Human Recognition project report

Implementation of a logistic regression algorithm and application of it to fingerprint dataset

Prepared by:

Tetiana Bakai

2019

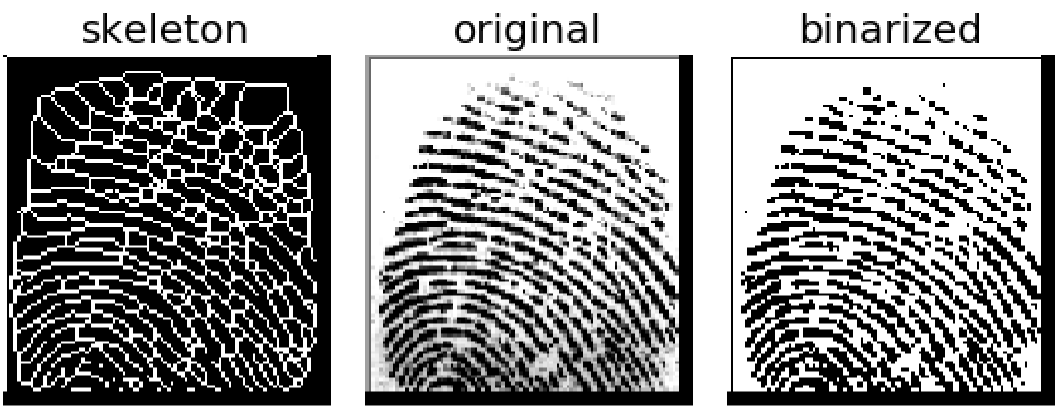
Sokoto Coventry Fingerprint Dataset (SOCOFing) was used for the project. The program was created in Spyder(Python 3.6) editor. The following Python libraries were used for the implementation of logistic regression and preprocessing of images:

```
import cv2, re, csv
from numpy import array
import glob
from IPython.display import display
from sklearn.linear_model import LogisticRegression
from skimage.morphology import skeletonize
from skimage.feature import ORB, match_descriptors, plot_matches
```

The main steps of the project are:

1. Results of preprocessing on the input images (e.g. thinning, binarization):

```
def thin_features(data):
    skeleton=[]
    for i in range(data.shape[2]):
       #skeleton.append(skeletonize((data[:,:,i] > 127).astype(np.int_)))
        #bw = images[i].point(lambda x: 0 if x<128 else 255, '1')
               skeleton = skeletonize(((data[:,:,i] > 127).astype(np.int_)))
```



2. Main features extraction:

```
descriptor_extractor = ORB(n_keypoints=30)
descriptor_extractor.detect_and_extract(skeleton)
responses = descriptor_extractor.responses
return ["{0:0.2f}".format(i) for i in responses.tolist()]
```

3. The csv file was created that contains subject number, gender and 30 main features in each row:

```python
with open("C:\\Users\\Tanya\\Downloads\\socofing\\image_features.csv", 'w',
newline='') as myfile:
    d_writer = csv.writer(myfile, quoting=csv.QUOTE_MINIMAL)
    for file in os.listdir(file_path):
        label = re.search("__(\w)_", file).group(1)
        name = re.search("^(\d+)__", file).group(1)
        dt = thin_features(data)
        d_writer.writerow([name]+[label] + dt)
                   print(file)
```

```
100,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
100,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
100,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
100,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
100,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
100,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
100,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
100,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
100,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
100,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
101,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
101,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
101,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
101,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
101,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
101,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
101,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
101,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
101,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
101,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
102,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
102,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
102,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
102,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
102,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
102,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
102,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
102,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
102,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
102,M,9.22,8.84,8.16,8.09,7.83,7.80,7.73,7.53,7.50,7.39,7.14,5.98,5.25,5.15,4.88,4.73,4.48,4.39,4.34,4.29,4.05,3.99,3.61,3.52,3.49,3.38,3.28,3.13,2.94,2.88
```

4. Splitting the dataset into train (80%) and validation part (20%):

```python
def train_test_split(X, Y, split=0.2):
    indices = np.random.permutation(X.shape[0])
    split = int(split * X.shape[0])

    train_indices = indices[split:]
    test_indices = indices[:split]

    x_train, x_test = X[train_indices], X[test_indices]
    y_train, y_test = Y[train_indices], Y[test_indices]

     return x_train, y_train, x_test, y_test
```

5. Logistic regression was implemented by using `sklearn library`:

```python
logisticRegr = LogisticRegression()
```

6. Training the model on training dataset:

```python
logisticRegr.fit(x_train, y_train)
```

7. Evaluating the model on validation dataset and calculating the accuracy on validation dataset:

```python
logisticRegr.predict(x_test)
score = logisticRegr.score(x_test, y_test)
print(score)
```

In this project a model was implemented and trained which recognizes a gender based on an image of a fingerprint. The accuracy of 80% was achieved on validation dataset. When it comes to the recognition of the subject based on an image of a fingerprint, the accuracy is almost equal to 0%. It means that logistic regression model is not suitable for this task at all.

When I run the code that contains the Logistic Regression class, the accuracy is equal to 79% :

```
class LogisticRegression:

    def __init__(self, lr=0.01, num_iter=1000, fit_intercept=True, verbose=False):
        self.lr = lr
        self.num_iter = num_iter
        self.fit_intercept =  fit_intercept
        self.verbose = verbose

    def __add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)

    def __sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def __cross_entropy(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        #weights initialization
        self.theta = np.zeros(X.shape[1])
        self.theta = self.theta.reshape(self.theta.shape[0],-1)
        for i in range(self.num_iter):
            z = np.dot(X, self.theta)
            z = z.reshape(z.shape[0],-1)
           # print(z.shape)
            h = self.__sigmoid(z)
            gradient = np.dot(X.T, (h - y)) / y.size
            #print(gradient.shape)
            self.theta -= self.lr * gradient


            if(self.verbose == True and i % 10000 == 0):
                z = np.dot(X, self.theta)
                h = self.__sigmoid(z)
                print (f'loss: {self.__cross_entropy(h, y)} \t')

    def predict_prob(self, X):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        return self.__sigmoid(np.dot(X, self.theta))


    def predict(self, X, threshold):
        return self.predict_prob(X) >= threshold
```