

Lista 2 de Machine Learning

Gabriel Dias Vilela

IMPA, Verão 2025

Sumário

1 Exercício 1a	3
2 Exercício 1b	4
3 Exercício 1c	5
4 Exercício 1d	6
5 Exercício 1e	7
6 Exercício 1f	8
7 Exercício 1g	9
8 Exercício 1h	10
9 Exercício 2a	12
10 Exercício 2b	14
11 Exercício 2c	19
12 Exercício 3a	21
13 Exercício 3b	22
14 Exercício 3c	24
15 Exercício 3d	25
16 Exercício 3e	26
17 Exercício 3f	27
18 Exercício 3g	29
19 Exercício 4a	30

20 Exercicio 4b	32
21 Exercicio 4c	33
22 Exercicio 4d	35
23 Exercicio 4e	37
24 Exercicio 4f	38

1 Exercício 1a

Suponha $\tilde{y}_i \in \{-1, 1\}$. Seja a perda dada por:

$$\begin{aligned} L_t(\tilde{\alpha}, \hat{\theta}_t) &= \sum_{i=1}^n w_{it} \exp(-\tilde{y}_i \tilde{\alpha} F_t(x_i, \hat{\theta}_t)). = \sum_{y_i=F(x_i, \hat{\theta}_t)} w_{it} \exp(-\tilde{\alpha}) + \sum_{y_i \neq F(x_i, \hat{\theta}_t)} w_{it} \exp(\tilde{\alpha}) = \\ &= W_t^= e^{-\tilde{\alpha}} + W_t^{\neq} e^{\tilde{\alpha}}, \end{aligned}$$

Primeiro, calculamos a derivada de L_t em relação a $\tilde{\alpha}$:

$$\frac{\partial L_t}{\partial \tilde{\alpha}} = -W_t^= e^{-\tilde{\alpha}} + W_t^{\neq} e^{\tilde{\alpha}},$$

Depois, da segunda derivada:

$$\frac{\partial^2 L_t}{\partial \tilde{\alpha}^2} = W_t^= e^{-\tilde{\alpha}} + W_t^{\neq} e^{\tilde{\alpha}} > 0 \text{ (pois cada termo é positivo)}$$

concluímos que L_t é função convexa em $\tilde{\alpha}$. Para achar o $\tilde{\alpha}$ que minimiza L_t , igualamos a primeira derivada a zero:

$$\frac{\partial L_t}{\partial \tilde{\alpha}} = 0 \implies -W_t^= e^{-\tilde{\alpha}} + W_t^{\neq} e^{\tilde{\alpha}} = 0 \implies W_t^= e^{-\tilde{\alpha}} = W_t^{\neq} e^{\tilde{\alpha}} \implies e^{\tilde{\alpha}} = \sqrt{\frac{W_t^=}{W_t^{\neq}}} = e^{\hat{\alpha}_t}$$

Substituindo $e^{\tilde{\alpha}}$ de volta em L_t , concluímos que

$$L_t(\hat{\alpha}_t, \hat{\theta}_t) = 2 \sqrt{W_t^= W_t^{\neq}}.$$

Nota-se ainda que, para a iteração anterior:

$$L_{t-1}(\tilde{\alpha}_{t-1}, \tilde{\theta}_{t-1}) = \frac{1}{n} \sum_{i=1}^n \exp[-\tilde{y}_i (\hat{f}_{t-2}(x_i) + \tilde{\alpha}_{t-1} F_{t-1}(x_i, \tilde{\theta}_{t-1}))] = \frac{1}{n} \sum_{i=1}^n \exp[-\tilde{y}_i \hat{f}_{t-1}] = \sum_{i=1}^n w_{it} = W_t^= + W_t^{\neq}$$

2 Exercício 1b

Tome

$$p_i = \frac{w_{it}}{\sum_{i=1}^n w_{it}} = \frac{w_{it}}{W_t^= + W_t^{\neq}} \quad (\text{pesos normalizados para respeitar o domínio da probabilidade}).$$

Seja F um algoritmo γ -fraco na iteração t . Assim, existe $\hat{\theta}$ tal que

$$\sum_{i=1}^n \frac{w_{it}}{W_t^= + W_t^{\neq}} \mathbf{I}_{(\tilde{y}_i \neq F(x_i, \hat{\theta}))} \leq \frac{1}{2} - \gamma \implies \frac{1}{W_t^= + W_t^{\neq}} \sum_{i:(\tilde{y}_i \neq F(x_i, \hat{\theta}))} w_i \leq \frac{1}{2} - \gamma$$

Note que omitimos t na ultima passagem.

3 Exercício 1c

Seja F um algoritmo γ -fraco na iteração t . Isso significa que existe um classificador $\hat{\theta}_t$ tal que

$$\sum_{i=1}^n p_i \mathbf{I}_{(F(x_i, \hat{\theta}_t) \neq \tilde{y}_i)} \leq \frac{1}{2} - \gamma.$$

Escrevendo essa soma em termos dos pesos $W_t^=$ e W_t^{\neq} :

$$\frac{1}{W_t^= + W_t^{\neq}} \sum_{\tilde{y}_i \neq F(x_i, \hat{\theta}_t)} w_i \leq \frac{1}{2} - \gamma.$$

Porém, por definição,

$$\sum_{\tilde{y}_i \neq F(x_i, \hat{\theta}_t)} w_i = W_t^{\neq}.$$

Logo,

$$\frac{W_t^{\neq}}{W_t^= + W_t^{\neq}} \leq \frac{1}{2} - \gamma.$$

4 Exercício 1d

A partir da expressão obtida acima e sabendo que $W_t^=$ e W_t^{\neq} são maiores do que zero (e que γ é pequeno em relação a 1 de tal modo que $1 - 2\gamma > 0$), obtemos:

$$2W_t^{\neq} \leq (1 - 2\gamma)(W_t^= + W_t^{\neq}) \implies W_t^{\neq}(1 + 2\gamma) \leq (1 - 2\gamma)W_t^= \implies \frac{1 + 2\gamma}{1 - 2\gamma}W_t^{\neq} \leq W_t^=$$

5 Exercício 1e

Como $L_t(\hat{\alpha}_t, \hat{\theta}_t) \leq L_t(\tilde{\alpha}, \hat{\theta}_t)$, basta provar que

$$L_t(\tilde{\alpha}, \hat{\theta}_t) \leq (1 - 4\gamma^2)^{1/2} L_{t-1}(\hat{\alpha}_{t-1}, \hat{\theta}_{t-1}).$$

Assim:

$$\begin{aligned} L_t(\tilde{\alpha}, \hat{\theta}_t) &= W_t^= e^{-\tilde{\alpha}} + W_t^{\neq} e^{\tilde{\alpha}} = W_t^{\neq} \sqrt{\frac{1+2\gamma}{1-2\gamma}} + W_t^= \sqrt{\frac{1-2\gamma}{1+2\gamma}} = \\ &= (1 - 4\gamma^2)^{1/2} \left(\frac{W_t^=}{1+2\gamma} + \frac{W_t^{\neq}}{1-2\gamma} \right) \end{aligned}$$

Como queremos mostrar que

$$L_t(\tilde{\alpha}, \hat{\theta}_t) \leq \sqrt{1 - 4\gamma^2} L_{t-1}(\hat{\alpha}_{t-1}, \hat{\theta}_{t-1}),$$

basta verificar que

$$\begin{aligned} L_{t-1}(\hat{\alpha}_{t-1}, \hat{\theta}_{t-1}), = W_t^= + W_t^{\neq} &\geq \frac{W_t^=}{1+2\gamma} + \frac{W_t^{\neq}}{1-2\gamma} \iff W_t^{\neq} \frac{1}{1-2\gamma} \leq W_t^= \frac{1}{1+2\gamma} \iff \\ &\iff W_t^{\neq} \left(\frac{1+2\gamma}{1-2\gamma} \right) \leq W_t^=. \end{aligned}$$

Ora, como já constatado no item (d), a desigualdade acima é verdadeira. Logo, isso conclui a demonstração.

6 Exercício 1f

Utilizando a desigualdade mostrada no item anterior:

$$L_t(\hat{\alpha}_t, \hat{\theta}_t) \leq \sqrt{1 - 4\gamma^2} L_{t-1}(\hat{\alpha}_{t-1}, \hat{\theta}_{t-1}) \leq \dots \leq (1 - 4\gamma^2)^{\frac{t}{2}} L_0(\hat{\alpha}_0, \hat{\theta}_0).$$

Utilizando a desigualdade $1 + x \leq e^x$, $x \in \mathbb{R}$, e tomando $L_0 = 1$:

$$L_t(\hat{\alpha}_t, \hat{\theta}_t) \leq e^{-\frac{4\gamma^2 t}{2}} = e^{-2\gamma^2 t}.$$

7 Exercício 1g

Para encontrar T^* tal que $L_t(\hat{\alpha}_t, \hat{\theta}_t) < \frac{1}{n}$, usamos a relação de decaimento exponencial da perda:

$$L_t(\hat{\alpha}_t, \hat{\theta}_t) \leq e^{-2t\gamma^2}.$$

Impondo a condição $e^{-2t\gamma^2} < \frac{1}{n}$ e tomando logaritmos naturais:

$$-2t\gamma^2 < -\ln n \quad \Rightarrow \quad t > \frac{\ln n}{2\gamma^2}.$$

Logo, o número mínimo de iterações necessárias é:

