# Homework I
# ATFMS

Arianna Abis 303876
Riccardo Kiefer 301286

14 maggio 2023

## 1 Descrizione del codice

L'obiettivo dell'homework è quello di implementare un algoritmo in python per il riconoscimento automatico dei volti basato sul paper *Face Recognition Using Eigenfaces* del 1991 pubblicato da Matthew Turk *(Computer Science Department, Università della California)* e da Alex Pentland *(Mit Media Laborator)*. Viene eseguita un'analisi delle componenti principali (PCA) sulle immagini del volto per estrarre le cosiddette "eigenfaces". Le eigenfaces rappresentano le direzioni principali della variazione nei dati di addestramento dei volti. L'eigenfaces recognition sfrutta l'idea che le immagini di volti condividano alcune caratteristiche comuni, rappresentate dalle eigenfaces. Questo approccio può essere utilizzato per il riconoscimento facciale, la verifica dell'identità o altre applicazioni che richiedono il confronto di volti. Segue nei dettagli la nostra implementazione per la Face Recognition:

1. **Preprocessing e split in training e test set:** per prima cosa definiamo il numero di immagini per ciascun soggetto da inserire nel training e nel test set. Chiamiamo $L$ il numero totale di immagini di training e $T$ il numero totale di immagini di test. Per eseguire lo split viene fatto un loop che accede alla sottocartelle $s1 \dots s40$ della cartella principale *archive*, e per ciascuna delle sottocartelle assegna ciascuna immagine del soggetto al training o al test secondo i valori decisi in fase iniziale. Le immagini vengono inserite nella matrice di training o di test dopo essere state trasformate in vettori. In particolare la dimensione di ogni immagine è $92 * 112$ pixel, per cui la

matrice di training avrà numero di righe pari a $L$ e numero di colonne pari a $10304 = 92 * 112$. Si procede in maniera analoga per la matrice dei test data.

```
# path del dataset
dataset_path = "/Users/riccardokiefer/Desktop/archive/"

# Definiamo il numero di soggetti e per ciascun soggetto quante immagini mettiamo nel training set e quante nel test set
num_subjects = 40
num_train_images = 5 #numero di immagini usate come training (modificabile)
num_test_images = 5
L = num_subjects * num_train_images

# Creiamo due array vuote dove conservare i dati di training e di test
train_images = np.empty((num_subjects * num_train_images, 92 * 112))
test_images = np.empty((num_subjects * num_test_images, 92 * 112))

# Loop nelle sottocartelle di ciasciun soggeto
for i in range(1, num_subjects + 1):  # valori da 1 a 40 per propriet  di range
    for j in range(1, num_train_images + 1):  # va da 1 a 6
        # Definizione del path dell'immagine correnre
        image_path = os.path.join(dataset_path, "s" + str(i), str(j) + ".pgm")
        # lettura dell'immagine
        img = np.array(plt.imread(image_path), dtype='uint8')
        # immagine trasformata in un vettore in R^(m*n)
        img = img.flatten()
        # immagine conservata nell'array train images (per righe)
        train_images[(i - 1) * num_train_images + (j - 1)] = img

    for j in range(num_train_images + 1, num_train_images + num_test_images + 1):
        image_path = os.path.join(dataset_path, "s" + str(i), str(j) + ".pgm")
        img = np.array(plt.imread(image_path), dtype='uint8'
```

```
    )
        img = img.flatten()
        test_images[(i - 1) * num_test_images + (j - 1 -
    num_train_images)] = img
```

2. Viene calcolata l'immagine media sui dati di training facendo la media per colonne della matrice *train_images*.

```
# b) Calcolo della faccia media sulle immagini di training

mean_face = np.mean(train_images, axis=0)  #

# visualizzazione faccia media
mean_face_img = mean_face.reshape(112, 92)

plt.imshow(mean_face_img, cmap='gray')
plt.title('Mean face')
```

