

Trabalho de Aprendizagem Computacional I

Integrantes

- Miguel Almeida 202303926
- Bruno Costa 202108459
- Pedro Julião 202204034

PL4-G1

Sumário

Objetivo

- Implementar regressão logística multiclass do zero
- Analisar o impacto de características dos dados na performance
- Tornar o modelo mais robusto à complexidade multiclass

Abordagem

- Implementação de duas variantes: Softmax e One-vs-Rest
- Benchmark em datasets multiclass (Iris, Vehicle, Wine)
- Expansão polinomial como proposta de melhoria (grau 1, 2, 3)

Resultados

- Acurácia aumentou até **+22%** com features polinomiais
- Melhor performance no dataset **Wine** (95–100%)
- A escolha entre OvR e Softmax deve considerar o **contexto do problema**

Regressão Logística – Modelo de Classificação Binária

- A **Regressão Logística** é um modelo estatístico utilizado para **classificação binária**, ou seja, quando existem **duas classes possíveis** (ex: positivo vs negativo).
- A ideia principal é modelar a **probabilidade** de uma amostra pertencer à classe positiva $y=1$ com base nas suas características x , usando uma função sigmoide.
- O modelo aprende os coeficientes que melhor separam as duas classes, minimizando uma **função de custo logarítmica** (log-loss ou cross-entropy).

```
class LogisticRegression(BasicRegression):  
    def init_cost(self):  
        self.cost_func = binary_crossentropy  
  
    def _loss(self, w):  
        loss = self.cost_func([self.y, self.sigmoid(np.dot(self.X, w))])  
        return self._add_penalty(loss, w)  
  
    @staticmethod  
    def sigmoid(x):  
        return 0.5 * (np.tanh(0.5 * x) + 1)  
  
    def _predict(self, X=None):  
        X = self._add_intercept(X)  
        return self.sigmoid(X.dot(self.theta))
```

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

Classificação Multiclass

- A **Classificação Multiclass** é um problema onde existem **mais de duas classes possíveis** (ex: 3 ou mais categorias).
- A Regressão Logística tradicional **não lida diretamente com mais de duas classes**, o que exige **estratégias de extensão**, como:
 - **One-vs-Rest (OvR)**: divide o problema em vários binários.
 - **Softmax Regression**: modelo unificado para múltiplas classes.
- **Desafios:**
 - Decisões mais complexas: cada amostra pode pertencer a uma entre muitas classes.
 - Comparação justa entre todas as classes requer métricas apropriadas (ex: accuracy, F1 por classe, confusion matrix).

Estendendo a regressão logística para classificação multiclass

Motivação:

- A Regressão Logística é um modelo estatístico amplamente utilizado em problemas de classificação, mas a sua formulação padrão é limitada à classificação binária.
- Muitos problemas reais envolvem mais de duas classes.
- Neste projeto, enfrentámos exatamente esse cenário: como aplicar Regressão Logística a um problema com três ou mais classes.
- A motivação central foi compreender quais as abordagens possíveis para adaptar o modelo binário ao contexto multiclass e avaliar o impacto de cada uma.

• Objetivo da proposta:

- Investigar e implementar duas estratégias clássicas para tratar classificação multiclass com Regressão Logística:
 - One-vs-Rest (OvR), baseada em múltiplos classificadores binários independentes.
 - Softmax Regression, uma generalização da Regressão Logística binária.
- Avaliar comparativamente estas abordagens em termos de:
 - Desempenho preditivo global e por classe.
 - Comportamento em relação à complexidade computacional.
 - Robustez e sensibilidade aos dados.