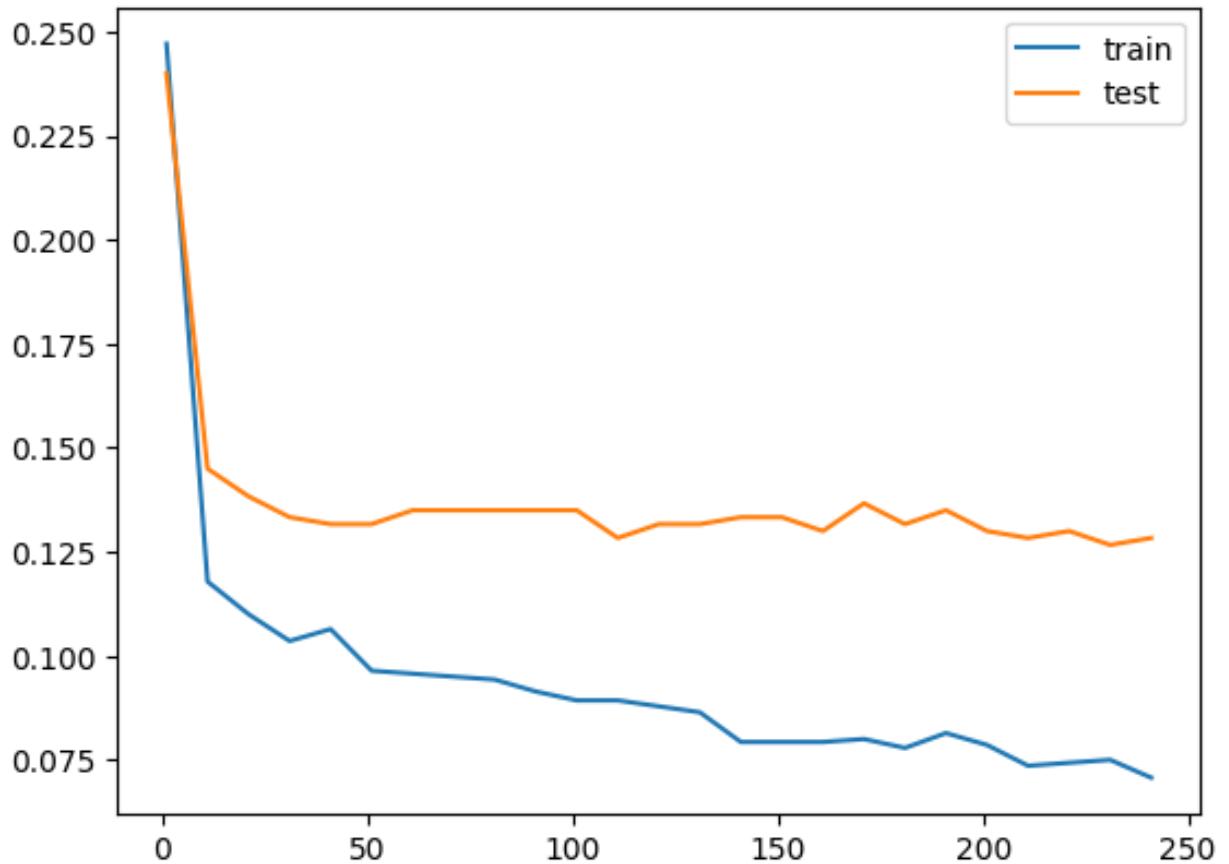
$$T^* = \left\lceil \frac{\ln n}{2\gamma^2} \right\rceil.$$

Como $L_t(\hat{\alpha}_t, \hat{\theta}_t)$ é a média dos termos $e^{-y_i f_t(x_i)}$, ter $L_t(\hat{\alpha}_t, \hat{\theta}_t) < \frac{1}{n}$ implica que cada termo individualmente é pequeno, ou seja, $y_i f_t(x_i)$ é suficientemente grande e positivo. Isso significa que $f_t(x_i)$ tem o mesmo sinal de y_i para todos os exemplos de treino, garantindo acurácia de treino igual a 1.

8 Exercício 1h

Completando o código, obtemos:

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from tqdm import tqdm
5 from sklearn.ensemble import AdaBoostClassifier
6 from sklearn.datasets import make_classification
7 from sklearn.model_selection import train_test_split
8
9 # Cria um problema de classificacao artificial
10 X, y = make_classification(n_samples=2000, n_features=10, n_informative=8,
11     n_redundant=0, random_state=0, shuffle=False)
12
13 # Separa dados em treino e teste
14 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
15     random_state=0)
16
17 train_errors = []
18 test_errors = []
19
20 interval = np.arange(1, 250, 10)
21
22 for i in tqdm(interval):
23     clf = AdaBoostClassifier(n_estimators=i, random_state=0)
24     clf.fit(X_train, y_train)
25     train_errors.append(1 - clf.score(X_train, y_train))
26     test_errors.append(1 - clf.score(X_test, y_test))
27
28 plt.plot(interval, train_errors, label="train")
29 plt.plot(interval, test_errors, label="test")
30 plt.legend()
```



Nota-se, pela curva, que os erros de treino e teste apresentam de fato um decaimento aproximadamente exponencial em função do número de iterações do AdaBoost. Além disso, observa-se que o erro de treino tende a diminuir mais do que o erro de teste e que, a partir de um certo ponto (aproximadamente 50 iterações), o erro de teste se estabiliza dentro de uma faixa específica. Isso indica a ocorrência de overfitting para $t > 50$ iterações, pois, enquanto o erro de treino continua diminuindo, o erro de teste permanece estável e a distância entre as curvas aumenta.

Adicionalmente, partindo da conclusão do item anterior, podemos supor que o erro de treino irá convergir para 0 a medida que $t \rightarrow T^* = \left\lceil \frac{\ln n}{2\gamma^2} \right\rceil$

9 Exercício 2a

Código usado para treino, teste e cálculo do tempo de execução:

```
1 def get_time_to_train(model, X_train, y_train):
2     start = time.time()
3     model.fit(X_train, y_train)
4     end = time.time()
5     print(f"Tempo para treinar o modelo {model}: {end - start}")
6     return model
7
8 def get_time_to_predict(model, X_test, y_test):
9     start = time.time()
10    y_pred = model.predict(X_test)
11    end = time.time()
12    print(f"Tempo para prever com o modelo {model}: {end - start}")
13    print(f"Acrúacia do modelo {model}: {np.sum(y_pred == y_test)/len(y_test)}")
14    return y_pred
15
16 nb = get_time_to_train(nb, X_train, y_train)
17 lda = get_time_to_train(lda, X_train, y_train)
18 qda = get_time_to_train(qda, X_train, y_train)
19 lr = get_time_to_train(lr, X_train, y_train)
20 knn = get_time_to_train(knn, X_train, y_train)
21 svc = get_time_to_train(svc, X_train, y_train)
22 rf = get_time_to_train(rf, X_train, y_train)
23 nn = get_time_to_train(nn, X_train, y_train)
24
25 y_pred_nb = get_time_to_predict(nb, X_test, y_test)
26 y_pred_lda = get_time_to_predict(lda, X_test, y_test)
27 y_pred_qda = get_time_to_predict(qda, X_test, y_test)
28 y_pred_lr = get_time_to_predict(lr, X_test, y_test)
29 y_pred_knn = get_time_to_predict(knn, X_test, y_test)
30 y_pred_svc = get_time_to_predict(svc, X_test, y_test)
31 y_pred_rf = get_time_to_predict(rf, X_test, y_test)
32 y_pred_nn = get_time_to_predict(nn, X_test, y_test)
```

A partir deste código, os tempos de treino obtidos foram:

```
1 Tempo para treinar o modelo BernoulliNB(): 0.8559560775756836
2 Tempo para treinar o modelo LinearDiscriminantAnalysis(): 29.260023832321167
3 Tempo para treinar o modelo QuadraticDiscriminantAnalysis(): 14.72248125076294
4 Tempo para treinar o modelo LogisticRegression(random_state=42):
   11.886216640472412
5 Tempo para treinar o modelo KNeighborsClassifier(n_neighbors=6): 0.126478910446167
6 Tempo para treinar o modelo SVC(C=100, class_weight='balanced'): 212.8464126586914
7 Tempo para treinar o modelo RandomForestClassifier(max_depth=30, random_state=0):
   54.42084717750549
8 Tempo para treinar o modelo MLPClassifier(hidden_layer_sizes=[100], max_iter=300,
   random_state=42): 95.84694528579712
```

Já as acruacias e os tempos gastos nas previsões foram:

```
1 Tempo para prever com o modelo BernoulliNB(): 0.14716768264770508
2 Acrúacia do modelo BernoulliNB(): 0.8348571428571429
3 Tempo para prever com o modelo LinearDiscriminantAnalysis(): 0.039583683013916016
4 Acrúacia do modelo LinearDiscriminantAnalysis(): 0.8677142857142857
5 Tempo para prever com o modelo QuadraticDiscriminantAnalysis(): 2.6857593059539795
```

```

6 Acrucia do modelo QuadraticDiscriminantAnalysis(): 0.5312857142857143
7 Tempo para prever com o modelo LogisticRegression(random_state=42):
    0.03498053550720215
8 Acrucia do modelo LogisticRegression(random_state=42): 0.9202857142857143
9 Tempo para prever com o modelo KNeighborsClassifier(n_neighbors=6):
    11.559824228286743
10 Acrucia do modelo KNeighborsClassifier(n_neighbors=6): 0.969
11 Tempo para prever com o modelo SVC(C=100, class_weight='balanced'):
    123.24246501922607
12 Acrucia do modelo SVC(C=100, class_weight='balanced'): 0.9824285714285714
13 Tempo para prever com o modelo RandomForestClassifier(max_depth=30, random_state
    =0): 0.8036403656005859
14 Acrucia do modelo RandomForestClassifier(max_depth=30, random_state=0):
    0.9673571428571428
15 Tempo para prever com o modelo MLPClassifier(hidden_layer_sizes=[100], max_iter
    =300, random_state=42): 0.05862832069396973
16 Acrucia do modelo MLPClassifier(hidden_layer_sizes=[100], max_iter=300,
    random_state=42): 0.9755

