Relatório - Projeto Final CoCADa
Emotion Recognition by EigenFaces + Logistic Regression

Moises Auad Werneck
BCC - UFRJ - 2025.2

Resumo: *No presente relatório, discorro sobre o processo da elaboração do projeto final da disciplina de Computação Científica e Análise de Dados. O projeto consiste num algoritmo que dispõe de imagens padronizadas de rostos que esboçam determinada emoção e deve reconhecer a emoção de uma imagem-alvo corretamente baseado na amostra disponível. O relatório informa sobre a escolha do dataset, o tratamento e normalização dos dados, a implementação de PCA e Logistic Regression a fins de reconhecimento, além dos resultados e visualização final.*

1. Dataset

Optei por utilizar o dataset ckextended.csv pelo formato padronizado das imagens (rostos centralizados e na mesma posição). Logo notei alguns problemas, como grandes amostras de algumas emoções: 'Neutral' com mais de 500 imagens, e pequenas amostras para outras: 'Fear' com menos de 30. Mas prossegui com esse dataset pois não encontrei outro nesse formato que acabou me agradando bastante

1.1 Limpeza e normalização.

Iniciei a normalização trocando os índices numéricos das emoções por strings mais intuitivas e ordenando as linhas do dataset pela ordem alfabética das emoções.

```
df["emotion"] = df["emotion"].replace({
    0: "Anger",
    1: "Disgust",
    2: "Fear",
    3: "Happy",
    4: "Sadness",
    5: "Surprise",
    6: "Neutral"
}).astype(str)

df = df.sort_values(by="emotion").reset_index(drop=True)
```

Depois criei o ***df_flatten*** que consistia numa cópia do dataset, mas que continha a coluna pixels em um array unidimensional. O que vai permitir que nossas funções posteriores trabalhem melhor.

```
df_flatten = df.copy()
df_flatten["pixels"] = df["pixels"].apply(lambda x: process_pixels(x).flatten())
```

E então criei as variáveis ***faces*** e ***faces_flatten*** que continham apenas os pixels (imagens) das emoções em 2D e 1D, respectivamente.

```
faces = df["pixels"].apply(lambda x: process_pixels(x).reshape(48, 48))
faces_flatten = df["pixels"].apply(lambda x: process_pixels(x).flatten())
```

Abaixo segue uma pequena amostra do dataset utilizado:

Por fim, a quantidade de amostras para cada emoção disponível:

```
  0   ->   44   =  Anger   (45)
 45   ->  103   =  Disgust   (59)
104   ->  128   =  Fear   (25)
129   ->  197   =  Happy   (69)
198   ->  790   =  Neutral   (593)
791   ->  818   =  Sadness   (28)
819   ->  900   =  Surprise   (83)
```

2. Ideia inicial.

A princípio, tive a ideia de separar as imagens em listas das diferentes emoções, calcular a direção de maior variância (autovetor associado ao maior autovalor da matriz AAt) de cada emoção e depois projetar a imagem-alvo em cada autovetor. Depois reconstruir a matriz da forma $A_1 = B * C^t$. E calcular o erro $A - A_1$. O menor erro, então, acusa a emoção correta.

2.1 Funções

Nesse propósito, criei 2 funções. A primeira:

- ***training_PCA(faces_flatten, df_flatten, target_index):*** Consiste em gerar os autovetores principais de cada emoção baseado na imagem que enviamos. A imagem é importante, já que se ela for considerada no 'treinamento', o erro será zero sempre. Dessa forma:

Um for para cada emoção, separando as imagens que iremos usar para 'treinar' o PCA.

```
for emotion in emotions:
    data = df_flatten[df_flatten["emotion"] == emotion]["pixels"]

    if data.shape[0] == 0:
        continue
```

Remove-se a imagem da base para o PCA caso ela esteja presente e atribui em ***X_train*** os pixels que serão usados para treinar.

```
target_row = target_index

if target_row in emotion_indices:
    data = data.drop(target_row)      # Evita overfitting. (Exclui a própria imagem-alvo do treinamento)

X_train = np.vstack(data).astype(np.float32) / 255.0  # Normalização simples
```

Passa o ***X_train*** para a função PCA e atribui os autovetores obtidos para aquela determinada emoção.

```
    pca = PCA(n_components= size_train).fit(X_train)
    emotion_pcas[emotion] = pca

return emotion_data, emotion_pcas
```

