resultados obtidos, em ambas as versões Primal e Dual com Kernel Polinomial. No caso dos datasets mais pequenos vamos dar prioridade ao aspeto visual do resultado, devido a ter apenas dois atributos conseguimos observar a divisão que está a ser feita entre as classes.

Todos os resultados comparados entre as imagens seguiram o mesmo critério de paragem, a accuracy do modelo não sofrer uma alteração (com uma tolerância de 0.001) durante x iterações seguidas, porém alguns dos casos que vão ser indicados deixamos correr até obter uma "divisão perfeita"(uma accuracy de 100% dos elementos da base de dados). Os valores da learning rate η utilizados foram 0.1 e 0.01.

3.1.1 Primal

Valores mais utilizados para leraning rate η 0.1 0.01

No primeiro conjunto de imagens temos lado a lado o resultado do metodo ECOC e Ova no formato Primal. Conseguimos ver de imediato que a imagem da esquerda apresenta uma divisão para as fronteiras mais

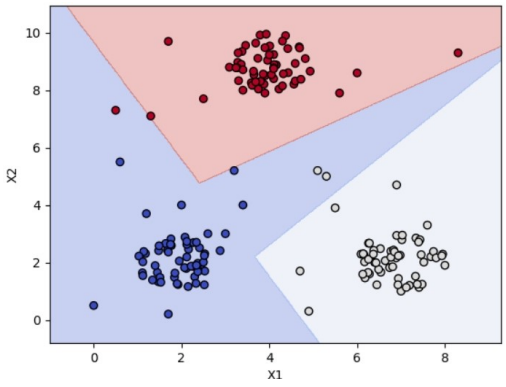

Figura 3.1: Divisão do ECOC Primal GD

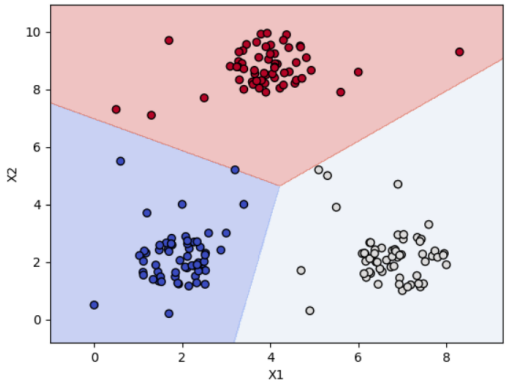

Figura 3.2: Divisão do OvA Primal GD

A figura número 2 porém foi obtida com a condição de paragem da accuracy chegar a 100%. Algo que não foi possível na figura 1. Para chegar ao resultado pedido, foram precisas 30 mil iterações, com learning rate de 0.1.

Notámos que era possível diminuir o número de iterações ao aumentar a learning rate, porém os valores da função loss começavam a aumentar nas primeiras iterações, no entanto com o passar das iterações o valor estabilizava para

valores menores a 0.1, atingindo valores em que ainda podemos afirmar que o modelo está a fazer previsões corretas e com confiança.

Evolução da Loss com learning rate mais alta. Valores da loss nas primeiras 10 iterações e a evolução durante as iterações totais.

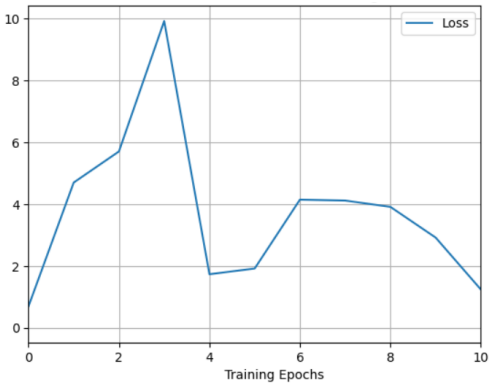

Figura 3.3: Primeiras 10 iterações

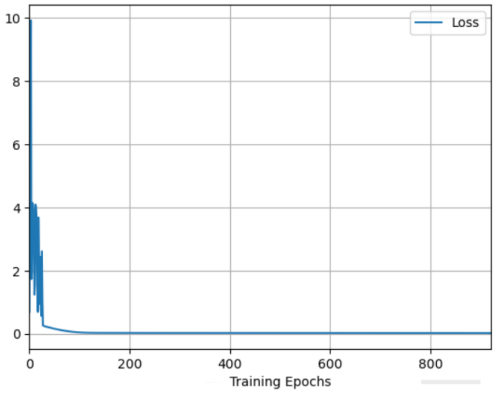

Figura 3.4: Evolução nas 900 iterações

As quatro figuras seguintes comparam os resultados entre diferentes batch sizes para o método ECOC. Podemos observar que conforme o tamanho do batch vai diminuindo os resultados tendem a melhorar, de notar que o número de iterações totais também vai diminuindo conforme o tamanho do batch, o critério de paragem foi igual para os 4 casos.