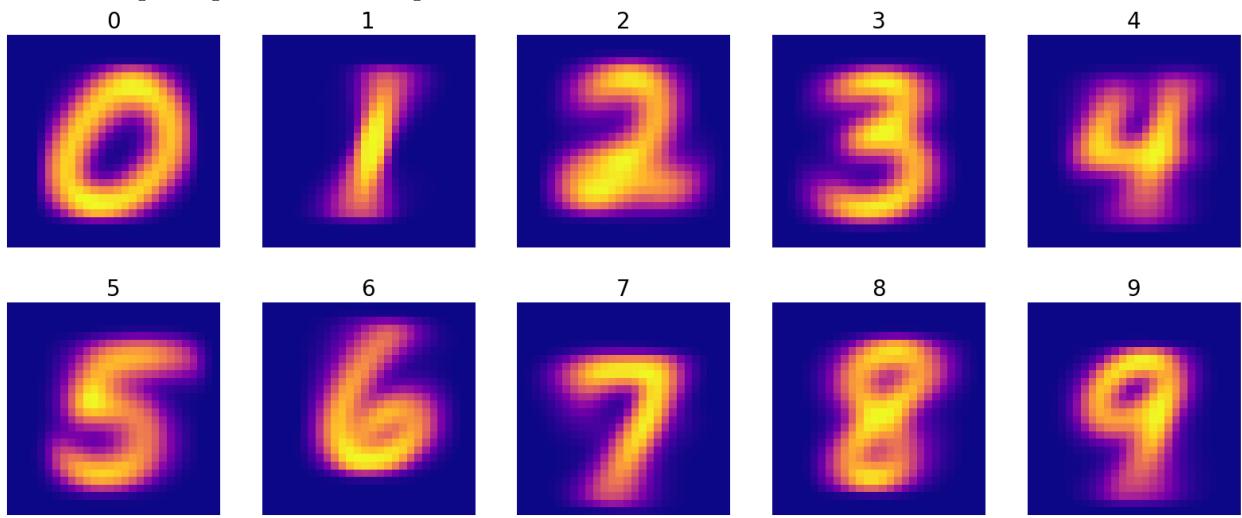
```

Assim, concluímos que os modelos vistos na primeira parte do curso são, no geral (e na média), treináveis mais rapidamente se comparados aos modelos vistos na segunda parte do curso. O mesmo porém não se pode afirmar sobre o tempo necessário para realizar previsões pois um comportamento misto é observado quando se compara os modelos do inicio do curso com os modelos do final do curso (por exemplo, dentre os 5 modelos que realizam previsões mais rapidamente, 3 pertencem a primeira metade do curso e 2 pertencem a segunda metade). Contudo, na média, os modelos da primeira metade realizam previsões mais rápido se comparados aos modelos da segunda metade.

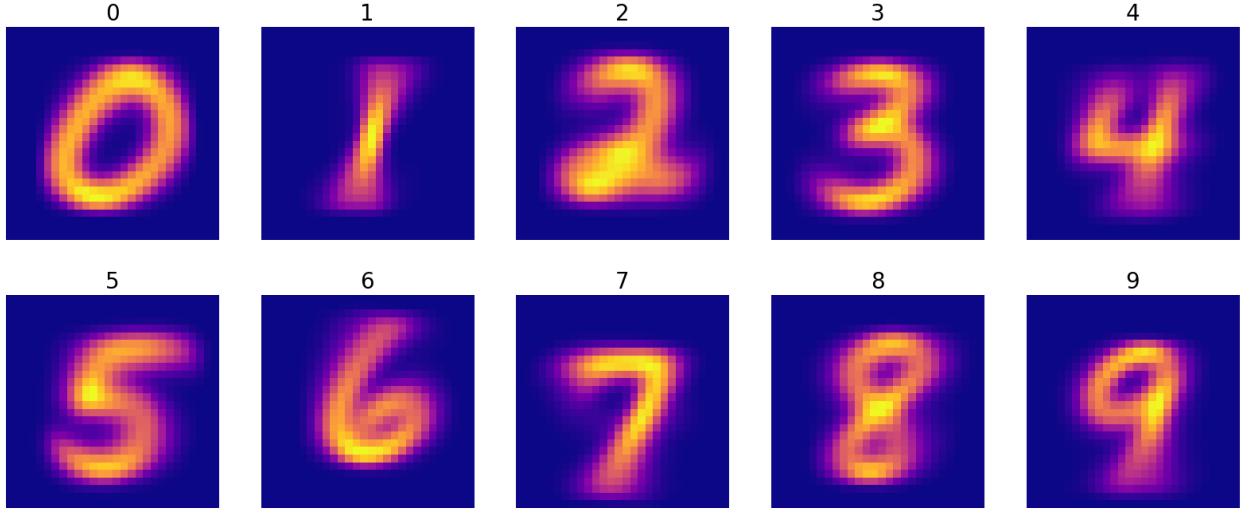
Quanto a acrúacia, verificamos que os modelos vistos na segunda metade do curso apresentam no geral (e na média) uma acrúacia maior.

10 Exercício 2b

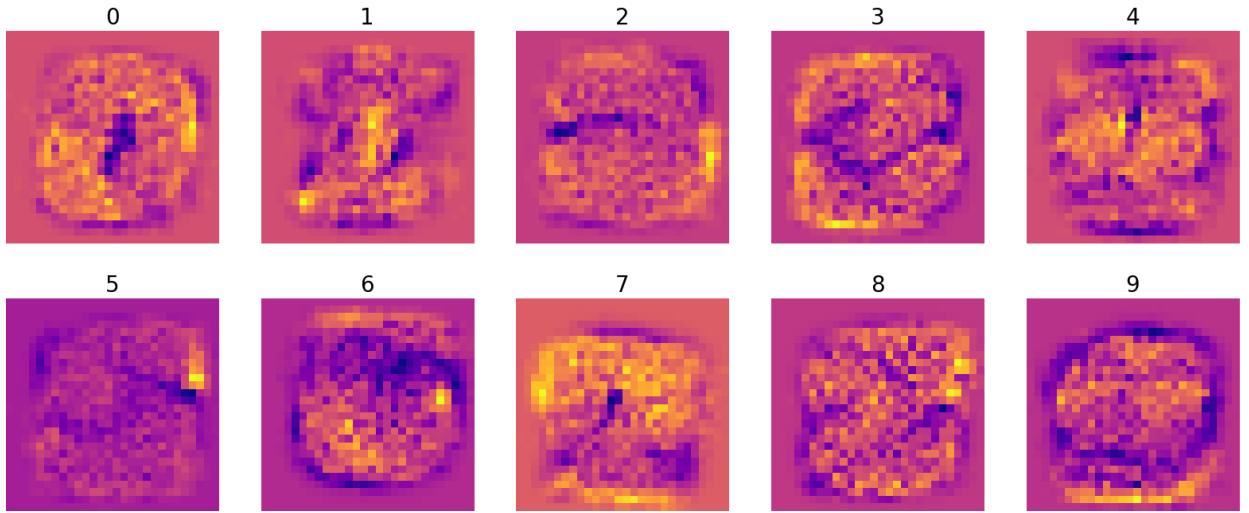
(i): A imagem gerada a partir deste código exibe como um heatmap a probabilidade de cada pixel estar ativado para cada classe de acordo com os parâmetros do modelo NB treinado. Desse modo, é esperado que esse heatmap exiba uma imagem próxima à do número correspondente a sua classe pois os pixels mais prováveis de estarem ativos neste contexto são aqueles que esboçam o desenho do número. O resultado pode ser facilmente interpretado e mostra, de certa forma, como o modelo "enxerga" cada classe (ou melhor, quais foram os principais padrões extraídos pelo modelo relativo a cada classe). Além disso, essa imagem se relaciona com as previsões pois, dado um input, o modelo buscará classificar esse input de acordo com a sua similaridade com o heatmap, buscando especialmente pelos pixels com alta probabilidade de estarem ativos.



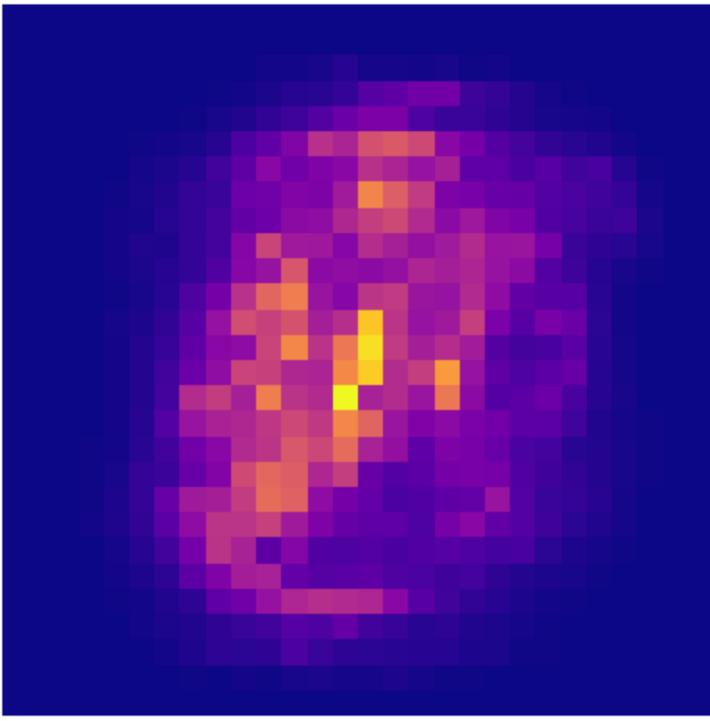