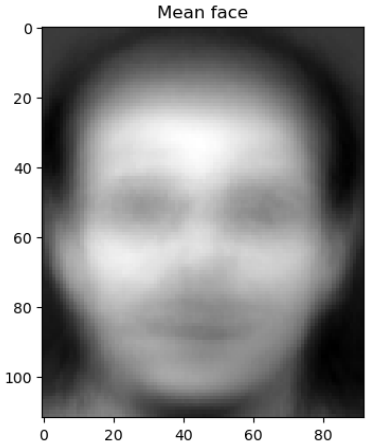


Figura 1: Faccia media

3. Tutte le immagini vengono centrate tramite sottrazione del vettore *mean_face* alla matrice *train_images*. La matrice che ha sulle righe i vettori corrispondenti alle immagini centrate viene denominata *phi*.

```

# c) Sottrazione della media dalle immagini del training set

phi = train_images - mean_face  # dim phi = 240*10304 ->
    ogni riga corrisponde ad un immagine

```

4. Viene calcolata la covarianza della matrice *phi*, ottenendo una matrice 10304∗ 10304.

```

C = 1 / L * np.dot(phi.T, phi)  #matrice di covarianza


```

5. Calcolando gli autovettori e autovalori della matrice di covarianza si ottengono le cosiddette componenti principali, ovvero gli autovettori i cui autovalori sommati spiegano almeno il 95% della varianza. Indichiamo con *n* l'indice dell'ultimo autovettore preso in considerazione (gli autovettori sono ordinati in ordine decrescente in base al valore dell'autovettore corrispondente). Questo permette di ridurre la dimensionalità del problema e lavorare su uno spazio di dimensione minore, in cui vengono proiettate le immagini.

```

# e) Scegliamo un valore sopra il quale non considerare pi
    gli autovalori di relativo indice
# \\\Per farlo creiamo una soglia che preservi il 95% della
    varianza totale.

# Calcoliamo la varianza totale come somma degli autovalori
total_variance = np.sum(eigenvalues)

# Calcoliamo la varianza spiegata da ogni autovalore
variance_explained = eigenvalues / total_variance

# Calcoliamo la varianza cumulata spiegata (ovvero come
    cambia la percentuale di varianza spiegata aggiungendo
    di volta in volta un autovalore)
cumulative_variance = np.cumsum(variance_explained)
```

```
13
14 # Decidiamo la soglia e cerchiamo l'indice n pi  alto nel
      vettore cumulative variance, ovvero la somma degli
      autovalori
15 # di indici nel range(n+1) spiega il 95% della varianza
16 threshold_var = 0.95
17 idx = np.argmax(cumulative_variance >= threshold_var)
18
19 # Manteniamo gli autovettori i cui autovalori corrispondenti
       hanno valore maggiore
20 eigenvalues = eigenvalues[:idx + 1]
21 eigenvectors = eigenvectors[:, :idx + 1]
22
23
```

6. Creazione della matrice $w$ contenente per ogni immagine di training le $n$ componenti rispetto alle $n$ componenti principali.

```
1
2 # f)proiezione delle facce nel database iniziale
3 w = np.dot(phi, eigenvectors)
4
5
6
7
```