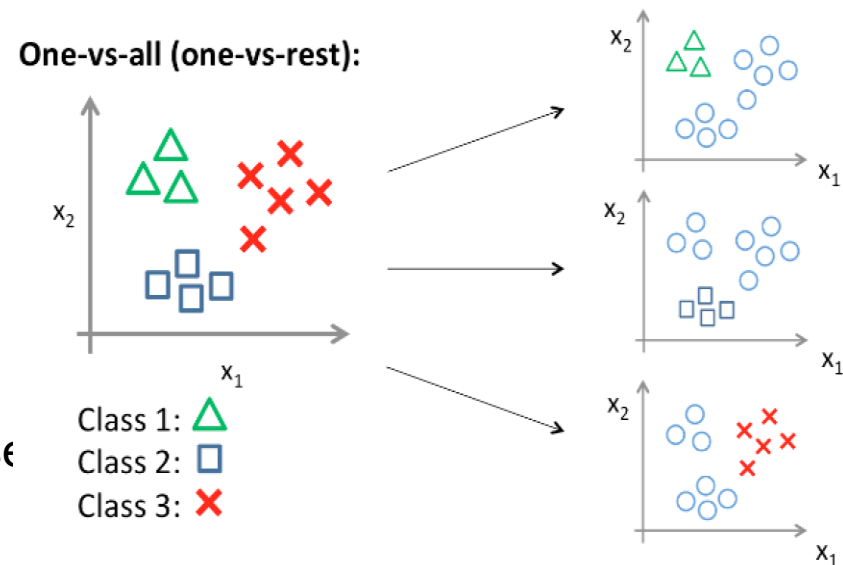
One-vs-Rest Logistic Regression (OvR)

Descrição do Algoritmo

- OvR é uma estratégia de decomposição para problemas de classificação multiclass.
- Transforma um problema multiclass em múltiplos problemas binários.
- Para cada classe, treina-se um classificador binário que distingue de todas as outras.
- A previsão é feita com base na maior probabilidade atribuída entre os classificadores binários.

Porque escolhemos OvR:

- Simplicidade de implementação e interpretação.
- Compatível com regressão logística (algoritmo base).
- Permite avaliar diretamente o impacto de características como o desbalanceamento de classe em cenários multiclass.



Código de One-vs-Rest

BinaryLogisticRegression

- Implementa a Regressão Logística Binária com:
 - - Função de ativação sigmoid;
 - - Função de custo entropia cruzada binária;
 - - Suporte a regularização L1 e L2;
 - - Treinamento via gradiente descendente.

OneVsRestLogisticRegression

- Usa internamente um BinaryLogisticRegression para cada classe.
 - - Treina um modelo para cada classe (vs. todas as outras);
 - - No momento da previsão, calcula a probabilidade de cada classe separadamente;
 - - A classe com maior probabilidade é a escolhida.

```
class OneVsRestLogisticRegression(BaseEstimator):
    def __init__(self, lr=0.001, penalty="None", C=0.01, tolerance=0.0001, max_iters=1000):
        self.lr = lr
        self.penalty = penalty
        self.C = C
        self.tolerance = tolerance
        self.max_iters = max_iters
        self.classifiers = {}
        self.label_encoder = LabelEncoder()
        self.classes_ = None

    def fit(self, X, y=None):
        self._setup_input(X, y)

        self.y_encoded = self.label_encoder.fit_transform(self.y)
        self.classes_ = self.label_encoder.classes_

        for class_label in range(len(self.classes_)):
            binary_target = (self.y_encoded == class_label).astype(int)

            classifier = BinaryLogisticRegression(
                lr=self.lr, penalty=self.penalty, C=self.C,
                tolerance=self.tolerance, max_iters=self.max_iters
            )
            classifier.fit(X, binary_target)
            self.classifiers[class_label] = classifier

    def predict(self, X):
        if not isinstance(X, np.ndarray):
            X = np.array(X)
```

```
    def predict(self, X):
        if not isinstance(X, np.ndarray):
            X = np.array(X)
        return self._predict(X)

    def predict_proba(self, X):
        """Predict class probabilities using one-vs-rest approach"""
        n_samples = X.shape[0]
        n_classes = len(self.classes_)
        probabilities = np.zeros((n_samples, n_classes))

        for class_label, classifier in self.classifiers.items():
            probabilities[:, class_label] = classifier.predict_proba(X)

        probabilities = probabilities / np.sum(probabilities, axis=1, keepdims=True)
        return probabilities

    def _predict(self, X):
        """Predict class labels using one-vs-rest approach"""
        probabilities = self.predict_proba(X)
        predicted_encoded = np.argmax(probabilities, axis=1)
        return self.label_encoder.inverse_transform(predicted_encoded)
```