- ***classify_emotion(faces_flatten, emotion_pcas, target_index):*** Consiste em determinar a emoção da imagem baseada na reconstrução considerando o PCA de cada emoção. Assim:

Para os autovetores obtidos por emoção, projeta-se a imagem -alvo, reconstruímos as imagens com base nas coordenadas obtidas e calcula-se o erro.

```
# Projetar a imagem-alvo em cada PCA gerado pelo X_Train de um emoção específica
for emotion, pca in emotion_pcas.items():

    # Projetamos a imagem alvo no espaço gerado pelo PCA daquela emoção
    projection = pca.transform([target_image])

    # Reconstruir a imagem usando aquela fórmula A1 = B @ C.T
    reconstruction = pca.inverse_transform(projection).flatten()

    # Calcula erro
    error = np.linalg.norm(target_image - reconstruction) / np.linalg.norm(target_image)
```

Armazena os erros num vetor e retorna a emoção correspondente a o menor deles.

```
    all_errors[emotion] = error
    if error < min_error:
        min_error = error
        predicted_emotion = emotion

return predicted_emotion, min_error, all_errors
```

Com essas duas funções definidas, basta aplicá-las às imagens de escolha.

2.2 Execução

Ao escolher um index para identificarmos no dataset, basta executar as linhas a seguir:

```
data, pcas = training_PCA(faces_flatten, df_flatten, target_index)
detected_emotion, min_error, all_errors = classify_emotion(faces_flatten, pcas, target_index)

print(f"\nEmoção detectada: {detected_emotion} | Erro: {min_error:.4f}\n\nTodos os erros:")
for emotion, error in all_errors.items():
    print(f"Emoção: {emotion:>10} -> Erro: {(error):.4f}")
```

✓ 0.2s

As saídas, porém, não foram conforme o esperado.

```
Emoção detectada: Anger | Erro: 0.1692

Todos os erros:
Emoção:      Anger -> Erro: 0.1692
Emoção:    Disgust -> Erro: 0.2453
Emoção:       Fear -> Erro: 0.3047
Emoção:      Happy -> Erro: 0.2408
Emoção:    Neutral -> Erro: 0.2824
Emoção:    Sadness -> Erro: 0.2010
Emoção:   Surprise -> Erro: 0.2100
```

A imagem escolhida neste teste foi da emoção ‘Happy’, mas o algoritmo não detecta ‘Happy’ e erra para ‘Anger’..

2.3 Resultados

Apesar de a ideia parecer boa inicialmente, os números de precisão não foram satisfatórios. Ao executar em todas as imagens, obtêm-se os seguintes resultados:

```
Emotion:  Anger
Casos avaliados: 45 | Acertos : 5 | Erros: 40 | Acc : 11.111 % | Erros pra neutral: 6

Emotion:  Disgust
Casos avaliados: 59 | Acertos : 22 | Erros: 37 | Acc : 37.288 % | Erros pra neutral: 11

Emotion:  Fear
Casos avaliados: 25 | Acertos : 0 | Erros: 25 | Acc : 0.000 % | Erros pra neutral: 12

Emotion:  Happy
Casos avaliados: 69 | Acertos : 50 | Erros: 19 | Acc : 72.464 % | Erros pra neutral: 13

Emotion:  Neutral
Casos avaliados: 593 | Acertos : 104 | Erros: 489 | Acc : 17.538 % | Erros pra neutral: 13

Emotion:  Sadness
Casos avaliados: 28 | Acertos : 0 | Erros: 28 | Acc : 0.000 % | Erros pra neutral: 21

Emotion:  Surprise
Casos avaliados: 82 | Acertos : 75 | Erros: 7 | Acc : 91.463 % | Erros pra neutral: 23

Acertos Totais : 256 | Erros Totais: 645
Total Accuracy: 28.38 %
```

- Houve um bom acerto para ‘Surprise’, já que a boca aberta característica das imagens permite distinguí-las com facilidade.

- Emoções negativas como ‘Anger’, ‘Sadness’, ‘Fear’ tiverem acertos baixíssimos por conta das amostras pequenas e por não apresentarem tantas diferenças em termos matriciais.

E então pesquisei sobre outras formas de usar PCA (EigenFaces) para, mantendo a essência do projeto, acrescentar precisão ao algoritmo.