7. **Fase di test su un unico soggetto:** a questo punto l'algoritmo viene testato su una sola immagine di test, che viene correttamente riconosciuta. L'immagine, come vettore di 10304 elementi, viene proiettata nello spazio delle autofacce (dopo essere stata centrata) e poi viene calcolata la distanza euclidea tra la proiezione del suddetto vettore e la proiezione di ciascuna delle immagini di training. A questo punto si sceglie il valore minimo di distanza, si controlla che sia sotto la soglia di classificazione (che viene scelta in questo caso arbitrariamente alta sulla base del vettore di distanze), e si procede alla classificazione del soggetto. Per facilitare la lettura dei risultati viene creato un dizionario che ha come chiavi i numeri interi da 1 a 40, che indicano i soggetti del dataset, e come valori per ciascuna chiave gli indici riga della matrice di training che corrispondono alle immagini del soggetto. Inoltre viene definita la funzione $get\_keys\_from\_val$ che dato in input un valore tra 1 e $L$, restituisce il numero del soggetto a cui l'immagine appartiene.

```
1
2  # b)prendiamo una nuova immagine f_new (gi  vettorizzata)
3
4  f_new = test_images[0, :]
5  f_new_image = f_new.reshape(112, 92)
6  plt.imshow(f_new_image, cmap='gray')
7
8  # c)sottrarre la media da f_new
9
10 phi_new = f_new - mean_face
11
12 # d) proiettiamo phi_new nello spazio delle autofacce
13
14 w_new = np.dot(phi_new, eigenvectors)
15 #%%
16 # e)calcoliamo la minima distanza euclidea eps tra la
       proiezione di phi_new e le proiezioni del training set
17
18 d = np.zeros(len(train_images))
19 for i in range(len(train_images)):
20     d[i] = np.linalg.norm(w_new - w[i, :])
21
22 eps = np.min(d)
23 #%%
24 # f)controlliamo che eps sia minore della soglia per la
       classificazione e classifichiamo il soggetto
25
26 if eps <= threshold1:
27     j = np.argmin(eps)
28     print("L'immagine pi  vicina alla nuova immagine
       quella numero:", format(j))
29 else:
30     print("L'immagine non   nel dataset")
31
32 # Creiamo un dizionario che associa ad ogni soggetto le
       immagini corrispondenti sia nel training che nel test
33 dict_train = {}
34 for i in range(num_subjects):
35     dict_train[i + 1] = np.array(range(i * num_train_images,
        (i + 1) * num_train_images))
36 dict_test = {}
37 for i in range(num_subjects):
38     dict_test[i + 1] = np.array(range(i * num_test_images, (
       i + 1) * num_test_images))
39
```

```
# funzione che dato il numero di un'immagine restituisce il soggetto a cui l'immagine appartiene
def get_keys_from_value(d, val):
    return [k for k, v in d.items() if (val in v)]

# si applica la funzione per classificare un soggetto specifico
subject = get_keys_from_value(dict_train, j)

# viene stampato il risultato
print('La faccia appartiene al soggetto {}'.format(str(subject)[1:-1]))