(ii): A imagem gerada a partir deste código exibe, como um heatmap, a média do valor (ativado/desativado) de cada pixel para cada classe de acordo com o modelo LDA treinado. Assim, essa imagem fornece um panorama geral do comportamento das amostras de cada classe, bem como indica quais são os padrões mais comuns em cada uma (as zonas com cor mais intensa na imagem). Tal imagem é facilmente interpretável e exibe visualmente quais padrões o modelo irá buscar ao classificar um novo número.



(iii): A imagem gerada a partir deste código exibe, na forma de um heatmap e para cada classe, o coeficiente associado a cada pixel na equação linear que define a probabilidade do input pertencer a essa classe em função das features. Dessa forma, a imagem transmite uma noção da importância de cada pixel na determinação da classe de uma amostra. Observamos que, embora a imagem seja menos clara do que as anteriores, ainda é possível extrair informações relevantes para compreender o funcionamento do modelo na classificação. Por exemplo, no caso do número 1, a região central da imagem parece exercer um impacto significativo no cálculo da probabilidade de uma imagem pertencer a essa classe realizado pelo modelo.



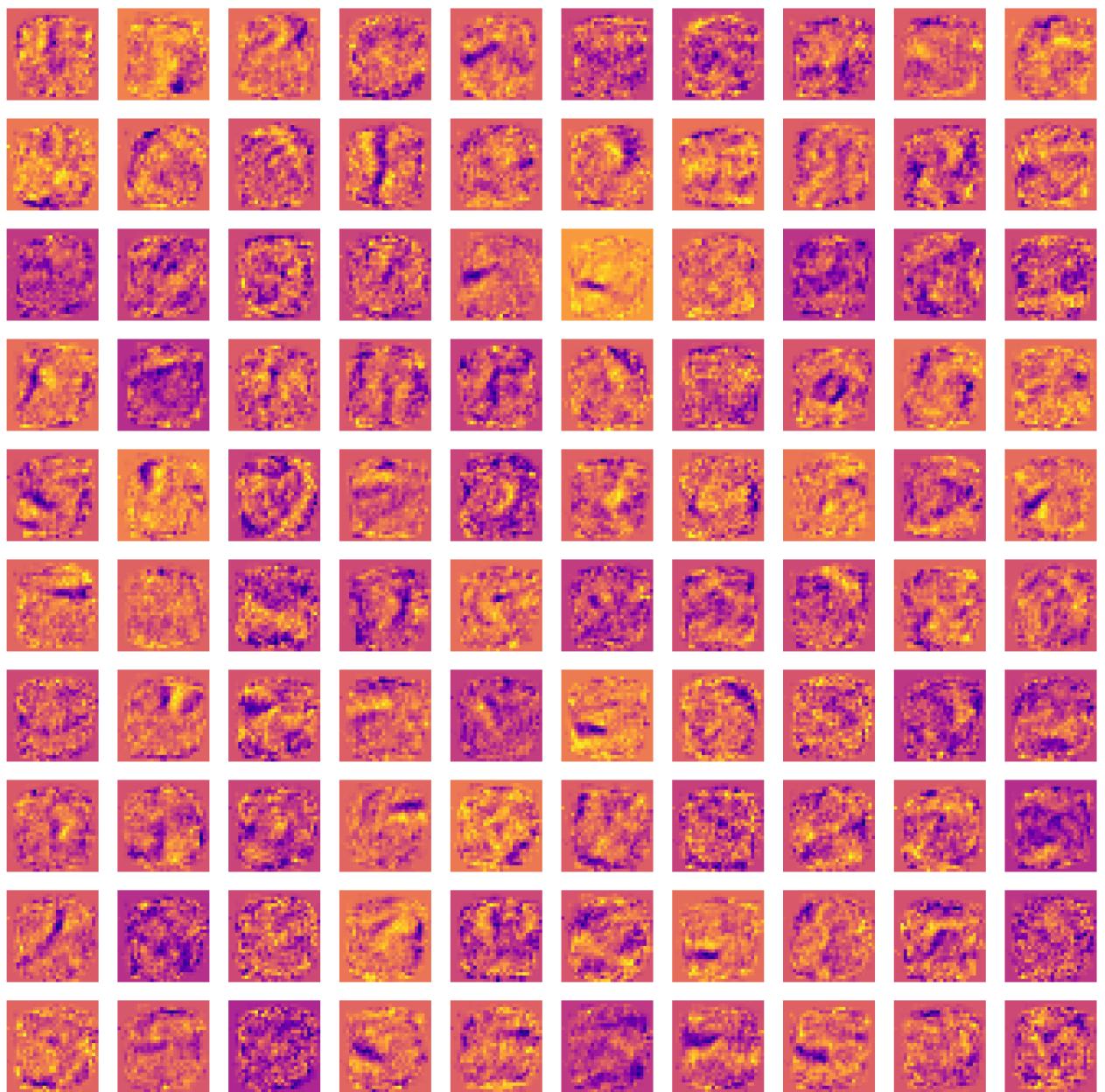
(iv): A imagem gerada a partir deste código exibe, na forma de um heatmap, o valor da importância de Gini de cada feature para a árvore de decisão treinada. Assim, podemos extrair da imagem quais features são as mais relevantes para que o modelo decida em qual classe enquadrar uma amostra. Como se pode notar na imagem, seu centro parece ser a região com padrões mais importantes para determinar a classe das amostras.

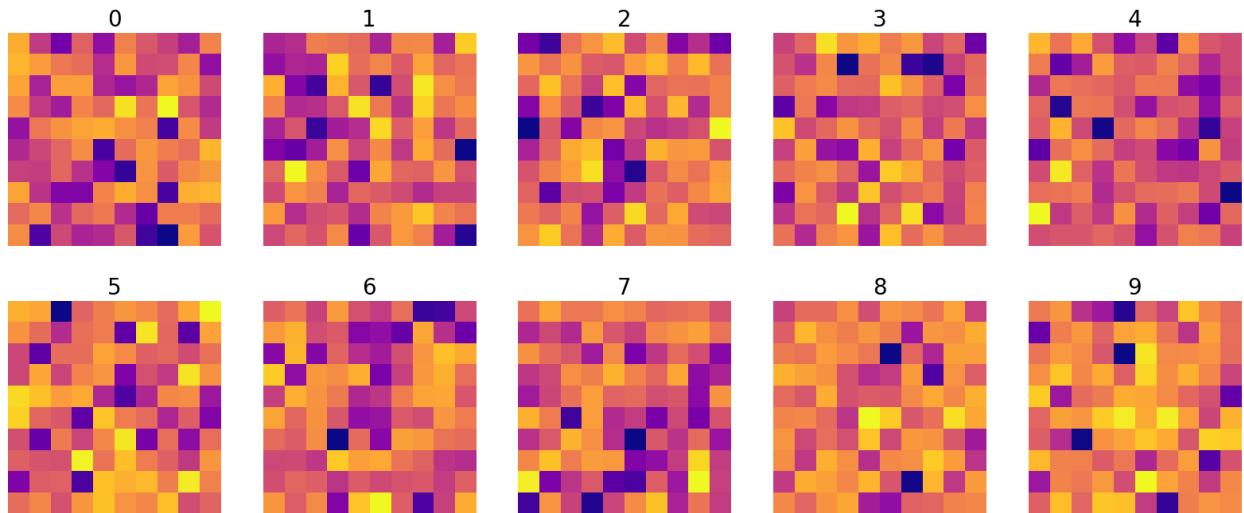


(v): As imagens geradas a partir do código exibem, respectivamente:

- 1) Os pesos atribuídos a cada feature (ou seja, a matriz de pesos W) em cada *hiddenlayer* da rede neural.
- 2) Os pesos para cada feature na *outputlayer*.

Neste caso, tais imagens são de difícil interpretação devido ao grau de complexidade de uma rede neural e à possível falta de significado da matriz de pesos de uma camada específica. Ainda sim, podemos extrair da primeira imagem uma noção de qual tipo de padrão cada camada da rede neural tenta capturar.





11 Exercício 2c

A partir do código abaixo:

```
1 confusion_matrix(y_test, y_pred_nb)
```

Foi obtido a seguinte matrix de confusão:

```
1 array([[1198,      1,      6,     13,      3,     63,     31,      1,     25,
2      [ 0, 1540,      8,      8,      1,     18,      3,      1,     17,
3      [ 17,    25, 1133,     39,     29,      7,     57,     12,     55,
4      [  8,    36,     67, 1148,      3,     31,     13,     22,     60,
5      [  4,     7,      8,      0, 1042,      8,     22,      6,     26,
6      [ 31,    22,      9,   190,     38,    906,     25,      7,     17,
7      [ 15,    42,    32,      2,     19,     33, 1250,      0,      3,
8      [  8,    28,     11,      5,     34,      3,      0, 1274,     30,
9      [ 17,    55,     23,   103,     14,     45,      6,      6, 1033,
10     [ 11,    26,      8,     11,   101,      7,      0,     55,     37,
11     [ 11,    26,      8,     11,   101,      7,      0,     55,     37, 1164]], ,
12      dtype=int64)
```

Todas as justificativas abaixo foram feitas com base na visualização do heatmap gerado no item b)(i) que representa os padrões mais prováveis para cada classe de acordo com o modelo NB treinado.

Classe 0: O erro mais comum é prever 5. Notamos que há similaridade entre o 0 e 5 quanto a parte das bordas dos números (superior, inferior e lateral inferior direita). Todas essas bordas são, de acordo com o heatmap, padrões muito prováveis de se encontrar no 5.

Classe 1: O erro mais comum é prever 5. Notamos que 5 e 1 apresentam, aproximadamente, um padrão provável em comum na região superior da imagem, inferior da imagem e ao longo do eixo paralelo à imagem do número 1 passando por seu centro geométrico.

Classe 2: O erro mais comum é prever 6. Notamos que ambos 2 e 6 possuem um padrão provável em comum na região inferior da imagem.

Classe 3: O erro mais comum é prever 2. Notamos que o 2 e o 3 possuem um padrão provável em comum na parte superior e inferior da imagem.

Classe 4: O erro mais comum é prever 9. Notamos que o 4 e o 9 apresentam padrões prováveis em comum em grande parte da imagem. A única exceção é a parte superior da imagem, onde o 9 apresenta pixels provavelmente ativos e o 4 não.

Classe 5: O erro mais comum é prever 3. Notamos que o 5 e o 3 apresentam como padrão provável em comum toda a região abaixo da altura média da imagem.

Classe 6: O erro mais comum é prever 1. Notamos que 6 e 1 apresentam, aproximadamente, um padrão provável em comum na região superior da imagem e ao longo do eixo paralelo à imagem do número 1 passando por seu centro geométrico.

Classe 7: O erro mais comum é prever 9. Notamos que o 7 e o 9 apresentam padrões prováveis em comum em grande parte da imagem. A única exceção é a região superior esquerda dela,

onde o 9 apresenta pixels provavelmente ativos e o 7 não.

Classe 8: O erro mais comum é prever 3. Notamos que o 3 e o 8 apresentam como padrão provável em comum a região direta, a partir da largura média, da imagem.

Classe 9: O erro mais comum é prever 4. Notamos que o 4 e o 9 apresentam padrões prováveis em comum em grande parte da imagem. A única exceção é a parte superior da imagem, onde o 9 apresenta pixels provavelmente ativos e o 4 não.

12 Exercício 3a

Essa visualização representa uma matriz de atenção (*att*) 32x32 aplicada a um conjunto de 32 amostras compostas por uma frase (vetor) e um output (classe). Essa matriz de atenção é construída a partir do produto entre b e b^T , onde b é uma matriz com 32 linhas (cada amostra) e 8025 colunas (tamanho do vocabulário de treino definido pelo modelo de vetorização).

Note que essa matriz *att* (definida pelo produto interno 2 a 2 entre os vetores que representam os inputs) representa uma relação entre as amostras e , portanto, carrega consigo uma relação entre os outputs (tema dos artigos).

Assim, essa visualização ilustra como cada output se relaciona com os outros, em termos de semelhança, de acordo com um mecanismo de atenção definido através do produto escalar entre vetores de features. Podemos abstrair essa atenção como uma representação de correlação entre os textos de cada artigo analisado.

13 Exercício 3b

(i)

Seja $P_u(v)$ a projeção ortogonal de v sobre u .

Sabemos que $v - P_u(v) \perp u \implies \langle v - P_u(v), u \rangle = 0 \implies \langle v, u \rangle = \langle P_u(v), u \rangle$.