3. Ideia Aprimorada.

3.1. Regressão Logística

Esse método procura determinar probabilidades de eventos acontecerem baseado em variáveis aleatórias independentes. No caso desse projeto, queremos que o algoritmo determine qual é a emoção que a imagem-alvo tem maior probabilidade de esboçar.

Para isso, o método utiliza a equação:

$$p(classe = i|x) = \frac{(e^{W_i * x + b_i})}{\sum(e^{W_i * x + b_i})}$$

A regressão logística pretende estabelecer 'scores' para cada emoção. Para isso, a regressão inventa uma função linear $Z_i = W_i * x + b_i$ sendo 'i' de 0 a 6, já que há 7 emoções. Após isso, é transformado esse 'score' arbitrário em probabilidade por meio da função e^{Z_i}. E, então, essa função é normalizada, já que pretende-se encontrar as probabilidades. Dividindo, assim, pelo somatório das funções das outras emoções, criando assim uma partição entre todas as e^{Z_i}.

Conhecendo a função que mede a probabilidade, o que algoritmo quer é maximizar a seguinte função:

$$P = \prod_{k=1}^{N} p(y_k|x_k)$$

Ou seja, maximizar o produtório das probabilidades verdadeiras (achou emoção k quando a foto esboçava de fato a emoção k).

Porém essa equação resulta num número muito pequeno, então o modelo usa outra métrica. Aplicando log dos dois lados:

$$ln\, P = \sum_{k=1}^{N} ln(p(y_k|x_k)$$

O que gera o conhecido *log-verossimilhança* (log-likelihood).

Porém, o modelo do sklearn não procura maximizar funções, e sim minimizar. Logo o algoritmo busca minimizar a função:

$$L(W,b) = -\sum_{k=1}^{N} ln(p(y_k|x_k)$$

Dado um conjunto de ***W*** e ***b*** coeficientes.

A função de probabilidade já é conhecida. Então, substituindo, temos:

$$L(W,b) = -\sum_{k=1}^{N} ln(\frac{(e^{W_i*x+b_i})}{\sum(e^{W_i*x+b_i})})$$

Usando a propriedade de log de fração, a equação fica:

$$L(W,b) = -\sum_{k=1}^{N} [W_{yk} * x_k + b_{yk} - ln(\sum(e^{W_i*x+b_i}))]$$

E essa é exatamente a função que a biblioteca sklearn implementa no presente projeto.

Logo após é aplicado o método gradiente descendente que realiza as iterações abaixo a fim de achar W_i e b_i que minimizam L(W,b):

$$W_i \leftarrow W_i - \eta\frac{\partial L}{\partial W_i}$$

$$b_i \leftarrow b_i - \eta\frac{\partial L}{\partial b_i}$$

Com taxa de aprendizado η. Isso é feito até que haja estabilidade ou que o ***max_interactions*** seja atingido.

Ao descobrir W_i e b_i, é possível calcular as probabilidades e estipular a maior delas como a emoção detectada. E é exatamente isso que a segunda parte do projeto realiza.

c3.2 Execução

Na segunda parte, há as funções:

- ***training_PCA_LR(faces_flatten, df_flatten, n_components=80)***
  Repete parte do código da função da ideia inicial, mas contém as próximas linha como diferença:

Treina o modelo de Regressão Logística estabelecendo um máximo de iterações. Depois retorna o PCA (agora global) e o modelo clf. Ambos usaremos na próxima função.

```
    # Treina Logistic Regression no espaço PCA
    clf = LogisticRegression(max_iter=2000)
    clf.fit(X_proj, y)

    return pca, clf
```

- ***classify_emotion_PCA_LR(faces_flatten, pca, clf, target_index):*** Em parte, realiza a mesma coisa que a ***classify*** antiga, mas na hora de tomar decisões, muda completamente:

Normaliza a imagem-alvo (1D) e projeta no PCA global.

```
# Normaliza a imagem-alvo
target = faces_flatten[target_index].astype(np.float32) / 255.0

# Projeta no espaço PCA
target_proj = pca.transform([target])
```

Logo após o modelo prevê a emoção retratada na imagem (index 0 indica o y com maior probabilidade) além de retornar também as probabilidades das outras emoções.

```
    # Predição via logistic regression
    predicted_emotion = clf.predict(target_proj)[0]

    # Probabilidades (softmax)
    probs = clf.predict_proba(target_proj)[0]

    return predicted_emotion, probs
```