```

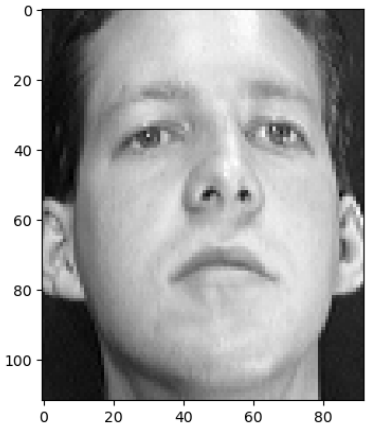


Figura 2: Faccia del soggetto riconosciuto

8. **Fase di test utilizzando tutto il test set**: eseguiamo quanto descritto al punto precedente per tutte le immagini del test set (otterremo quindi una matrice di proiezioni di dimensione $T * n$ e una matrice di distanze $L * T$, in quanto per ogni proiezione di un immagine di test dobbiamo calcolare la distanza tra questa e ciascuna proiezione del training, in ogni riga della matrice di distanza viene poi selezionato il valore minimo e l'indice corrispondente.). All'interno della funzione che conta gli individui correttamente classificati, abbiamo salvato in una cartella tutte le immagini di soggetti che non vengono riconosciuti o che non vengono classificati correttamente, allo scopo di individuare delle eventuali cause come bassa qualità della foto/viso non centrato. Tuttavia la ricerca non ha prodotto risultati interpretabili.

```
1
2  # Applichiamo l'algoritmo a tutto il test set:
3
4  # Si centrano i dati togliendo ad ogni valore della colonna
       la media della colonna stessa
5  phi_test = test_images - mean_face
6
7  # Vettore proiezione delle immagini di test
8  w_test = np.dot(phi_test, eigenvectors)
9
10 # Calcolo delle distanze per ogni vettore del traning set
11 distances = np.zeros((len(test_images), len(train_images)))
12
13 for i in range(len(test_images)):
14     for j in range(len(train_images)):
15         distances[i, j] = np.linalg.norm(w_test[i, :] - w[j,
       :])
16
17 # Indici delle distanzi minori
18 min_indices = np.argmin(distances, axis=1)
19 #%%
20 # Vettore delle varie soglie scelte
21 threshold_array = np.array([10000, 3000, 800])
22
23 accuracy = np.zeros(len(threshold_array))
24 n_recognized = np.zeros(len(threshold_array))
25 n_unrecognized = np.zeros(len(threshold_array))
26
27 # Funzione che restituisce il numero di classificazioni
       corrette e salva le immagini che non soddisfano i
       requisiti di
28 # classificazione (condizioni all'interno dell'if) ovvero
       immagini troppo distanti dallo spazio della autofacce o
       che vemgono
29 # #attribuite ad un soggeto sbagliato
30 def corrected_identified(distances, threshold):
31     correctly_identified = 0 # Contatore delle immagini
       correttamente classificate
32     unsatisfied_images = []  # Lista delle immagini che non
       soddisfano i requisiti
33     save_directory = "/Users/riccardokiefer/Desktop/
       Unclassified/"
34
35     for i in range(len(test_images)):
36         if np.min(distances[i]) <= threshold and
```

```
    get_keys_from_value(dict_train, min_indices[i]) ==
    get_keys_from_value(dict_test, i):
37              correctly_identified = correctly_identified + 1
38          else:
39              unsatisfied_images.append(test_images[i])
40              save_directory_i = os.path.join(save_directory,
    str(threshold))
41              os.makedirs(save_directory_i, exist_ok=True)
42              image_path = os.path.join(save_directory_i, f"
    unsatisfied_image_{i}.png")
43              image = Image.fromarray(test_images[i].reshape
    (112, 92))
44              image = image.convert('L')  # Converte in scala
    di grigio
45              image.save(image_path)
46
47      return correctly_identified
48
49  correctly_identified = np.zeros(len(threshold_array))
50
51  # Loop che calcola la bont  di classificazione al variare
    del valore di soglia
52  for i in range(len(threshold_array)):
53      threshold = threshold_array[i]
54      recognized = distances[np.arange(len(test_images)),
    min_indices] <= threshold
55      n_recognized[i] = np.sum(recognized)
56      n_unrecognized[i] = len(test_images) - n_recognized[i]
57      correctly_identified[i] = corrected_identified(distances
    , threshold)
58
59      accuracy[i] = correctly_identified[i]/len(test_images)
60
61  # Stampa dei risultati
62  print('Test thresholds:', threshold_array)
63  print('Number of people correctly identified in each case:',
      correctly_identified)
64  print('Number of recognized test images in each case:',
    n_recognized)
65  print('Number of unrecognized test images in each case:',
    n_unrecognized)
66  print('Accuracy in each case:', accuracy)
67
68  # Barplot dei valori di accuracy al variare della soglia
69  plt.figure(figsize=(8, 6))
```

```
bars = plt.bar(threshold_array, accuracy, width=400)
plt.xlabel('Threshold')
plt.ylabel('Accuracy')
plt.title('Accuracy at Different Thresholds')

# Aggiunge l'altezza del barplot al grafico (valore accuratezza)
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height, f'{height:.2f}', ha='center', va='bottom')

# Aggiunge il valore di soglia sul quale ogni istogramma centrato
for bar, threshold in zip(bars, threshold_array):
    x = bar.get_x() + bar.get_width() / 2
    y = bar.get_height()
    plt.text(x, -0.052, f'{threshold:.0f}', ha='center', va='bottom', color='red')

plt.ylim(0, 1)  # Limiti verticali del grafico

# Salva gli istogrammi
plt.savefig('bar_plot_test5.png')
plt.show()
```

## 1.a Results

Per testare i risultati ottenuti lo script è stato eseguito più volte al variare del numero di immagini per ciascun soggetto inserite nel training set. In particolare inizialmente, come suggerito dal testo dell'homework, abbiamo inserito nel training set le prime sei immagini di ogni soggetto e le restanti quattro nel test set. Successivamente abbiamo testato tutte le seguenti combinazioni training-test: 8-2 (quella che viene normalmente suggerita in letteratura), 7-3, 9-1, 5-5. Inoltre per ciascuna delle diverse divisioni in training e test abbiamo considerato tre possibili soglie di classificazione. Con soglia di classificazione intendiamo il valore tale che, se la distanza minima tra il vettore delle proiezioni dell'immagine test e i vettori proiezione delle immagini del training è sopra questa soglia, l'algoritmo conclude che l'immagine di test non appartiene a nessun soggetto del dataset. In