Comportamento OvR vs Característica de Dados

Vantagens com base nas características dos dados:

- **Simple e modular:** adapta-se bem a datasets com **número moderado de classes**
- **Eficaz com classes bem separadas:** quando as **fronteiras entre classes são claras**, OvR apresenta ótimo desempenho.
- **Robusto a ruído em algumas classes**
- **Permite diagnóstico individual**

Impacto na prática (dataset Digits):

- Com 10 classes bem definidas (0–9), OvR conseguiu **alta performance (~92% accuracy)**.
- Beneficiou de classes **visualmente distintas** (ex: “1” vs “8”) → fronteiras fáceis de separar binariamente.
- No entanto, apresentou **confusão entre classes similares** (ex: “4” e “9”) devido à separação isolada dos modelos.

Softmax — Regressão Logística Multiclass

Descrição do Algoritmo:

- Generalização da regressão logística binária para problemas multiclass.
- Em vez de treinar K classificadores binários, ajusta um único modelo que estima diretamente as probabilidades de cada classe via função Softmax.
- Todas as classes são tratadas simultaneamente, permitindo modelar dependências entre elas.

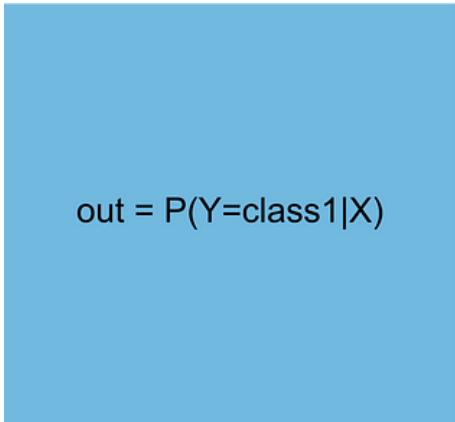
Porque escolhemos Softmax:

- Modelo probabilístico elegante e unificado.
- Mais eficiente e teoricamente sólido para problemas puramente multiclass.
- Permite uma interpretação direta das probabilidades preditas.

$$\frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

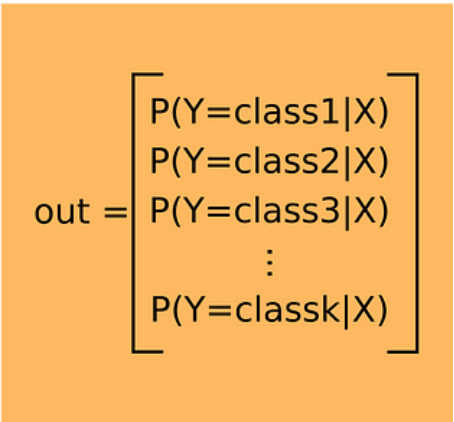
Sigmoid

2 classes


$$\text{out} = P(Y=\text{class1}|X)$$

SoftMax

k>2 classes


$$\text{out} = \begin{bmatrix} P(Y=\text{class1}|X) \\ P(Y=\text{class2}|X) \\ P(Y=\text{class3}|X) \\ \vdots \\ P(Y=\text{classk}|X) \end{bmatrix}$$

