# *Perceptron* Multi-classe

## Otimização em *Machine Learning*

## Grupo A

Bruno José Infante de Sousa, A78997
Bruno Manuel Macedo Nascimento, A67647
João Miguel Freitas Palmeira, A73864
Rafael Machado Silva, A74264
Ricardo Barros Pereira, A77045
Tiago Filipe Gonçalves Vilar Pereira, A61032

Mestrado Integrado em Engenharia Informática,
Universidade do Minho

8 de Junho de 2020

# Índice

# Índice

# Abordagem Científica

- É implementado um neurónio artificial através de um algoritmo, que recebe *input* de outros neurónios.
- Cada conexão realizada com os intervenientes contém um peso.
- **Ativação** refere-se ao somatório de todos os valores de peso proveniente dos *inputs* num neurónio.

$$a = \sum_{d=1}^{D} \omega_d \chi_d$$

- Se $a > 0$, a iteração continua para os próximos neurónios.
- Se $a < 0$, não há ativação.

# Abordagem Científica

$$a = \sum_{d=1}^{D} \omega_d \chi_d$$

- O vector $\chi = (x_1, x_2, \ldots, x_D)$ é o vector de *input*.
- D contém os pesos associados ao neurónio, $\omega_1, \omega_2, \ldots, \omega_D$.
- É aconselhado um *Threshold* com valor diferente de zero para obter previsões positivas.
- É necessário introduzir **bias** no neurónio, para a ativação ser incrementada de forma fixa. É assim criada uma variável $b$ que é adicionada ao valor do somatório.

$$a = \left( \sum_{d=1}^{D} \omega_d \chi_d \right) + b.$$

# Orientação por Erro - Algoritmo de Treino

- O *Perceptron* é um algoritmo de aprendizagem neuronal simples e eficaz.
- O *dataset* é **particionado** e trata cada parte de cada vez.
- Tem a particularidade de ser **orientado pelo erro**, ou seja, enquanto o cálculo não apresentar erros, não atualiza os parâmetros estabelecidos.
- **Calcula** uma parte dos dados e **compara** a previsão com o *dataset*.
- Se a **previsão for correta**, avança sem alterações.
- **Caso contrário** altera os valores para os corretos, antes de avançar para a próxima partição.

# Orientação por Erro - Algoritmo de Treino

**Algoritmo 1:** Treino *Perceptron*

$w_d \leftarrow 0, \forall d = 1...D;$
$b \leftarrow 0;$
**for** $iter = 1 \ldots MaxIter$ **do**
  **forall** $(x, y) \in D$ **do**
    $a \leftarrow \sum_{d=1}^{D} \omega_d \chi_d + b;$
    **if** $ya \leq 0$ **then**
      $w_d \leftarrow w_d + yx_d, \forall d = 1...D$ ;
      $b \leftarrow b + y;$
    **end**
  **end**
**end**
return $w_0, w_1, \ldots, w_D, b$

# Orientação por Erro - Algoritmo de Treino

- Na linha 6 do algoritmo de treino, é verificado se é necessário atualizar os parâmetros.
- Como o rótulo possui valor $+1$ ou $-1$, apenas é preciso verificar o valor do produto $ya$.
- Quando a previsão for correta, o produto $ya$ é positivo.

# Orientação por Erro - Algoritmo de Teste

**Algoritmo 2:** Teste *Perceptron*

1 $\sum_{d=1}^{D} \omega_d \hat{x}_d + b$
2 return SIGN (a)

- Quando ocorre um erro e é calculada uma nova ativação, é necessário atualizar os valores dos pesos e *bias*.
- O peso $\omega_d$ é aumentado por $yx_d$ e o *bias* é incrementado por $y$.
- O objectivo do *update* é ajustar os parâmetros para que futuras iterações possuam previsões certas.