particolare abbiamo considerato tre diverse soglie: 10000, 3000 e 800. Poiché sapevamo che tutte le immagini di test appartenevano ad un soggetto le cui immagini erano state inserite nel training, non si è posto il problema dell'overfitting, ovvero di scegliere una soglia che garantisse un trade-off tra falsi positivi (sconosciuto classificato come conosciuto) e falsi negativi (conosciuto classificato come sconosciuto). Quindi la scelta delle soglie è stata semplicemente dettata dall'osservazione della matrice delle distanze tra le proiezioni delle immagini test e le proiezioni delle immagini di training, e sono stati scelti dei valori che ci permettessero di vedere i cambiamenti nei risultati per soglie troppo basse o troppo alte. Ovviamente i risultati migliori sono stati ottenuti per la soglia di classificazione più alta. Per valutare i risultati ottenuti sono state scelte le seguenti metriche:

- *n_recognized*: il numero di soggetti riconosciuti come appartenenti al dataset.
- $correctly\_classified$: numero di soggetti riconosciuti che sono stati anche correttamente classificati.
- *accuracy*: $\frac{correctly\_classified}{numero\ immagini\ di\ test}$

Tabella per i valori dell'accuratezza al variare della proporzione tra dati di training e dati di test, e al variare della soglia per la classificazione:

| Num immagini training - Num immagini test | soglia 10000 | soglia 3000 | soglia 1000 |
|---|---|---|---|
| 9-1 | 0.925 | 0.85 | 0.025 |
| 8-2 | 0.95 | 0.875 | 0.125 |
| 7-3 | 0.95 | 0.892 | 0.017 |
| 6-4 | 0.944 | 0.869 | 0.006 |
| 5-5 | 0.88 | 0.815 | 0.01 |

Tabella che contiene il numero di immagini riconosciute come appartenenti al dataset al variare della proporzione tra dati di training e dati di test, e al variare della soglia per la classificazione:

| Num immagini training - Num immagini test | soglia 10000 | soglia 3000 | soglia 1000 |
|---|---|---|---|
| 9-1 | 40 | 36 | 1 |
| 8-2 | 80 | 72 | 1 |
| 7-3 | 120 | 110 | 2 |
| 6-4 | 160 | 142 | 1 |
| 5-5 | 200 | 170 | 2 |

Tabella per i valori dei soggetti correttamente classificati al variare della proporzione tra dati di training e dati di test, e al variare della soglia per la classificazione:

| Num immagini training - Num immagini test | soglia 10000 | soglia 3000 | soglia 1000 |
|---|---|---|---|
| 9-1 | 37 | 34 | 1 |
| 8-2 | 76 | 70 | 1 |
| 7-3 | 114 | 107 | 2 |
| 6-4 | 151 | 139 | 1 |
| 5-5 | 176 | 163 | 2 |

Tabella per i valori di runtime:

| Num immagini training - Num immagini test | runtime |
|---|---|
| 9-1 | 270.8s |
| 8-2 | 269.8s |
| 7-3 | 275.4s |
| 6-4 | 383.4s |
| 5-5 | 284.1s |

Per permettere di confrontare i risultati in maniera più immediata, abbiamo creato dei barplot che riportiamo di seguito.
Come si è già notato i risultati migliori si ottengono con la soglia più alta e con uno split training-test pari a 7-3 o 8-2, quest'ultima però garantisce un tempo di esecuzione leggermente minore.