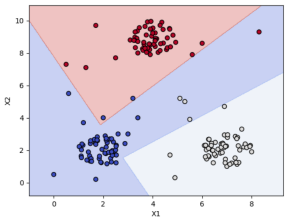

Figura 3.5: Size 128

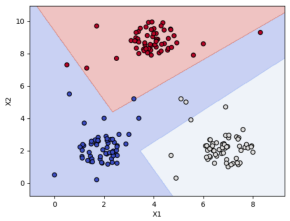

Figura 3.6: Size 32

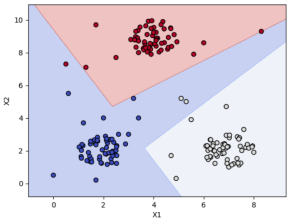

Figura 3.7: Size 16

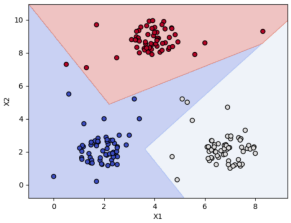

Figura 3.8: Size 8

No caso do método **OvsA** vamos comparar os dois casos de paragem possíveis, na primeira imagem com menos de 100 iterações conseguimos um modelo que apresenta uma accuracy de aproximadamente 98% , neste caso o modelo parou após não sofrer alterações significativas nesse valor (tolerância de 0.01) durante 30 iterações. No entanto ao trocar de caso de paragem (accuracy chegar a 100%) conseguimos a divisão da direita,que apresenta todos os elementos da base de dados bem classificados e um conjunto de fronteiras mais uniformes entre as classes, porém foram precisas 3700 iterações.

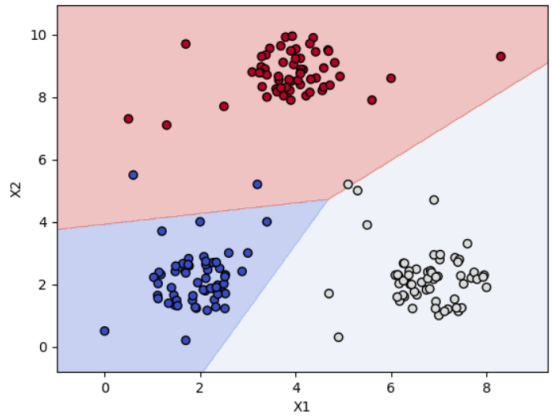

Figura 3.9: Divisão OvA Accuracy 0.98

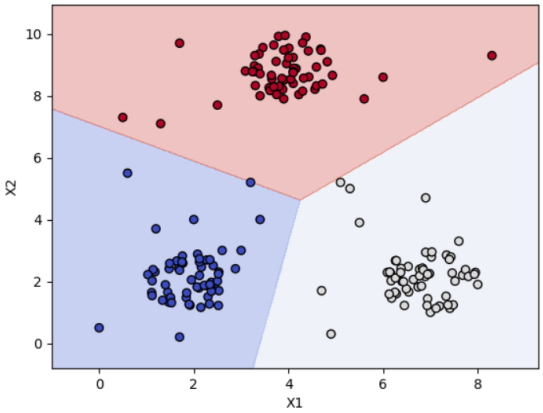

Figura 3.10: Divisão OvA Accuracy 1

Em relação aos resultados do modelo estocástico, que percorre sequencialmente a base de dados, foram muito semelhantes aos obtidos com batch sizes mais pequenos nos modelos anteriores, a única coisa que se notou de diferente foi no tempo de computação. Neste casos os modelos demoravam um pouco mais a chegar ao mesmo resutlado.

3.1.2 Dual com Kernel Polinomial

Agora vamos abordar alguns casos na versão Dual com Kernel Polinomial. Os valores dos parâmetros utilizados para os resultados obtidos foram quase sempre os mesmos. Os primeiros testes foram feitos com valores default, porém os valores finais com que trabalhamos em maior parte dos casos e obtivemos melhores resultados foram:

1. $\gamma = 0.1$ ou 0.05
2. c = 1
3. η 0.1 ou 0.01
4. λ 0.1 ou 0.01

Tendo em conta que os resutlados obtidos na versão primal para esta base de dados já foram bastante satisfatórios e que se mantiveram na versão dual, vamos abordar duas situações diferentes (utilizando Kernel) e apresentar resutlados com outras duas funções para o Kernel.