# Interpretação Geométrica

$$B = \left\{ x : \sum_{d=1}^{D} \omega_d \chi_d = 0 \right\}.$$

- $\sum_d \omega_d \chi_d$, é o produto entre o vetor dos pesos, $\omega = \langle w_1, w_2, \ldots, w_D \rangle$, e o vetor vetor de *input*, $\chi = (x_1, x_2, \ldots, x_D)$ .
- Se o resultado do produto $\omega \cdot \chi$ for igual a 0 significa que os vetores são perpendiculares.
- O plano que resultar do produto é o limite da decisão entre os pontos positivos e negativos.
- Se o valor $b \geq 0$, então , o limite de decisão **afasta-se** do vetor $\omega$. Se $b \leq 0$ **aproxima-se** do vetor $\omega$.
- As classificações variam consoante estas movimentações.

# Interpretação dos Pesos do Perceptron

- A forma como o *Perceptron* aprende a classificar modificações durante a última classificação, podem ser respondidas com alterações na derivação.
- Ordena-se desde os valores mais altos até aos mais baixos e de seguida são extraídos os 10 maiores positivos e os 10 maiores negativos.

$$x \mapsto sign\left(\sum_{d=1}^{D} \omega_d \chi_d + b\right)$$

- Os 10 maiores positivos influenciam o modelo a fazer previsões positivas.
- Os 10 maiores negativos têm características que influenciam as previsões negativas.

# Separabilidade Linear

- Diz-se que dois subconjuntos $X$ e $Y$ de dimensão 2 são linearmente separáveis se existir um hiperplano, de maneira que os elementos de X e os de Y se encontrem em lados opostos.

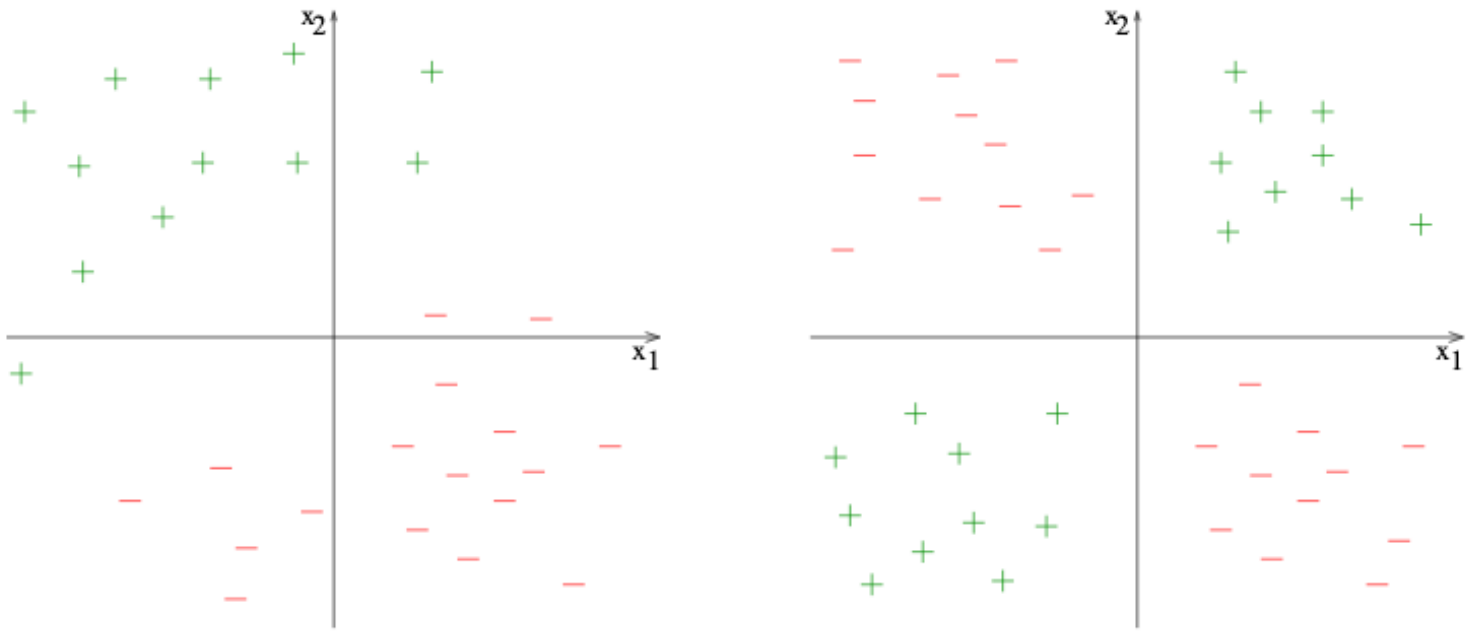


Figura: *Datasets* de dimensão 2 — O *dataset* da esquerda é linearmente separável, o da direita não.