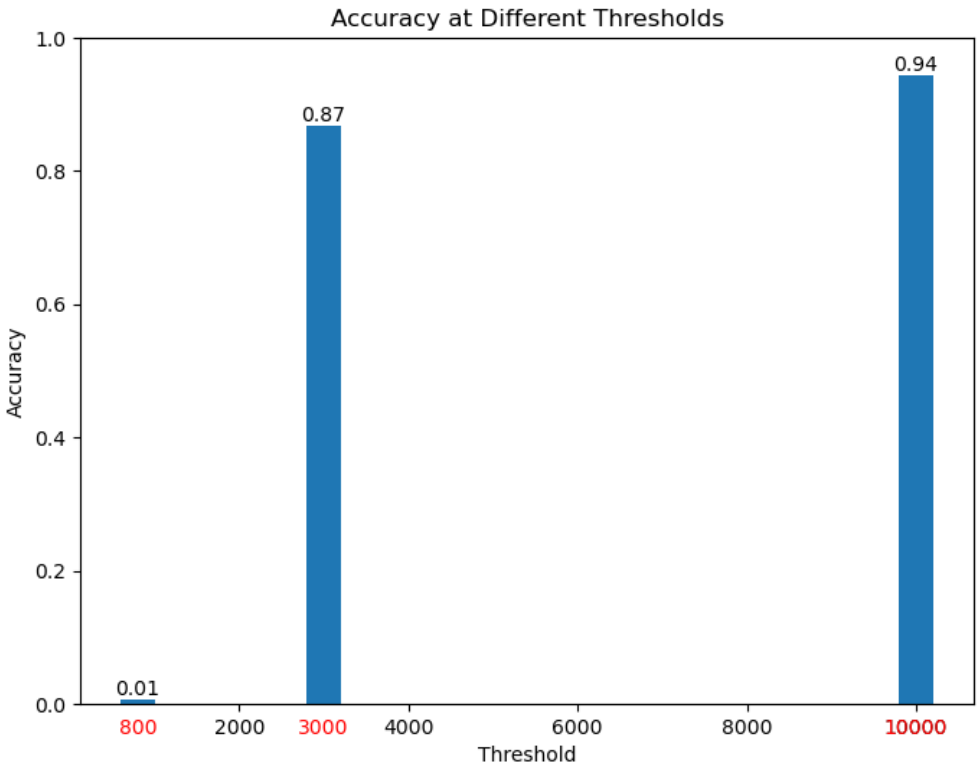


Figura 3: Barplot con diversi valori di accuratezza per soglie [10000, 3000, 800] e divisione 6-4

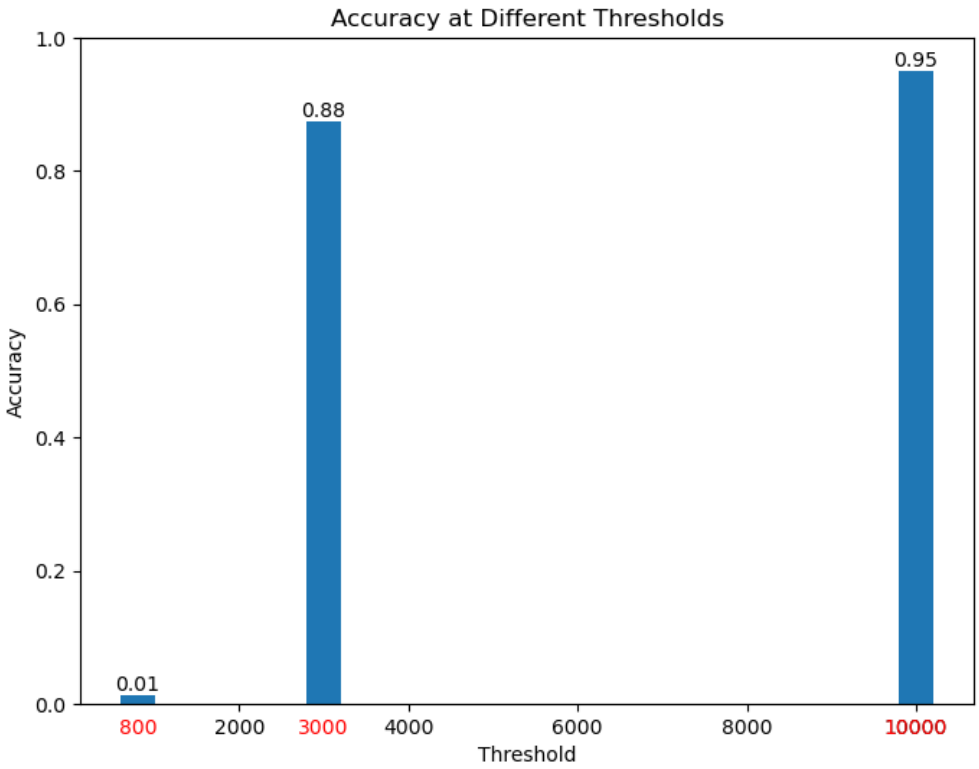


Figura 4: Barplot con diversi valori di accuratezza per soglie [10000, 3000, 800] e divisione 8-2