No caso do **OvsA** com Batch, apresentamos que diferentes condições de paragem podem influenciar o resultado obtido, obtendo com mais algumas iterações uma accuracy perfeita em conjunto com uma fronteira entre as classes bastante satisfatória.

Porém nem sempre isso é verdade, neste exemplo vamos apresentar o oposto, nem sempre uma accuracy a 100% pode corresponder ao melhor resultado a nivel das fronteiras. Nas duas imagens seguintes conseguimos ver que a divisão apresentada na direita apesar de apresentar um valor mais baixo para a accuracy apresenta uma fronteira com distâncias mais uniformes entre os pontos mais afastados das diferentes classes.

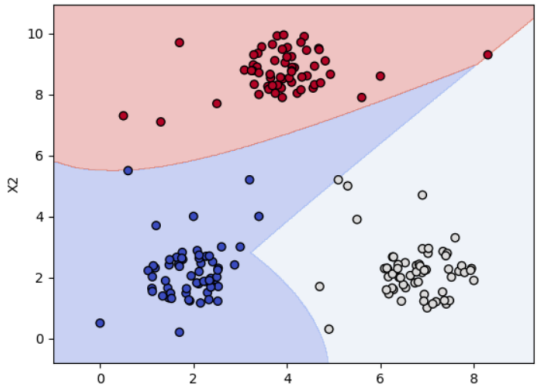

Figura 3.11: Divisão ECOC Accuracy 1

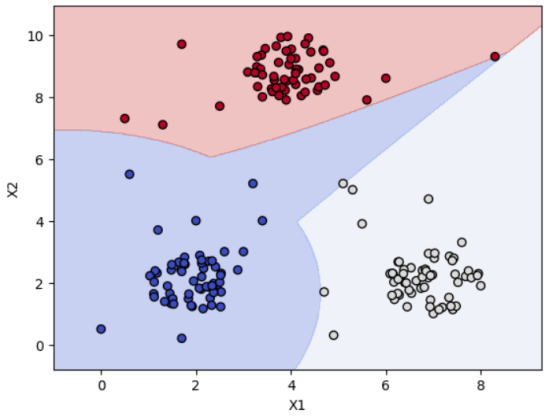

Figura 3.12: Divisão ECOC Accuracy 0.99

Um dos fatores mais importantes para obter um bom classificador é o processo de tuning, no caso do kernel polinomial são introduzidos dois novos valores, γ e c. O tuning destes valores pode ter um impacto enorme no resultado final que obtemos.

No exemplo seguinte vamos ilustrar dois resutlados com valores de γ diferentes,as restantes variáveis mantiveram-se iguais, até o número de iterações, que serviu como caso de paragem para comparar os exemplos.

O gamma vai controlar o impacto da magnitude do produto interno na função do kernel, atua como um fator de escala, aumentando ou reduzindo a influência das similaridades entre os vetores. Com valores mais baixos a função de decisão tende a ficar mais suave e menos sensível às variações entre os vetores
Nas imagens seguintes conseguimos ver o impacto de um γ mais baixo no resultado final.

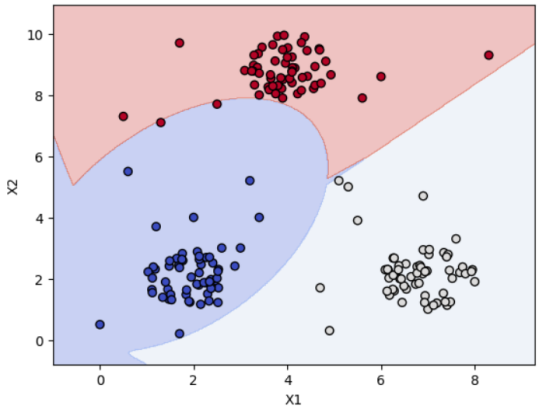

Figura 3.13: $\gamma = 0.1$

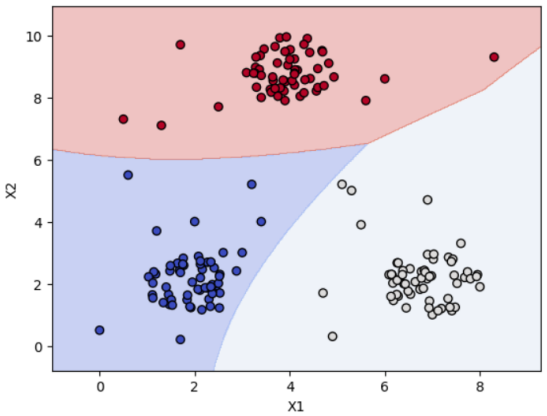

Figura 3.14: $\gamma = 0.05$

Para além do Kernel Polinomial, foram testados outras duas funções para o Kernel. Kernel Linear, que apresentou no geral resutlados piores que o polinomial, em todas as implementações, apresentando resultados semelhantes ao da imagem 15.