# Convergência do *Perceptron*

- No *Perceptron*, se os dados forem linearmente separáveis, eles convergirão para um vetor de pesos que os separa.
- A **velocidade** de convergência é **proporcional** à **complexidade** do problema em questão.
- Esta complexidade define-se utilizando a noção de *margin*, que é a distância entre do hiperplano ao ponto mais próximo.
- Quanto maior for a *margin*, mais simples é o problema, pois é mais fácil encontrar um hiperplano de separação.

# Convergência do *Perceptron* — *Margin*

$$\text{margin}(D, w, b) = \begin{cases} \min_{(x,y)\in D} = y(w^T x + b), & \text{se } w \text{ separa } D \\ -\infty, & \text{caso contrário.} \end{cases}$$

- A margem é definida apenas se o par $w, b$ separar os dados $D$.
- Será o ponto com ativação mínima, após a ativação ser multiplicada pela *label*.
- Num *dataset* recorre-se a todos os pares $w, b$, para calcular a margem.

# Generalização Melhorada: Votação

- Qualquer alteração nos *inputs* que apareça no final do treino pode tornar inválida toda aprendizagem até esse ponto.
- A solução passa por introduzir **votação** nos hiperplanos que obtiveram maior número de iterações do algoritmo estáveis.
- O voto funciona de forma eficaz até ocorrer uma situação previamente descrita, a equação seguinte é o cálculo tradicional do algoritmo *Vanilla Perceptron*.

$$\hat{y} = sign\left(\sum_{k=1} kc^{(k)} sign(\omega^{(k)} \cdot \hat{x} + b^{(k)})\right)$$

# Generalização Melhorada: Média

- Uma proposta de resolução trata de trocar o voto único por uma média de classificações.
- Aqui guarda-se os pesos e o seu tempo de sobrevivência, mas ao mesmo tempo compara-se com a média obtida até esse ponto e não pela votação do melhor resultado.

$$\hat{y} = sign\left(\sum_{k=1} kc^{(k)}(\omega^{(k)} \cdot \hat{x} + b^{(k)})\right)$$

- A principal diferença reside de *sign* não aparecer no interior do somatório.
- A vantagem desta abordagem permitir manter a soma em qualquer parte do cálculo do hiperplano.

# *Perceptron* e os seus limites

As redes *Perceptron* têm algumas limitações, como por exemplo:

- os valores de saída de um *Perceptron* podem assumir apenas um de dois valores (0 ou 1).
- os *Perceptrons* só podem classificar conjuntos de vetores linearmente separáveis.

# Classificação Multi-classe

- É uma extensão natural da classificação binária, mas neste caso com $K > 2$ classes para escolher.
- Neste tema temos três técnicas bastante utilizadas que vamos abordar mais à frente: *One vs All* (OVA), *One vs One* (OVO) e *Error-Correcting Output Codes* (ECOC).

# *Kernel*

- Uma maneira de fazer com que um modelo linear se comporte de maneira não linear é transformar o *input*. Por exemplo, adicionando pares de *features* como *inputs* adicionais.
- A família de técnicas que torna isso possível é conhecida como abordagens de *kernel*.
- Neste caso particular, o objetivo é que o algoritmo *Perceptron* consiga classificar modelos não lineares através do uso do *kernel*.