Código Softmax

```
def softmax(x):
    """Numerically stable softmax implementation"""
    x_shifted = x - anp.max(x, axis=1, keepdims=True)
    exp_x = anp.exp(x_shifted)
    return exp_x / anp.sum(exp_x, axis=1, keepdims=True)

def categorical_crossentropy(actual, predicted):
    """Categorical cross-entropy loss for multiclass classification"""
    predicted = anp.clip(predicted, EPS, 1 - EPS)
    return -anp.mean(anp.sum(actual * anp.log(predicted), axis=1))

def binary_crossentropy(actual, predicted):
    predicted = anp.clip(predicted, EPS, 1 - EPS)
    return anp.mean(-anp.sum(actual * anp.log(predicted) + (1 - actual) * anp.log(1 - predicted)))

# APPROACH 1: Multinomial Logistic Regression (Softmax)
class MultinomialLogisticRegression(BasicRegression):
    def __init__(self, lr=0.001, penalty="None", C=0.01, tolerance=0.0001, max_iters=1000):
        super().__init__(lr, penalty, C, tolerance, max_iters)
        self.label_encoder = LabelEncoder()
        self.n_classes = None

    def _one_hot_encode(self, y):
        """Convert labels to one-hot encoding"""
        n_samples = len(y)
        one_hot = anp.zeros((n_samples, self.n_classes))
        one_hot[anp.arange(n_samples), y] = 1
        return one_hot

    def fit(self, X, y=None):
        self._setup_input(X, y)
        self.y_encoded = self.label_encoder.fit_transform(self.y)
        self.n_classes = len(self.label_encoder.classes_)
        self.y_one_hot = self._one_hot_encode(self.y_encoded)

        self.n_samples, self.n_features = X.shape
        self.theta = anp.random.normal(size=(self.n_features + 1, self.n_classes), scale=0.01)

        self.X = self._add_intercept(self.X)

        self._train()

    def _loss(self, theta):
        """Compute the loss for multinomial logistic regression"""
        theta_matrix = theta.reshape(self.n_features + 1, self.n_classes)
        logits = anp.dot(self.X, theta_matrix)
        probabilities = softmax(logits)

        loss = categorical_crossentropy(self.y_one_hot, probabilities)

        if self.penalty == "l1":
            loss += self.C * anp.sum(anp.abs(theta_matrix[1:]))
        elif self.penalty == "l2":
            loss += 0.5 * self.C * anp.sum(theta_matrix[1:] ** 2)

        return loss

    def _train(self):
        """Train using gradient descent"""
        theta_flat = self.theta.flatten()
        errors = []

        grad_loss = grad(self._loss)

        for i in range(self.max_iters):
            current_loss = self._loss(theta_flat)
            errors.append(current_loss)

            gradients = grad_loss(theta_flat)

            theta_flat -= self.lr * gradients

            if i > 0 and abs(errors[i-1] - errors[i]) < self.tolerance:
                logging.info(f"Convergence reached at iteration {i}")
                break

        self.theta = theta_flat.reshape(self.n_features + 1, self.n_classes)
        self.errors = errors

    def predict_proba(self, X):
        """Predict class probabilities"""
        X = self._add_intercept(X)
        logits = np.dot(X, self.theta)
        return softmax(logits)

    def _predict(self, X):
        """Predict class labels"""
        probabilities = self.predict_proba(X)
        predicted_encoded = anp.argmax(probabilities, axis=1)
        return self.label_encoder.inverse_transform(predicted_encoded)
```

Explicação código Softmax

Funções principais:

- `softmax(x)`: transforma logits em **probabilidades normalizadas** de forma numericamente estável.
- `categorical_crossentropy(actual, predicted)`: calcula a **função de custo** para problemas multiclass com rótulos one-hot.

Classe `MultinomialLogisticRegression`:

- Implementa regressão logística multiclasse com **softmax e regularização (L1/L2)**.
- Principais hiperparâmetros:
 - `lr`, `penalty`, `C`, `tolerance`, `max_iters`

Treino(`fit`):

- Codifica rótulos (LabelEncoder + one-hot)
- Adiciona bias, inicializa pesos aleatórios
- Treina com **gradiente descendente**, usando autograd para derivar o custo

Predição:

- `predict_proba`: devolve **probabilidades por classe**
- `_predict`: seleciona a classe com maior probabilidade e reverte codificação