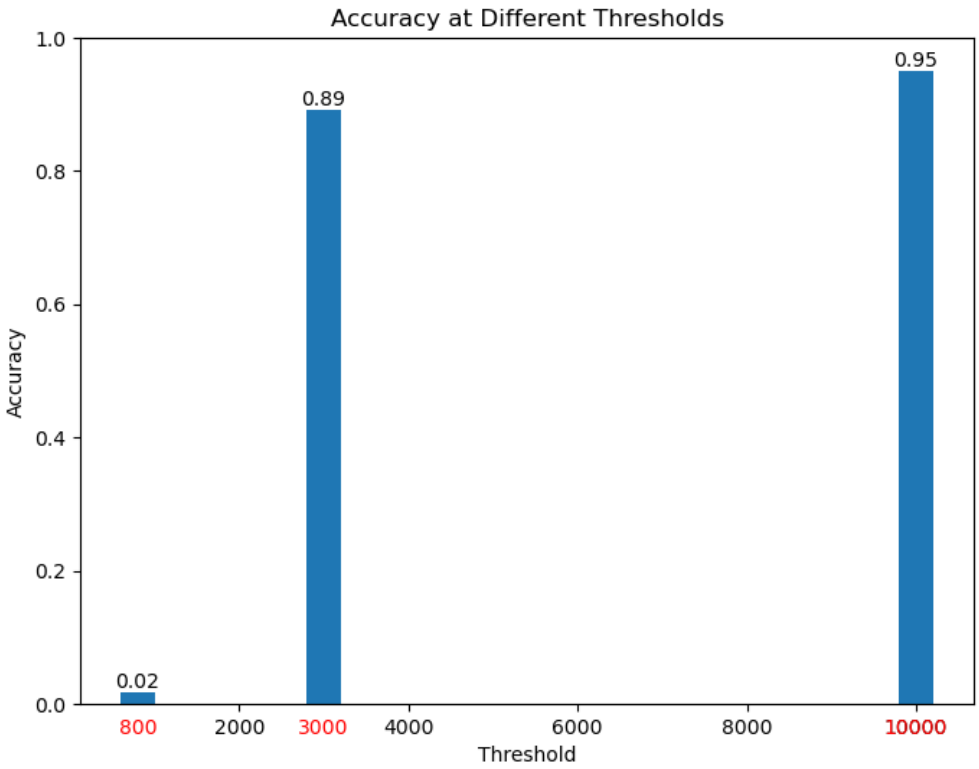


Figura 5: Barplot con diversi valori di accuratezza per soglie [10000, 3000, 800] e divisione 7-3