# *Kernel*

**Algoritmo 3:** *Perceptron* com *Kernel*

$a \leftarrow 0, b \leftarrow 0$
**for** $iter = 1 \ldots MaxIter$ **do**
  **forall** $(x_n, y_n) \in D$ **do**
    $a \leftarrow \sum_m a_m \phi(x_m).\phi(x_n) + b$
    **if** $y_n a \leq 0$ **then**
      $a_n \leftarrow a_n + y_n$
      $b \leftarrow b + y$
    **end**
  **end**
**end**
return $a, b$

# Índice

# *One vs All*

- Permite transformar problemas de classificação **multi-classe em binários**.
- Utiliza um classificador binário por cada *label*.
- Cada classificador prevê se um *label* pertence a um dado valor da classe.
- Para dados que possuam $n$-classes, são treinados $n$ classificadores, **um classificador por classe**.
- Para o treino de um classificador, toma os valores dessa **classe como amostras positivas** e as **restantes classes como negativas**.

# *One vs All*

- Na fase de pontuação, todo o classificador $n$ prevê a probabilidade de determinada classe e é selecionada a **classe com maior probabilidade**.
- As *labels* de classes discretas podem levar a ambiguidades, onde várias classes são previstas para uma única amostra.
- Um dos problemas desta estratégia é a **escala dos valores de confiança** que pode diferir entre os classificadores binários.
- Mesmo com uma distrbuição de treino equilibrada, vamos obter **classificações desequilibradas**, pois, por norma, o **conjunto de negativos** é muito **maior** que o **conjunto de positivos**.

# *One vs All* - Exemplo

Consideremos o exemplo com **4 classes** (A,B,C,D) e os respetivos **4 classificadores**. Após a **previsão**, cada classificador contém as seguintes **probabilidades**:

- classificador_A = 40%
- classificador_B = 30%
- classificador_C = 60%
- classificador_D = 50%

# *One vs One*

- Considera cada par binário de classes e treina o classificador no subconjunto de dados que contém essas classes.
- Para um problema com $L$ classes, são utilizados $L(L-1)/2$ classificadores binários.
- Cada classificador prevê uma classe e a classe mais prevista é a resposta.
- A previsão é feita através da contagem da incidência dos elementos em cada um dos classificadores.

# *One vs One* - Exemplo

Consideremos o seguinte exemplo com **4 classes** e os respetivos **6 classificadores**:

- classificador_AB atribui a classe A
- classificador_AC atribui a classe A
- classificador_AD atribui a classe A
- classificador_BC atribui a classe B
- classificador_BD atribui a classe D
- classificador_CD atribui a classe C

Neste caso, a escolha recai pela **classe A** pela maioria das atribuições apesar de não existirem as probabilidades.

# *Error-Correcting Output Codes*

- Estratégia Baseada em Matriz de Códigos
- Para um problema com $L$ classes, são utilizados $2^{L-1} - 1$ classificadores binários
- Todos os classificadores prevêem uma *code-word*, é escolhida a classe mais parecida.
- Capacidade de superação de eventuais erros

# Índice

# Conjuntos de dados

- *Dataset digits*: 1797 imagens de tamanho 8x8
- *Dataset MNIST*: 60000 imagens de treino e 10000 imagens de teste com um tamanho de 28x28