Comportamento do Softmax vs. Característica dos Dados

Vantagens com base nas características dos dados:

- **Consistência global:** aprende **todas as classes em conjunto**, o que melhora a coerência das decisões entre elas.
- **Escalável para muitas classes:** mais eficiente que OvR quando há **muitas classes**, pois evita treinar múltiplos modelos.
- **Melhor gestão de classes similares**
- **Probabilidades calibradas:** resultado direto do softmax — útil para **interpretação probabilística e tomada de decisão baseada em confiança**.
- **Aproveita correlações entre classes**

Impacto na prática:

- Conseguiu **melhor performance geral** (~94% accuracy), especialmente entre **classes visualmente próximas** (ex: “4” vs “9”).
- Demonstrou **mais estabilidade** nos resultados e **menor taxa de confusão entre classes** do que OvR.
- Beneficiou do fato de o dataset ter **todas as classes bem representadas** e características numéricas normalizadas.

Estudo Empírico – Logistic Regression

Preparação inicial

- Dados balanceados, numéricos
- Foco de testar a ineficácia do algoritmo para multiclass classification e sua eficácia em binary classification
- Hiperparâmetros: learning rate – 0.01, iterações – 1000, penalidade – None, tolerância – 0.0001
- Estimativa de desempenho: Hold-out (80% treino, 20% teste), random state – 42

Análise de resultados

-Usando o dataset Íris, que contem multiclass, os resultados apresentados são ruins e com baixo accuracy – 33.33%

```
Accuracy: 0.3333333333333333
Confusion Matrix:
[[ 0 15  0]
 [ 0 15  0]
 [ 0 15  0]]
```

-Usando o dataset SPECT, que contem apenas duas classes é perceptível uma melhora nos resultados tendo assim, um accuracy – 83.95%

```
Accuracy: 0.8395061728395061
Confusion Matrix:
[[ 7 10]
 [ 3 61]]
```

Conclusão

Olhando para os resultados observa-se que como dito anteriormente, a **Logistic Regression** é ineficiente para classificação de mais de duas classes, enquanto para sua aplicação original ele consegue ser 142% mais preciso

Estudo Empírico – One-vs-Rest

Preparação inicial

- Datasets com mais classes para provar eficácia: Iris (3 classes), Wine (3 classes), JapanVowels(9 classes).
- Hiperparâmetros iguais ao **Logistic Regression**
- Foco em avaliar o desempenho em decompor os problemas de multiclass em vários problemas binários independentes
- Estimar desempenho: Hold-out (70%-treino, 30% teste), random state - 42

Análise de resultados

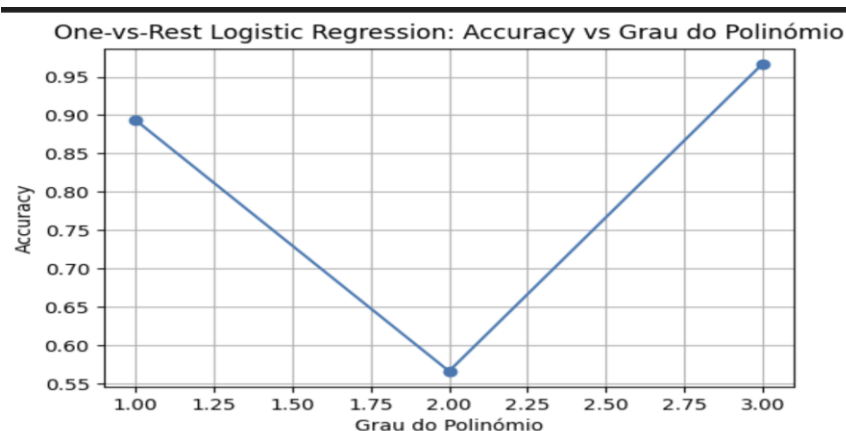
- Usando o dataset Iris, percebe-se o ganho de accuracy, que agora passou a ser 75%

```
Accuracy: 0.7555555555555555
```

```
Confusion Matrix:
```

```
[[15  0  0]
 [ 2  5  8]
 [ 0  1 14]]
```

-Usando o dataset JapanVowels, dependendo do grau seu accuracy muda, chegando no ápice de 96.7%



Conclusão

Percebe-se que esse algoritmo e sua implementação, destaca-se pela robustez e adaptabilidade, especialmente útil em datasets que contêm mais classes ou maior variabilidade.