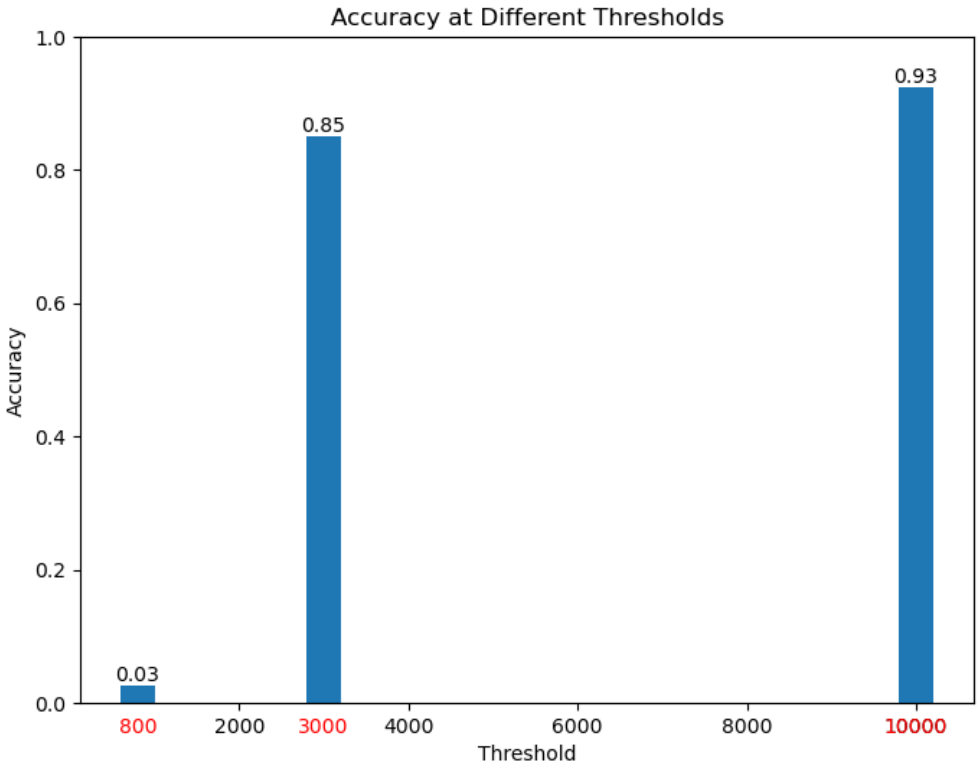


Figura 6: Barplot con diversi valori di accuratezza per soglie [10000, 3000, 800] e divisione 9-1

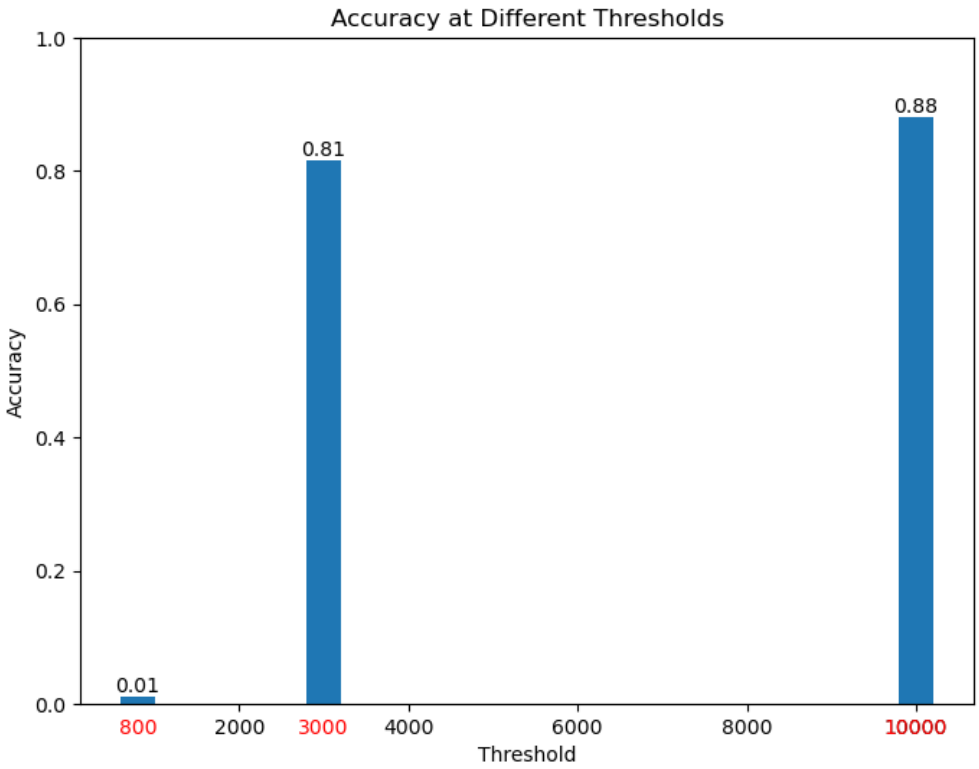


Figura 7: Barplot con diversi valori di accuratezza per soglie [10000, 3000, 800] e divisione 5-5