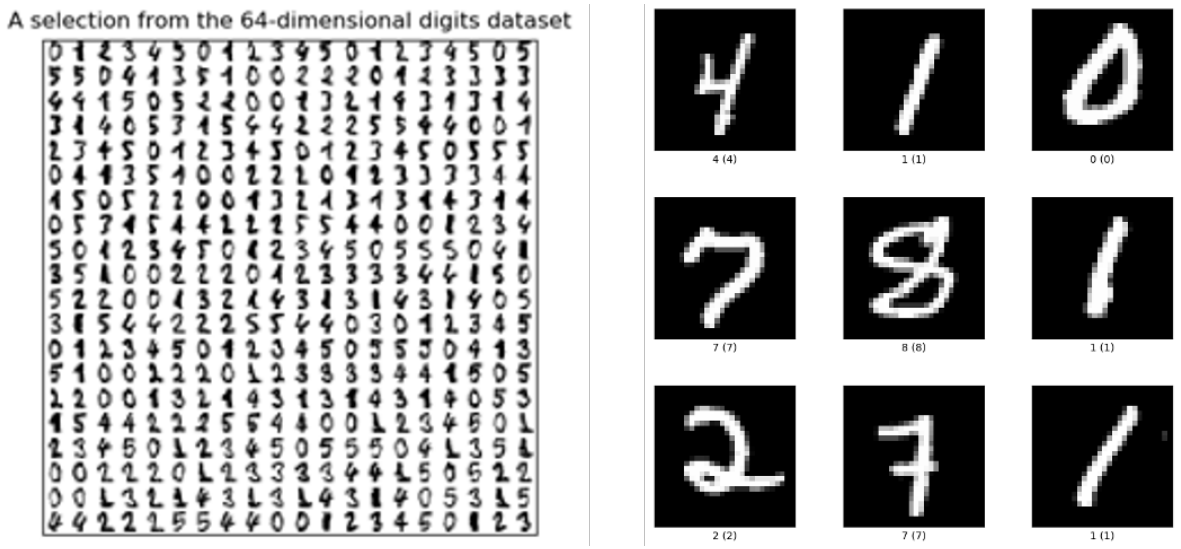


Figura: Conjuntos de dados: *digits* e MNIST

# Estrutura principal

- Dois tipos de *perceptron*: primal e dual
- A cada um dos *perceptrons* aplicou-se uma das três estratégias (OVA, OVO, ECOC)

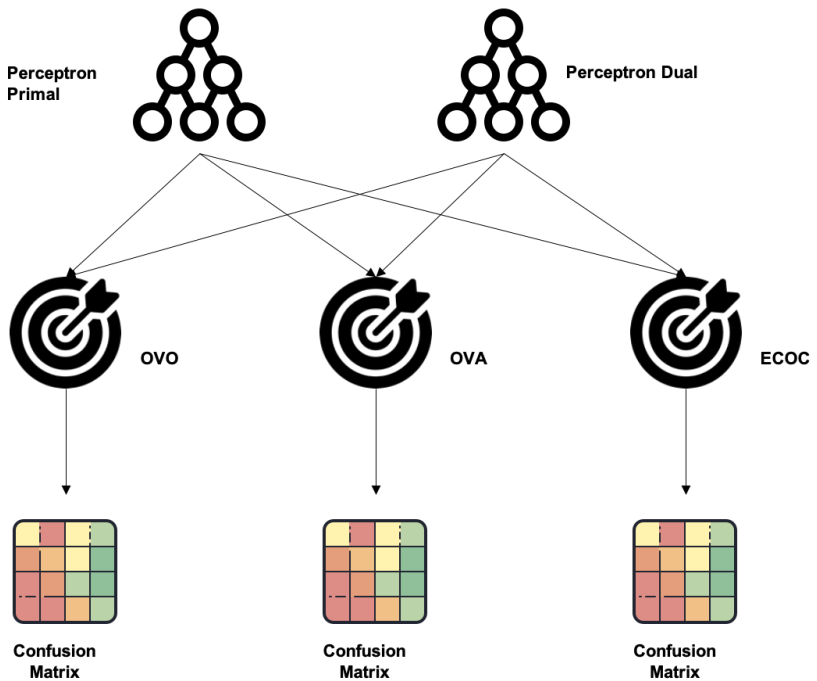


Figura: Estratégia do projeto

# Estrutura principal

- Desenvolvidos três *scrips*:
  - **server.py**
  - **MCP_Primal.py**
  - **MCP_Dual.py**
- *Main* do projeto - *server.py*:
  - *Load* dos *datasets*
  - Visualização dos *clusters* (PCA — regiões de decisão)
  - Cálculo de cada modelo
  - *Score* para cada tipo de classificador