Como $P_u(v)$ é paralelo a u , existe $k \in \mathbb{R}$ tal que $P_u(v) = k u$.

$$\text{Logo, } \langle v, u \rangle = \langle k u, u \rangle = k \langle u, u \rangle = k \|u\|^2 \implies k = \frac{\langle v, u \rangle}{\|u\|^2}.$$

$$\text{Portanto, } P_u(v) = \frac{\langle v, u \rangle}{\|u\|^2} u.$$

(ii)

Seja o triângulo retângulo ABC (retângulo em B), onde:

$$AB = \|v - P_u(v)\|, \quad BC = \|P_u(v)\|, \quad CA = \|v\|.$$

$$\text{Pelo triângulo retângulo, } \cos(\theta) = \frac{BC}{CA} = \frac{\|P_u(v)\|}{\|v\|}.$$

$$\text{Sabendo que } P_u(v) = \frac{\langle v, u \rangle}{\|u\|^2} u \implies \|P_u(v)\| = \left\| \frac{\langle v, u \rangle}{\|u\|^2} u \right\| = \frac{|\langle v, u \rangle|}{\|u\|^2} \|u\| = \frac{|\langle v, u \rangle|}{\|u\|}.$$

$$\text{Assim, } \cos(\theta) = \frac{\|P_u(v)\|}{\|v\|} = \frac{|\langle v, u \rangle|/\|u\|}{\|v\|} = \frac{|\langle v, u \rangle|}{\|v\| \|u\|}.$$

(iii)

Seja o triângulo ABC definido anteriormente. Defina D um ponto colinear a B e C tal que $DC = \|u_i\|$ e $DA = \|u_i - v_i\|, i \in \{1, 2\}$

Pela Lei dos Cossenos, para o par (u_1, v_1) :

$$\|u_1 - v_1\|^2 = \|u_1\|^2 + \|v_1\|^2 - 2 \|u_1\| \|v_1\| \cos(u_1, v_1).$$

Analogamente, para o par (u_2, v_2) :

$$\|u_2 - v_2\|^2 = \|u_2\|^2 + \|v_2\|^2 - 2 \|u_2\| \|v_2\| \cos(u_2, v_2).$$

Como $\|u_i\| = \|v_i\| = 1$, então:

$$\cos(u_1, v_1) = \frac{2 - \|u_1 - v_1\|^2}{2}, \quad \cos(u_2, v_2) = \frac{2 - \|u_2 - v_2\|^2}{2}.$$

Comparando $\cos(u_1, v_1)$ e $\cos(u_2, v_2)$, obtemos:

$$\cos(u_1, v_1) \geq \cos(u_2, v_2) \iff 2 - \|u_1 - v_1\|^2 \geq 2 - \|u_2 - v_2\|^2 \iff \|u_1 - v_1\| \leq \|u_2 - v_2\|.$$

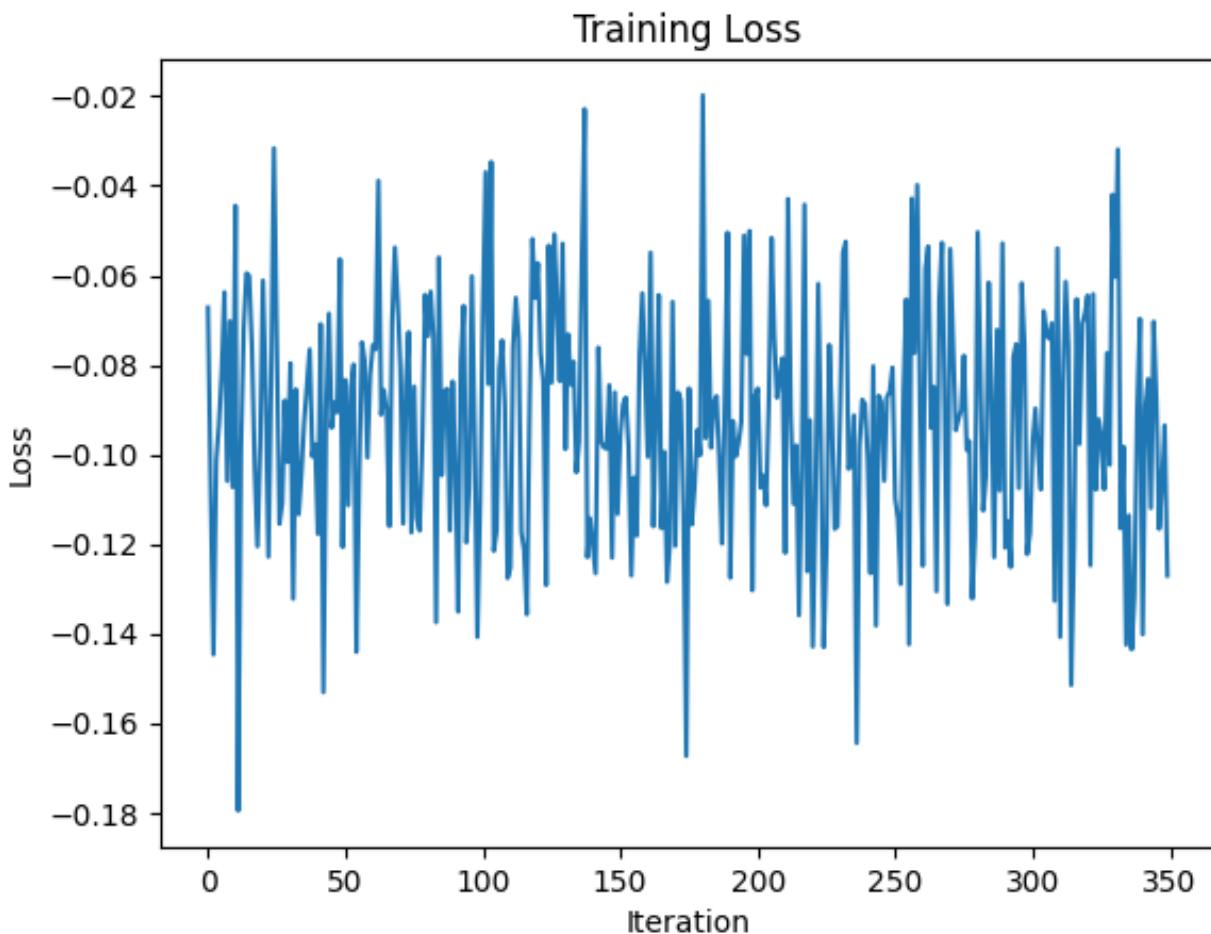
(iv) Note que o cosseno captura uma noção de distância compreensível quando é possível obter intersecção entre os vetores na direção definida por seus versores. Do contrário, quando tais vetores

definem subespaços reversos, o cosseno se torna de difícil interpretação. Além disso, ao extrapolarmos o ângulo para valores fora do primeiro quadrante, o cosseno passa a assumir valores negativos, distorcendo a interpretabilidade da métrica e retornando “distâncias” negativas e de mesmo módulo apesar de os vetores estarem mais afastados (em termos de distância euclidiana) se comparados ao caso em que o ângulo se encontra no primeiro quadrante.

Contudo, apesar dessas possíveis desvantagens quando tentamos pensar no cosseno em termos euclidianos, utilizá-lo como distância é interessante quando buscamos dar enfoque na diferença de direção entre os dois vetores.

14 Exercício 3c

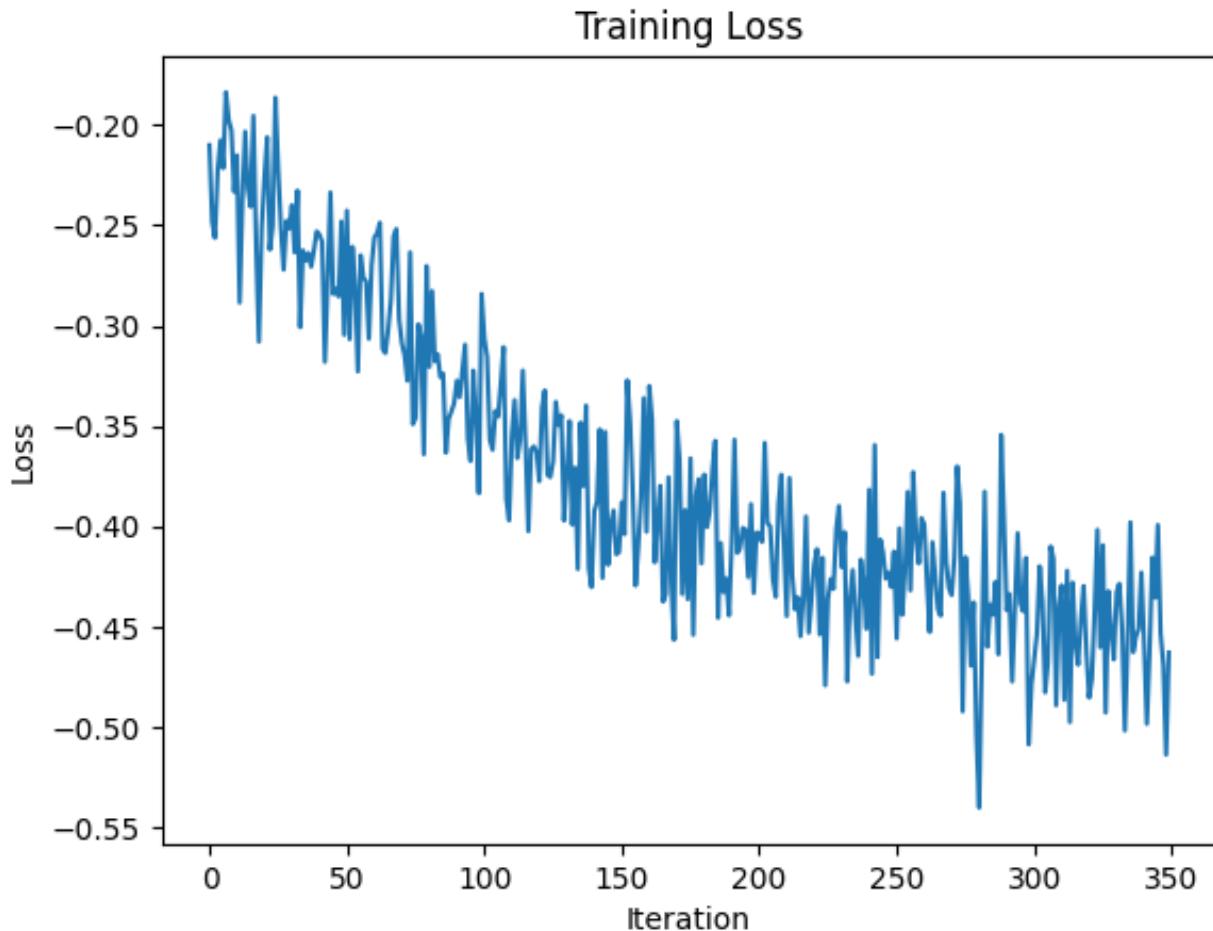
A rede não está aprendendo provavelmente por conta de uma inicialização ruim das camadas lineares. Como a normalização não é feita, nosso modelo fica suscetível a dissipações ou explosões de gradiente que implicam em uma baixa taxa de aprendizado. Para visualizar isso, podemos medir a sequência de erros a cada iteração de treino do modelo:



Como é possível notar, a perda não apresenta comportamento monotônico decrescente. Na realidade, ela oscila ao redor de uma média assumindo valores dentro de uma faixa fixa. Portanto, podemos concluir que a rede não está "aprendendo".

15 Exercício 3d

Descomentando a parte do código comentada, passamos a normalizar a inicialização das camadas lineares. Portanto, espera-se uma taxa de aprendizado maior devido à melhora na inicialização. Ao medir a sequência de erros em cada iteração de treino do modelo, obtemos:



De fato, como podemos visualizar, a média móvel do erro parece apresentar comportamento monotônico decrescente. Ou seja, podemos concluir que a rede neural está de fato "aprendendo".

16 Exercício 3e

(i): O produto $K[X]^T Q[X]$ é uma matriz $n \times n$, e seu elemento (i, j) será dado por:

$$(K[X]^T Q[X])_{ij} = \sum_{k=1}^D K[X]_{ki} Q[X]_{kj}$$

Como as entradas de $K[X]$ e $Q[X]$ têm média zero e são independentes entre si, temos:

$$\mathbb{E}[K[X]_{ki} Q[X]_{kj}] = \mathbb{E}[K[X]_{ki}] \cdot \mathbb{E}[Q[X]_{kj}] = 0 \cdot 0 = 0$$

Ou seja, a média de cada entrada (i, j) de $K[X]^T Q[X]$ é zero.

Como as entradas $K[X]_{ki}$ e $Q[X]_{kj}$ são independentes e seguem uma distribuição com variância σ^2 , temos:

$$\text{Var}\left(\sum_{k=1}^D K[X]_{ki} Q[X]_{kj}\right) = \sum_{k=1}^D \text{Var}(K[X]_{ki} Q[X]_{kj}) = \sum_{k=1}^D \mathbb{E}[K[X]_{ki}^2] \cdot \mathbb{E}[Q[X]_{kj}^2] = \sum_{k=1}^D \sigma^4 = D\sigma^4$$

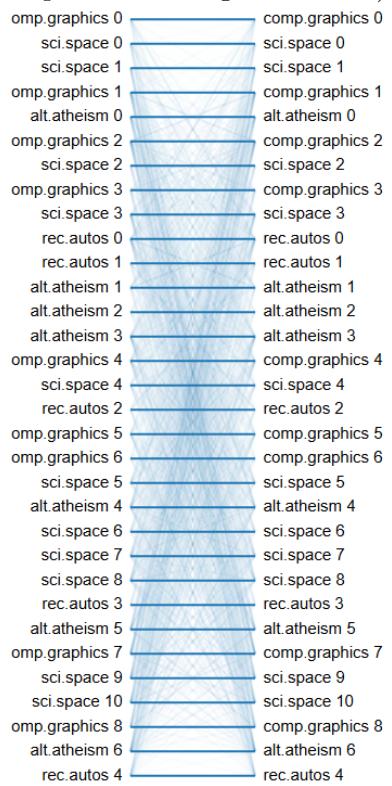
Ou seja, cada entrada de $K[X]^T Q[X]$ tem variância $\sigma^4 D$.

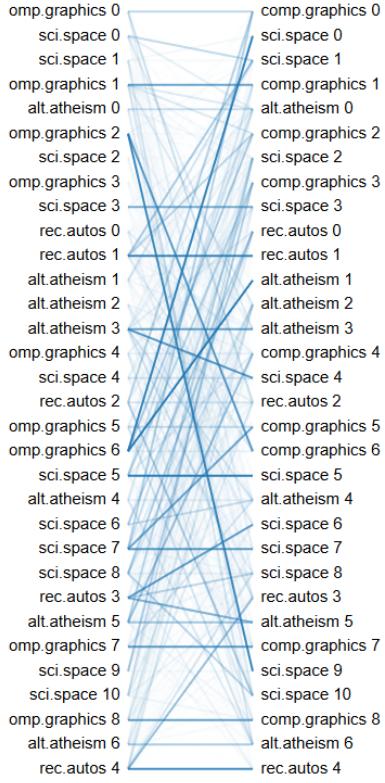
Portanto, a divisão do produto matricial analisado por $D^{1/2}$ tem por objetivo reduzir a variância das entradas da matriz resultante (ao valor de σ^4). Essa diminuição na variação reduz a probabilidade de ocorrência de gradientes dissipativos ou explosivos durante o treinamento da rede neural.

(ii): Não é necessário assumir que as entradas sejam gaussianas visto que as únicas informações sobre as distribuições que foram utilizadas ao longo da questão foram suas médias e suas variâncias. Além disso, não é necessário que as entradas apresentem a mesma distribuição, apenas que suas distribuições apresentem mesma média 0 e mesma variância σ^2 .

17 Exercício 3f

Segue abaixo, respectivamente, a figura obtida para o item a) e a figura obtida com a matriz *att* gerada a partir do código do item e).





É possível notar que, utilizando o produto escalar para medir similaridade, geramos um mecanismo de medição de similaridade simétrico onde cada artigo é similar (em termos de seu input, ou seja do texto que o compõe) apenas a si mesmo praticamente. Em contrapartida, utilizando uma camada de self-attention, obtivemos um mecanismo de medição de similaridade assimétrico onde cada artigo pode ser semelhante não apenas a si mesmo, mas também a muitos outros artigos distintos. Desse modo, ao deixarmos a rede aprender a similaridade entre os artigos, construímos um mecanismo mais robusto que parece de fato carregar informações relevantes sobre a similaridade entre os distintos tipos de tema dos artigos.

18 Exercício 3g

Na implementação de uma rede neural de um LLM de propósito geral tais detalhes são essenciais. A matriz V é responsável por realizar uma mudança semântica e sintática nas palavras a partir do contexto gerado pelas matrizes K e Q. Assim, no caso de modelos capazes de entender/gerar frases baseados em contextos, o uso de V é importante para um melhor entendimento dos inputs textuais por parte do modelo. Já o mecanismo de multi-head attention, no contexto de LLM, é responsável por extrair múltiplos significados complementares para o texto inputado (significado semântico, sintático, morfológico etc...). Essa extração também melhora o entendimento dos inputs textuais por parte do modelo.

19 Exercício 4a

A Figura 1 representa a imagem original. A Figura 2 exibe a imagem clusterizada por um K-Means com 3 clusters, onde o que está plotado são os centróides de cada cluster definido. A Figura 3 representa a diferença entre a imagem original e a clusterizada, fornecendo uma noção da distância (ou erro) de cada pixel da imagem original em relação ao centróide do cluster ao qual pertence. Essa figura permite visualizar o erro da clusterização: quanto mais escura é a imagem, melhor é a clusterização.

Além disso, à medida que o valor da variável *parameter* aumenta, o modelo K-Means divide os pixels em um maior número de clusters, resultando em uma maior variedade de cores na Figura 2 e adicionando mais detalhes à imagem.

Figura gerada com *parameter* = 3:

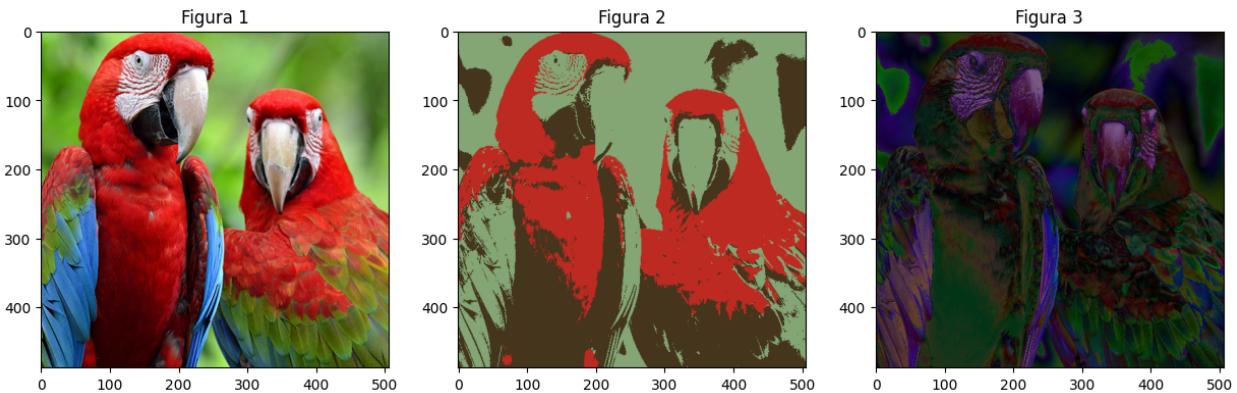


Figura gerada com *parameter* = 10:

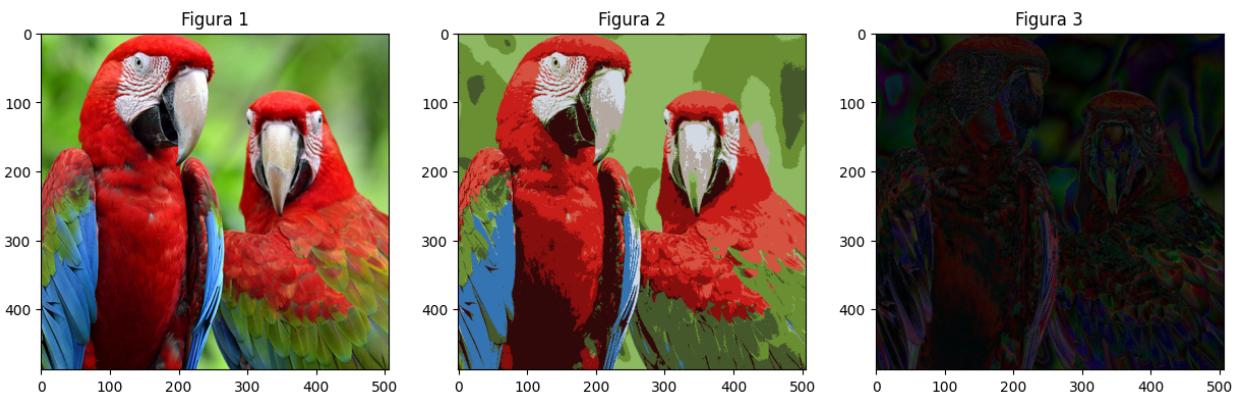
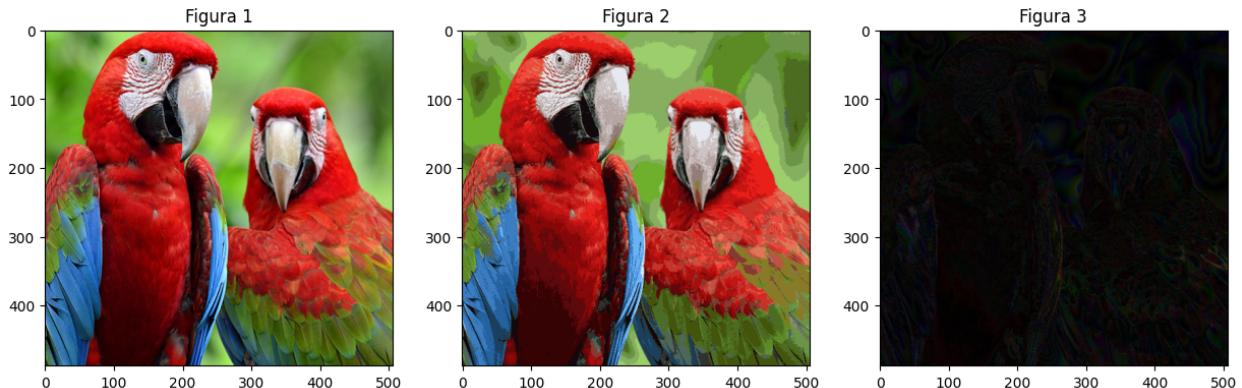


Figura gerada com *parameter* = 30:



20 Exercício 4b

O número de bits por pixel necessário em função do número de clusters é dado por:

$$bits = \log_2(n_{clusters})$$

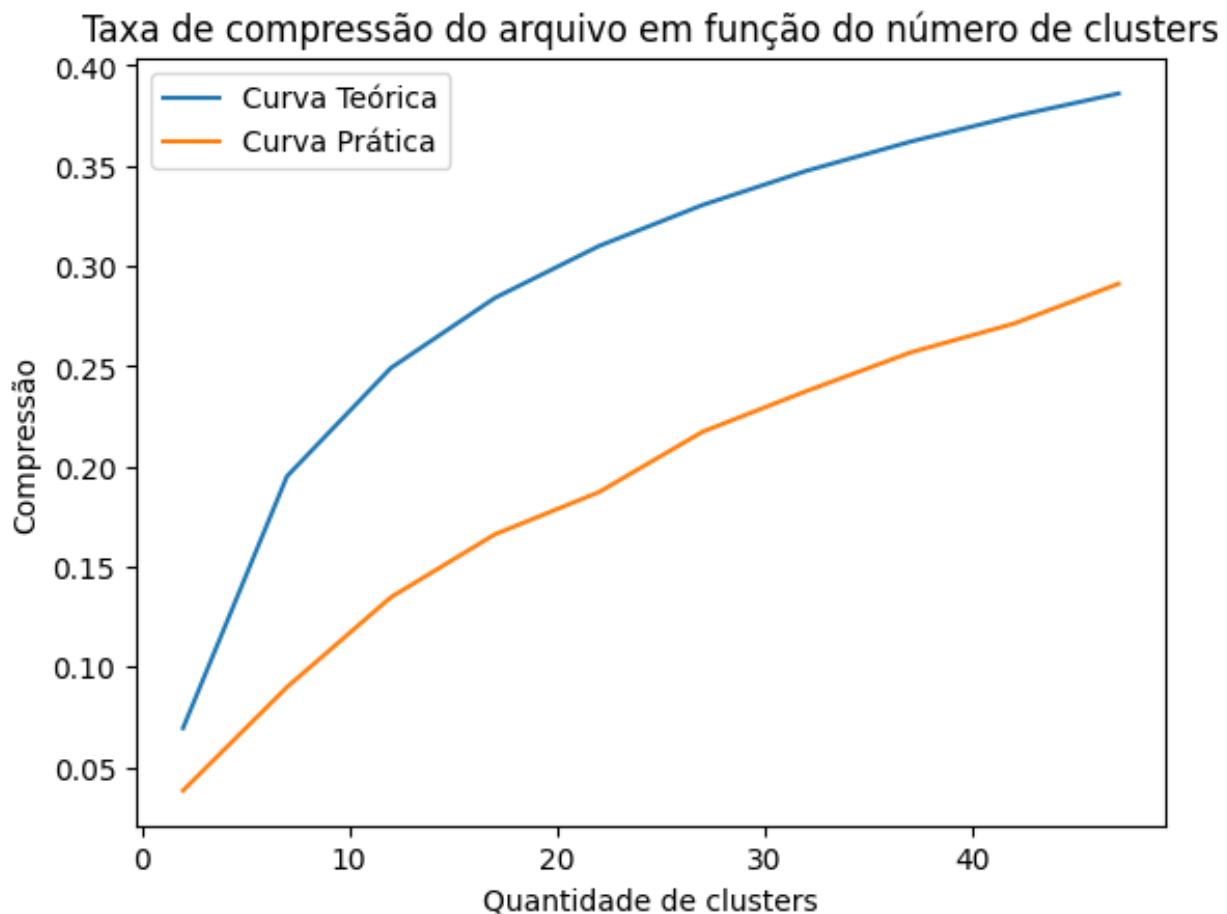
Afinal, buscamos diferenciar os pixels uns dos outros baseado apenas no cluster ao qual eles pertencem. Assim, tomamos o menor número de bits necessários para se construir uma quantidade suficiente de números binários de modo que cada número binário represente um cluster.

Note que os centroídes deverão possuir 24 bits pois eles carregam a informação da cor de seu cluster. Porém, supondo que a quantidade de pixels é muito maior que a quantidade de centroídes, podemos desprezar essa diferença. Essa será a abordagem utilizada no item abaixo.

21 Exercício 4c

Foi utilizado o seguinte código para gerar o gráfico:

```
1 from math import log
2
3 IMG_PIXELS = 506 * 488
4 BITS_TO_BYTES = 8
5
6 # Curva teorica
7 plt.plot(parameter_interval, np.array([log(i, 2) for i in parameter_interval]) *
       IMG_PIXELS / BITS_TO_BYTES / original_size)
8
9 # Curva pratica
10 plt.plot(parameter_interval, sizes)
11
12 plt.xlabel("Quantidade de clusters")
13 plt.ylabel("Compressão")
14 plt.title("Taxa de compressão do arquivo em função do número de clusters")
15 plt.legend(["Curva Teórica", "Curva Prática"])
```

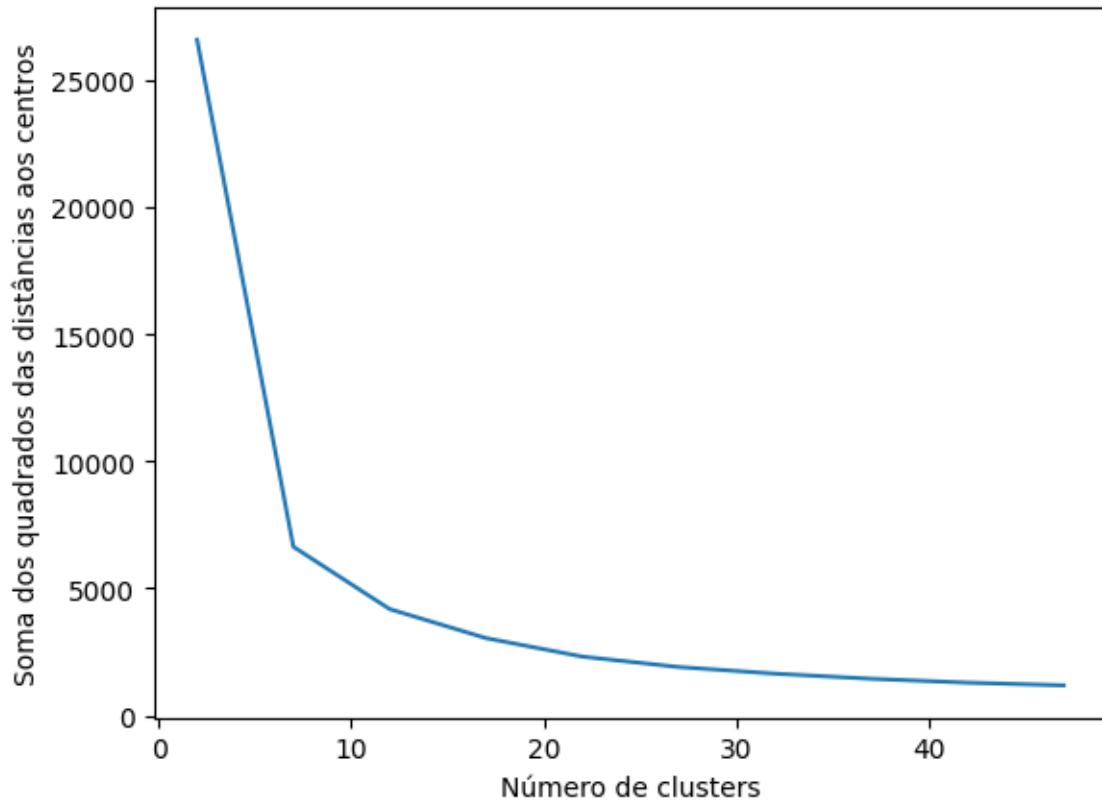


Acima, verificamos a curva teórica em azul e o resultado obtido com a compressão real das imagens em laranja (curva prática).

Como podemos verificar, a compressão real segue aproximadamente a curva teórica, o que indica que fizemos uma boa aproximação sugerindo a função logarítmica para o numero de bits por pixel. Ainda, notamos que a curva real está sempre abaixo da curva teórica. Isso se deve ao fato de que o salvamento de uma imagem pode comprimir seus dados, o que diminui o tamanho do arquivo e pode afetar o fator de compressão medido.

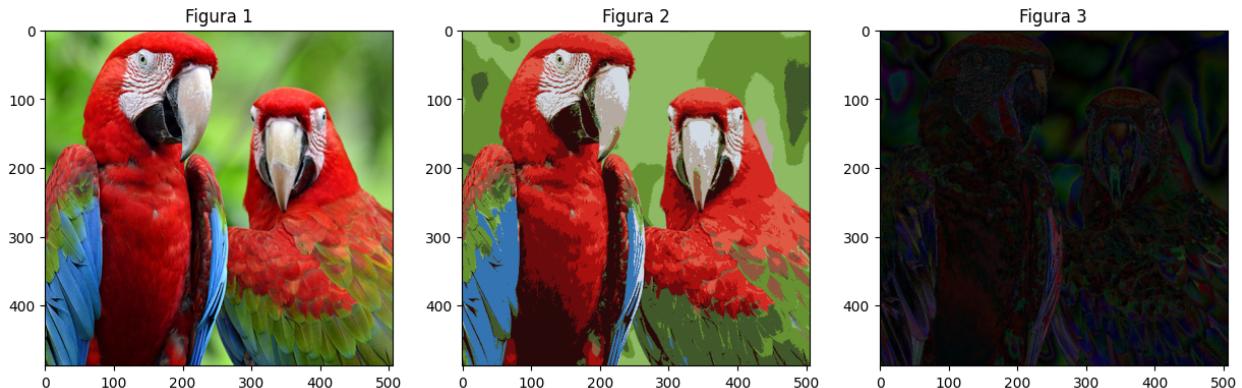
22 Exercício 4d

O gráfico obtido a partir dos valores de `model.inertia` para cada parâmetro é:



Intuitivamente, tal soma dos quadrados das distâncias trás uma noção de erro de clusterização. Quanto menor, melhor a clusterização pois os pontos estarão mais próximos dos centroides e portanto estarão formando regiões mais homogêneas (menor entropia intra-cluster). Assim, é natural buscar o número de clusters que minimiza essa distância para se obter a melhor clusterização possível. Contudo, a partir de certo ponto, o aumento do número de clusters gera pouca diminuição na métrica de erro, assim busca-se um ponto de inflexão da curva, onde, a partir deste, a diminuição do erro não é mais significativa. Assim, tomariamos um numero de clusters que divide bem a imagem em regiões homogêneas mas não é muito flexível/complexo. No caso da imagem acima, o ponto de inflexão parece estar em torno de 12 clusters.

Utilizando o código fornecido com $k = 12$ obtemos:



23 Exercício 4e

Código completo:

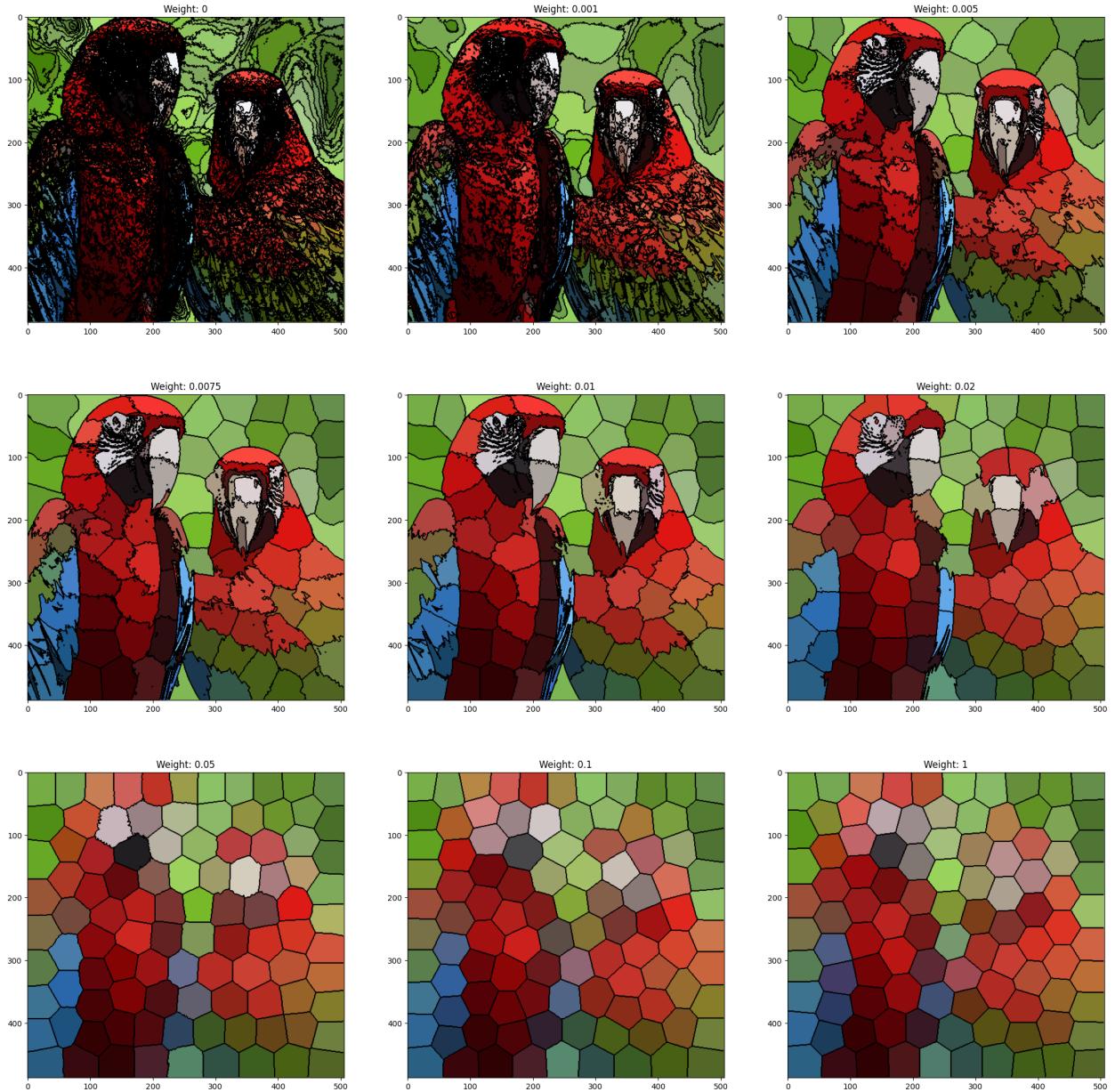
```
1 def function_3(img):
2     res = img.copy()
3
4     for i in range(1, img.shape[0] - 1):
5         for j in range(1, img.shape[1] - 1):
6             up = (img[i, j] != img[i + 1, j]).any()
7             down = (img[i, j] != img[i - 1, j]).any()
8             left = (img[i, j] != img[i, j - 1]).any()
9             right = (img[i, j] != img[i, j + 1]).any()
10
11             if up or down or left or right:
12                 res[i, j] = np.array([0, 0, 0])
13
14
15
16 imagem = mpimg.imread("araras.png")
17 output = function_1(imagem, 4)
18 edge = function_3(output)
19
20 fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(50, 50))
21
22 ax1.imshow(imagem)
23 ax1.set_title("Figura 1")
24
25 ax2.imshow(output)
26 ax2.set_title("Figura 2")
27
28 ax3.imshow(edge)
29 ax3.set_title("Figura 3")
30
31 plt.show()
```

Output gerado:



24 Exercício 4f

O output gerado pelo código é:



A inclusão das coordenadas x e y junto às intensidades das cores faz com que o modelo defina os clusteres levando em conta também essas coordenadas (elas se tornam features). O peso w é um fator que influencia o impacto da coordenada do pixel na clusterização. Quanto maior for w , maior será esse impacto. Assim, para $w = 0$, a clusterização ocorre exclusivamente com base nas cores dos pixels. Por outro lado, à medida que w aumenta, os clusters passam a formar padrões granulares na imagem devido ao impacto do uso das coordenadas como feature.

Além disso, quanto maior for w , mais uniformes em termos de área esses grãos se tornam. Isso

sugere que, à medida que w se aproxima de 1, a influência das cores dos pixels na clusterização diminui, resultando em uma segmentação homogênea baseada predominantemente na coordenada do pixel com fronteiras aproximadamente lineares e um formato poligonal.