3.3 Resultados

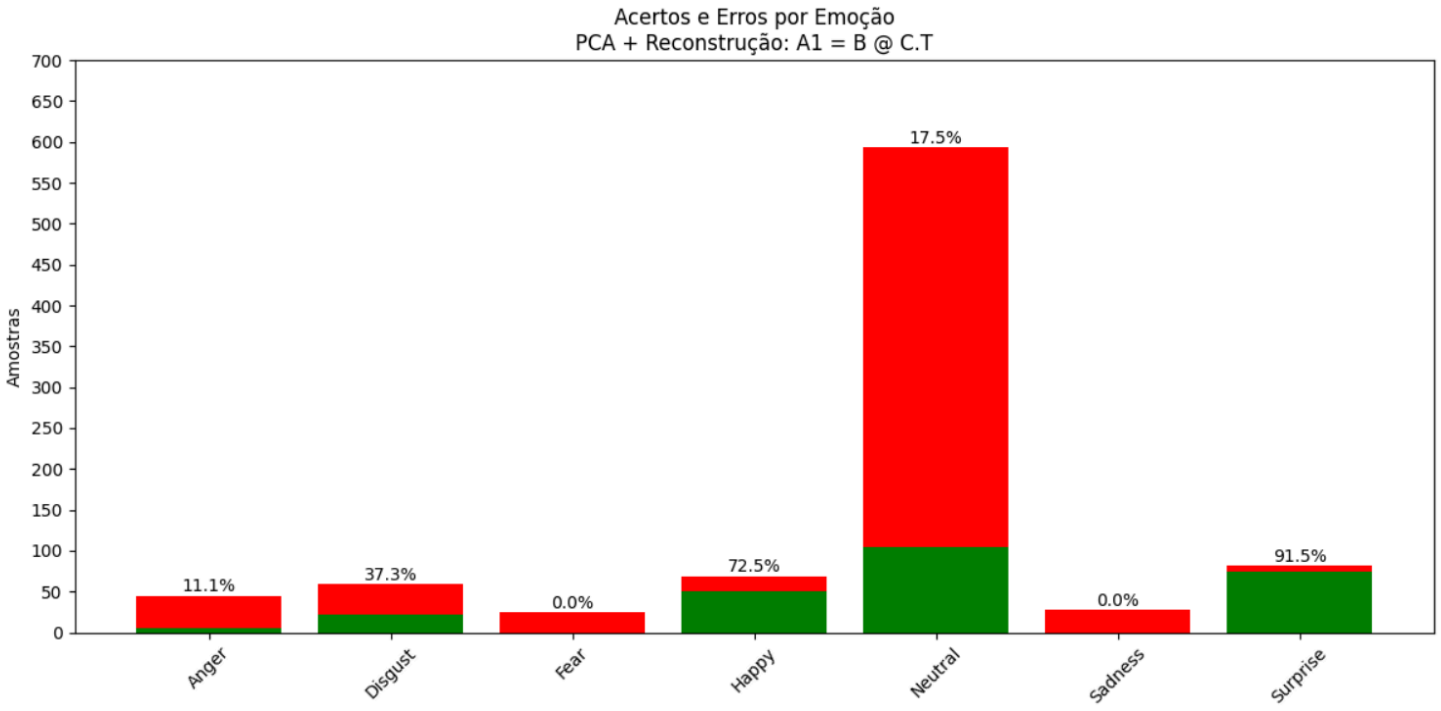
Como esperado, os resultados foram expressivamente positivos em termos de precisão. Abaixo segue o teste feito com todas as imagens do dataset:

```
Emotion:  Anger
Casos avaliados: 45 | Acertos : 38 | Erros: 7 | Acc : 84.44 % | Erros pra neutral: 7

Emotion:  Disgust
Casos avaliados: 59 | Acertos : 57 | Erros: 2 | Acc : 96.61 % | Erros pra neutral: 9

Emotion:  Fear
Casos avaliados: 25 | Acertos : 24 | Erros: 1 | Acc : 96.00 % | Erros pra neutral: 10

Emotion:  Happy
Casos avaliados: 69 | Acertos : 69 | Erros: 0 | Acc : 100.00 % | Erros pra neutral: 10

Emotion:  Neutral
Casos avaliados: 593 | Acertos : 593 | Erros: 0 | Acc : 100.00 % | Erros pra neutral: 10

Emotion:  Sadness
Casos avaliados: 28 | Acertos : 22 | Erros: 6 | Acc : 78.57 % | Erros pra neutral: 16

Emotion:  Surprise
Casos avaliados: 82 | Acertos : 81 | Erros: 1 | Acc : 98.78 % | Erros pra neutral: 17

Acertos Totais : 884 | Erros Totais: 17
Total Accuracy: 98.00 %
```