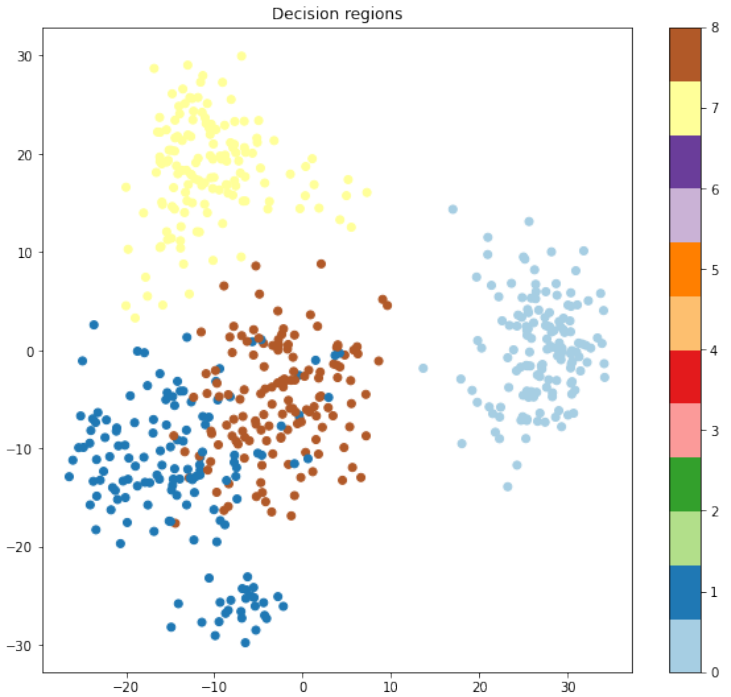


Figura: Regiões de decisão — *Clusters*

# *Perceptron*

- *Perceptron* primal (*MCP_Primal.py*)
  - *Learning rate*
  - Número máximo de iterações (épocas)
  - Tipo de classificador
- *Perceptron* dual (*MCP_Dual.py*)
  - *Kernel*
  - Tipo de classificador

*Kernel*:

- linear: $k(x, y) = x * y$
- RBF: $k(x, y) = e^{-\alpha||x-y||^2}$
- Polinomial: $k(x, y) = (x * y)^d$

# *One vs All*

Implementação do Classificador Binário **OVA**:

- Definir número de classificadores (igual ao nnúmero de classes)
- Alterar valores binários para valores de "+1"ou -1"
- Aplica-se o *fit* ao classificador
- Previsão com os dados dos classificadores criados
- *Argmax* da matriz de previsões

# *One vs One*

Implementação do Classificador Binário **OVO**:

- Definir o número de classificadores ($(n(n-1))/2$, onde n é o número de classes)
- Alterar valores binários para valores de "+1"ou -1"
- Definir dados de treino e *target*
- Estratégia de voto:
  - Criar amostras positivas e negativas
  - Calcula-se o *argmax* das amostras
  - Determina-se a classe mais predominante no classificar
- *Argmax* da matriz de previsões (semelhante ao OVA)

# *Error-Correcting Output Codes*

Implementação do Classificador Binário **ECOC**:

- Calcular o número de classificadores mediante as classes pretendidas
- Construir uma matriz *Random* para todos os classificadores e transformar os valores para 1 e $-1$
- Treinar todos os classificadores agrupando as classes consoante a matriz criada
- Prever as *code-words*, calcular as distâncias e associar às classes idênticas

# *Error-Correcting Output Codes* – Exemplo

Tabela: 7 ECOC para 4 classes

| **Classes** | **Code Word** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| $C_1$ | 1 | $-1$ | $-1$ | 1 | $-1$ | 1 | $-1$ |
| $C_2$ | $-1$ | 1 | $-1$ | $-1$ | 1 | 1 | 1 |
| $C_3$ | $-1$ | $-1$ | 1 | 1 | 1 | $-1$ | $-1$ |
| $C_4$ | 1 | $-1$ | 1 | $-1$ | 1 | $-1$ | 1 |

No classificador 4 separa se em dois grupos $\{C_1, C_3\}$ de $\{C_2, C_4\}$ para o treino.