O outro Kernel testado foi o RBF Kernel (Radial Basis Function), nesta implementação os resultados foram praticamente perfeitos, obtendo resultados semelhantes ao da imagem 16. Não é muito comum, pode ser devido ao data

set ser bastante simples, mas também pode estar algo incorreto na implementação ou no tuning dos parâmetros.

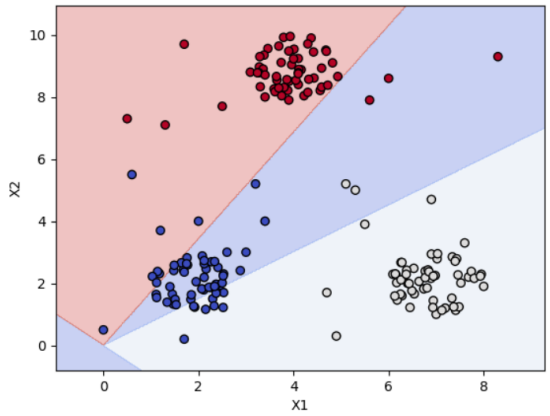

Figura 3.15: Kernel Linear

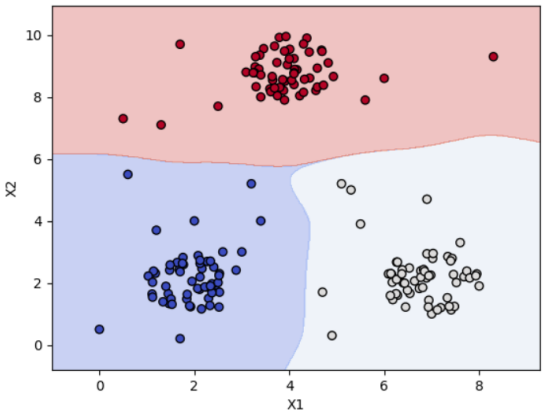

Figura 3.16: RDF Kernel

Uma vez que ambas as implementações Primal e Dual apresentaram resultados semelhantes na base de dados utilizada inicialmente, apresentando apenas uma pequena melhoria entre o metodo ECOC e OvA no geral, foram criadas mais duas base de dados que nos permitiram observar resultados com diferenças mais significativas entre os dois métodos.

Nas primeiras 4 imagens conseguimos observar a dificuldade que o método Primal apresenta ao tentar estabelecer fronteiras em ambos os casos, principalmente na Base de Dados número 2. Isto acontece ao facto da base de dados não ser linearmente separável, tornando o trabalho do método primal impossível.

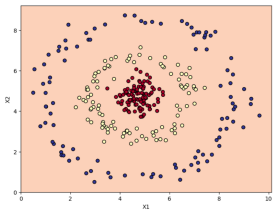

Figura 3.17: Base de Dados 2 ECOC

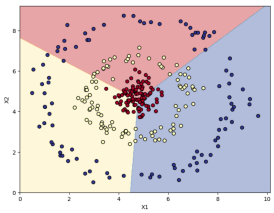

Figura 3.18: Base de Dados 2 OvA

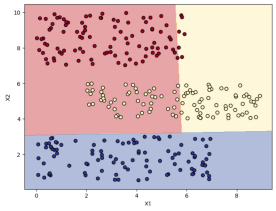

Figura 3.19: Base de Dados 3 ECOC

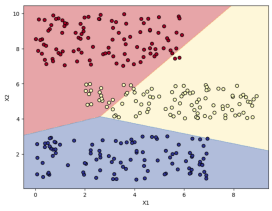

Figura 3.20: Base de Dados 3 OvA

Já na versão Dual já conseguimos observar algum progresso em ambas as base de dados os modelos já conseguem estabelecer fronteiras entre as classes, principalmente na base de dados 3 em que obtemos um resultado com uma

accuracy a 100% relativamente aos elementos da base de dados.