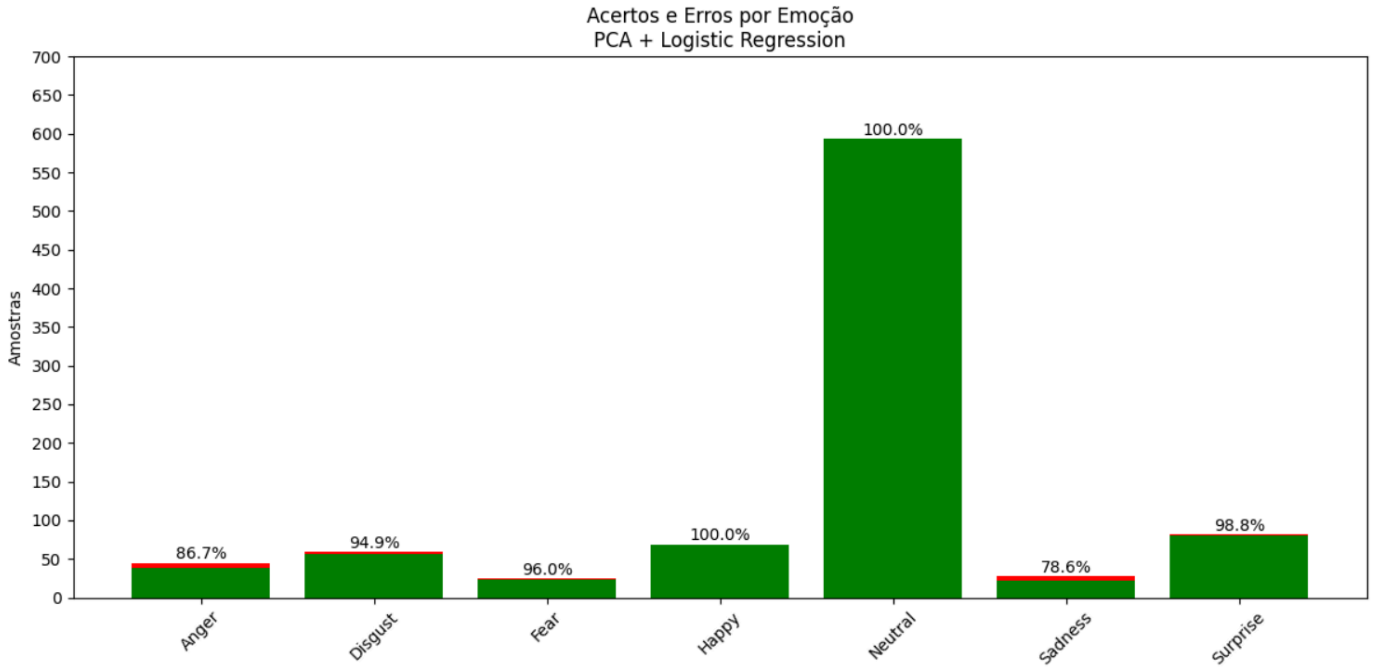
Um detalhe importante é que todos os erros foram pra ‘Neutral’. Ou seja, o modelo só se confunde com ‘Neutral’, talvez por conta do ‘Overfitting’ causado pela quantidade bem superior de amostras que a emoção ‘Neutral’ possui.

4. Visualização

Abaixo seguem gráficos que demonstram o desempenho dos dois esquemas montados a fim de reconhecer emoções:



Perceptivelmente, o modelo inicial baseado em PCA + Reconstrução não funcionou bem, exceto pela emoção ‘Surprise’ (boca aberta característica possivelmente ajudou) e ‘Happy’ por motivos desconhecidos.



Já a implementação de Regressão Logística conseguiu ser absolutamente superior à primeira versão de reconhecimento.

5. Considerações Finais

Esse documento tem prazo para ser entregue dia (10/12/2025) e pode ser que na apresentação de sexta (12) o código tenha sofrido algumas alterações e a visualização tenha sido incrementada com alguns insights, mas nada que remova o cerne do projeto.