# Classificadores *Scikit-learn*

- Como extra, utilizou-se a livraria *Scikit-learn* para se aplicar classificadores pré-definidos aos conjuntos de dados
- **Método** (3 *scripts*, uma para cada classificador):
  - *Load* do conjunto de dados
  - Criação de um *perceptron* para um dos três tipos de classificadores
  - Aplicar um classificador ao *perceptron* (ex: OneVsRestClassifier)
  - Aplica-se um *fit* e um *predict* ao modelo e utiliza-se a função *score* para retirar a *accuracy*
  - Cálculo da matriz de confusão (*metrics.confusion_matrix*)

# Interface Gráfica

- Como extra, optamos por criar um **ambiente gráfico interativo**, para permitir ao utilizador testar os diferentes clasificadores multiclasse que elaboramos.
- O objetivo é proporcionar uma experiência mais agradável desde a seleção dos classificadores, dos parâmetros, das classes ou dos conjuntos de dados, até à visualização dos resultados obtidos.

# Índice

# Classificadores Multi-Classe — *Dataset Digits*

| Perceptron | Classificadores | Accuracy | Time |
|---|---|---|---|
| Primal | *OneVsAll* | 98.6% | 0.05s |
| | *OneVsOne* | 99.3% | 0.02s |
| | *Error-Correcting Output Codes* | 98.6% | 0.1s |
| Dual | *OneVsAll* | 98.6% | 38.2s |
| | *OneVsOne* | 99.3% | 5.2s |
| | *Error-Correcting Output-Codes* | 97.2% | 67s |
| Dual com Kernel RBF | *OneVsAll* | 96.5% | 45.1s |
| | *OneVsOne* | 99.3% | 17.3s |
| | *Error-Correcting Output-Codes* | 97.9% | 71.9s |
| Dual com Kernel Polinomial | *OneVsAll* | 100% | 20.3s |
| | *OneVsOne* | 99.3% | 6.3s |
| | *Error-Correcting Output-Codes* | 100% | 35.2s |

# Classificadores Multi-classe — *Dataset Mnist*

| Perceptron | Classificadores | Accuracy | Time |
|---|---|---|---|
| Primal | *OneVsAll* | 91.2% | 0.2s |
| | *OneVsOne* | 96.8% | 0.2s |
| | *Error-Correcting Output Codes* | 93.1% | 0.6s |
| Dual | *OneVsAll* | 91.2% | 156.8s |
| | *OneVsOne* | 96.8% | 18.4s |
| | *Error-Correcting Output-Codes* | 93.7% | 285.6s |
| Dual com Kernel RBF | *OneVsAll* | 94.3% | 83.5s |
| | *OneVsOne* | 93.7% | 30.3s |
| | *Error-Correcting Output-Codes* | 95.6% | 126.6s |
| Dual com Kernel Polinomial | *OneVsAll* | 96.2% | 65.1s |
| | *OneVsOne* | 93.1% | 12.2s |
| | *Error-Correcting Output-Codes* | 96.8% | 76.1s |

# Classificadores Scikit-learn — *Dataset Digits*

- Resultados com *Scikit-learn* e *Perceptron* Primal utilizando o *Dataset Digits*

| **Classificadores/Resultados** | **Accuracy** | **Time** |
|---|---|---|
| *OneVsOne* | 95.7% | 0.29s |
| *OneVsAll* | 95.7% | 0.31s |
| *Error-Correcting Output-Codes* | 95.0% | 0.27s |

# Índice

# Conclusão

- Investigação e análise sobre *Perceptron* multi-classe
- Objetivos propostos alcançados
- Objetivos extra realizados (*Scikit-learn* e *Interface* Gráfica)
- Resultados obtidos

# *Perceptron* Multi-classe

## Otimização em *Machine Learning*
## Grupo A

Bruno José Infante de Sousa, A78997
Bruno Manuel Macedo Nascimento, A67647
João Miguel Freitas Palmeira, A73864
Rafael Machado Silva, A74264
Ricardo Barros Pereira, A77045
Tiago Filipe Gonçalves Vilar Pereira, A61032

Mestrado Integrado em Engenharia Informática,
Universidade do Minho

8 de Junho de 2020