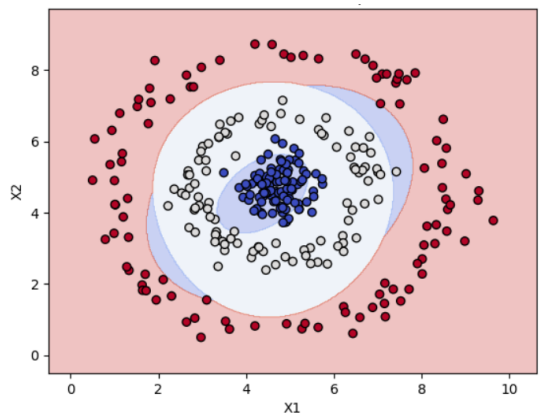

Figura 3.21: Base de Dados 2

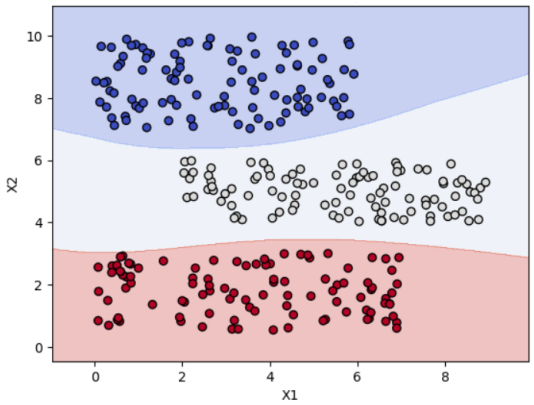

Figura 3.22: Base de Dados 3

3.2 *Dataset* Abalone

De forma a comparar os diferentes métodos usados, calculou-se a *accuracy* das previsões obtidas. Sendo um *dataset* maior e mais complexo, notou-se um aumento significativo de tempo computacional, especialmente para os algoritmos mais complexos. Em geral, a técnica *One-vs-All* demonstrou melhores resultados quando comparada com o método *Error-Correcting Output Codes*. Em termos de custo computacional, a melhor performance do OvA traz um tempo de execução maior, o uso de *mini-batch* melhora a *performance* com uma pequena diminuição de *accuracy* na versão *Kernel*, enquanto que as versões *kernel* aumento o tempo de execução significativamente, com uma pequena melhoria de precisão no método ECOC.

Método	**Accuracy**	**Tempo Computacional**
ECOC	0.37	Baixo
OvA	0.55	Médio
ECOC MB	0.38	Baixo
OvA MB	0.55	Baixo
ECOC Kernel	0.41	Alto
OvA Kernel	0.42	Alto
ECOC Kernel MB	0.40	Médio
OvA Kernel MB	0.44	Alto

Tabela 3.1: Valores de accuracy e tempo de execução relativo para os diferentes métodos

Capítulo 4

Conclusão

Este estudo permitiu comparar de forma abrangente as estratégias *One-vs-All* (**OvA**) e *Error-Correcting Output Codes* (**ECOC**), tanto na sua versão primal como na versão com *kernel*, utilizando a regressão logística como classificador

base. Através da análise em diferentes conjuntos de dados — desde bases sintéticas até um *dataset* real mais complexo — foi possível avaliar não só a eficácia de cada método, mas também o impacto de técnicas de otimização como o *mini-batch* e a *kernelização*.

Nos dados sintéticos, tanto versão primal como dual com kernel apresentaram resultados bastante bons, tendo se notado uma pequena diferença entre as técnicas **ECOC** e **OvsA**, destacando-se um pouco a versão **OvsA**.

A introdução do *kernel* polinomial trouxe um salto significativo na qualidade da classificação para problemas não linearmente separáveis. As duas base de dados criadas posteriormente realçam a eficácia deste método em comparação com a versão Primal. Aproveitando a flexibilidade do mapeamento para um espaço de características de maior dimensão.

Quando aplicados ao *dataset* real (**Abalone**), os resultados demonstram melhores resultados para o método OvA em cenários multiclasse mais complexos, ao custo de um maior tempo de execução face ao ECOC.

A exploração de técnicas de otimização, como o *mini-batch*, revelou um melhoramento na eficiência. Reduzir o tamanho do *batch* para valores intermédios (32 a 128 amostras) permitiu acelerar o treino em, sem perdas significativas de qualidade.

Em suma, este trabalho demonstrou que a escolha entre **OvA** e **ECOC** depende fortemente do contexto do problema. Enquanto que o **OvA** é ideal para cenários com poucas classes e dados linearmente separáveis, o **ECOC** destaca-se em problemas multiclasse complexos. Como trabalho futuro, seria interessante continuar a explorar *kernels* diferentes, tentar compreender melhor os resultados obtidos com o *RDF kernel* e dedicar mais tempo a explorar diferentes ténicas de aprendizagem como *Adam* ou *Adagrad*.