Estudo Empírico – Softmax

Preparação inicial

- Datasets com mais classes para provar eficácia: Iris(3 classes), Wine(3 classes), JapanVowels(9 classes).
- Hiperparâmetros iguais ao **Logistic Regression e One-vs-Rest**
- Avaliar a eficiência e precisão do Softmax ao modelar diretamente a distribuição de probabilidade entre múltiplas classes, e comparar com One-vs-Rest.
- Estimar desempenho: Hold-out – Iris(70%, 30%) e resto(80%, 20%), treino e teste respectivamente, random state - 42

Análise de resultados

-Usando o dataset Iris,
percebe-se um ganho
ainda maior em relação
ao OvR, accuracy – 95%

```
Accuracy: 0.9555555555555556
```

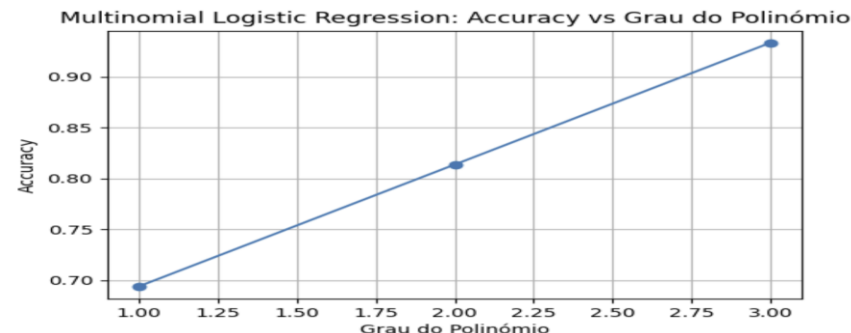
```
Confusion Matrix:
```

```
[[15  0  0]
```

```
[ 0 13  2]
```

```
[ 0  0 15]]
```

-Usando o dataset
JapanVowels, percebe-se que
consegue chegar a um o alto Accuracy
de 93.3%



Conclusão

- Demonstrou ser a abordagem mais eficaz para classificação multiclasse na maioria dos datasets
- Sua capacidade de modelar diretamente as probabilidades de todas as classes de forma conjunta resultou em maior accuracy e relatórios de classificação mais robustos, especialmente em datasets onde as classes são bem separáveis.

Conclusão Final

- A Regressão Logística, na sua forma binária, **não é adequada diretamente para problemas multiclasse**, sendo necessária uma extensão da sua formulação.
- As abordagens **One-vs-Rest** e **Softmax Regression** são alternativas eficazes, apresentando um **aumento significativo de performance** em classificações multiclass.
- Nos nossos testes, ambas as abordagens foram eficazes, mas a **Softmax Regression demonstrou melhor desempenho global**, com decisões mais consistentes entre classes.
- A escolha entre OvR e Softmax deve considerar o **contexto do problema**, incluindo:
 - Número de classes
 - Complexidade dos dados
 - Ruído

Bibliografia

- Machine Learning Mastery. One-vs-Rest and One-vs-One for Multi-Class Classification: <https://www.machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/>
- GeeksforGeeks. One-vs-Rest Strategy for Multi-Class Classification: <https://www.geeksforgeeks.org/one-vs-rest-strategy-for-multi-class-classification/>
- Wikipedia. Função Softmax: https://pt.wikipedia.org/wiki/Fun%C3%A7%C3%A3o_softmax
- GeeksforGeeks. Softmax Activation Function in Neural Networks: <https://www.geeksforgeeks.org/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applications/>