Adaline algorithm

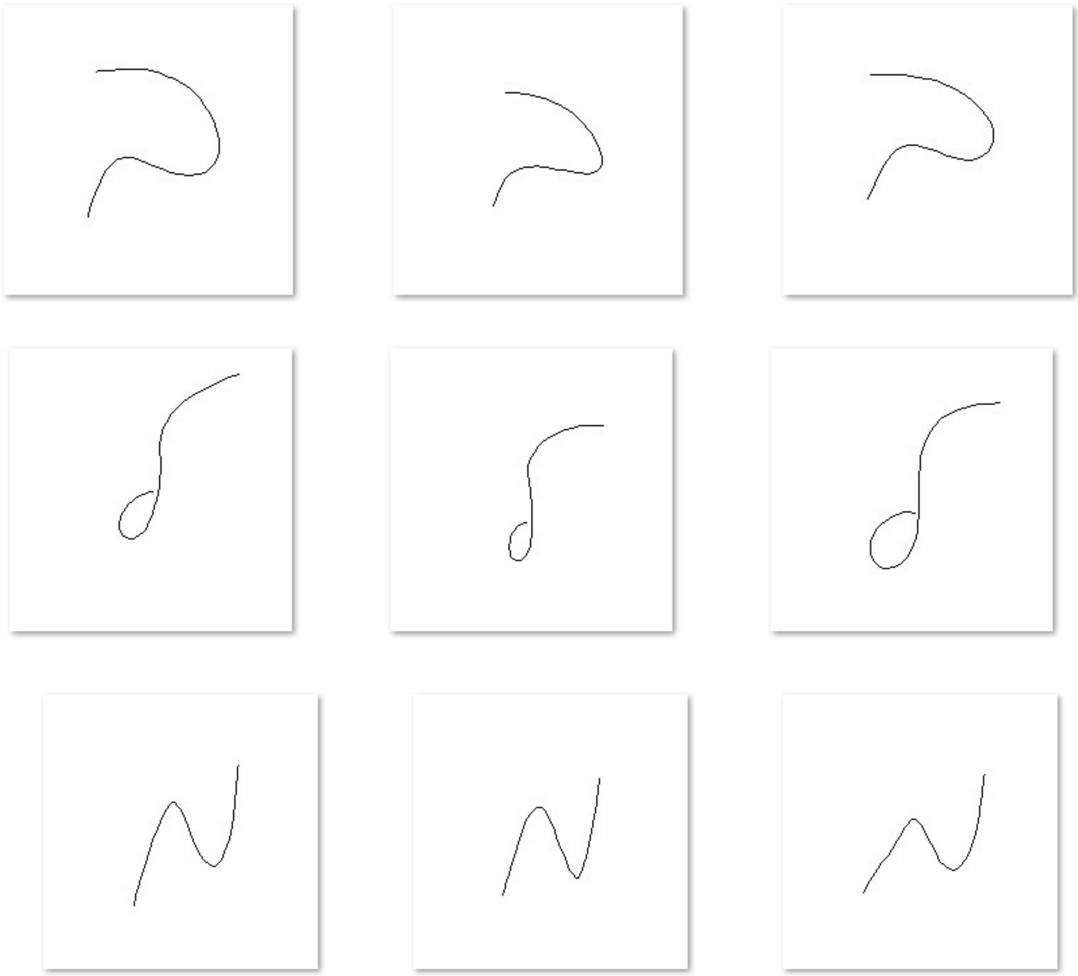
1. submitters names:

Abigail Isaacs - 212457535

Ye'ela Sitron - 209914050

Talor Langnes - 204240477

Ohad Wolfman – 316552496



2. <u>introduction:</u>

In this assignment we implemented the Adaline algorithm by identifying 3 letters from the Hebrew alphabet: B, L, M.
Each of the students is required to write 3 times each of these following letters by hand, and with a code to convert the image into a representative matrix of 10x10 and actually into a row vector with 100 elements, where each "on" pixel is converted to 1, and each "off" pixel is converted to -1.
In addition to making the dataset more complex the code generated vectors that represent the same 9 images but with 15 degrees rotate to left and to right.
The final vector contains 101 terms so the first term is the target, and is represented by 1/2/3. 1 represents the letter B, 2 represents the letter L, and 3 the letter M.
We downloaded the files of all the students, and wrote a script that goes through all the files and combines them into one text file:

```python
import os

# Specify the directory path containing the files
directory1 = 'קבצים (מ,ב,ל)'
directory2 = 'קבצים בהטייה של 15_ לשני הצדדים (מ,ב,ל)'

# Output file path
output_file1 = 'Data_regular_writing.txt'
output_file2 = 'Data_15%_rotated_writing.txt'

# Initialize the merged content
merged_content = ''

# Iterate over each file in the directory
for filename in os.listdir(directory2):
    file_path = os.path.join(directory2, filename)
    if os.path.isfile(file_path):
        with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
            content = file.read().strip()
            merged_content += content + '\n'

with open(output_file2, 'w', encoding='utf-8') as file:
    file.write(merged_content)

print("Merged content has been written to the output file.")
```

3. <u>success rates</u>
    a. comparison of ב vs ל
        1) accuracies in the cross-validation:
           0.87949,
           0.88751,
           0.88888,
           0.88063,
           0.88288
        2) Average accuracy across all folds: 0.8838
        3) Std accuracy scores: 0.003716
        4) Test accuracy score: 0.84946

    b. comparison of ב vs מ
        1) accuracies in the cross-validation:
           0.82596,
           0.83878,
           0.83759,
           0.83069,
           0.80090
        2) Average accuracy across all folds: 0.82679
        3) Std accuracy scores: 0.01375
        4) Test accuracy score: 0.85251

    c. comparison of ל vs מ
        1) accuracies in the cross-validation:
           0.82596,
           0.83878,
           0.83878,
           0.83069,
           0.80090
        2) Average accuracy across all folds: 0.82679
        3) Std accuracy scores: 0.01375
        4) Test accuracy score: 0.85251

4. Code Description:

1) Getting the files names

```python
print(" -----loading the files------- ")
# folder1_path = 'קבצים (מ,ב,ל)'
file1 = 'Data_regular_writing.txt'
# file_names1 = os.listdir(folder1_path)
print("first folder loaded")

# folder2_path = 'קבצים בהטייה של 15_ לשני הצדדים (מ,ב,ל)'
file2 = 'Data_15%_rotated_writing.txt'
# file_names2 = os.listdir(folder2_path)
print("second folder loaded")
```

2) This function accepts a file name and returns a Data Frame

```python
print("\n -----Convert the file's content into DataFrame------- ")
def read_and_create_df(fila_name):
    with open(fila_name, 'r', encoding='latin-1') as f:
        content = f.readlines()

    # remove newline characters and split each line into a list of values
    content = [s.replace(', ', ',').replace(' ,', ',').replace(']', '').replace('[', '') for s in content]
    content = [line.strip().strip('()').split(',') for line in content]
    df = pd.DataFrame(content)
    return df
```

3) We combined all the Data Frames and cleaned them

```python
# Join the DataFrames from both of the files
finaldf = pd.concat([read_and_create_df(file1),
                     read_and_create_df(file2)], ignore_index=True)

# Changing the first column to 'category' - that present the label column
finaldf.rename({0:'category'}, axis=1, inplace=True)

print(f"The full df has been loaded and contain {finaldf.shape[0]} rows")
print("Example of the data:")
print(finaldf.head(3))  # present 3 first rows


print("\n -----Clean the DataFrame------- ")
# Drop the rows with null, and in addition every row that returns an exception will be dropped too
rows_to_drop = finaldf[finaldf.isnull().any(axis=1)].index.tolist()

for i in range(len(finaldf)):
    try:
        finaldf.iloc[i, :] = finaldf.iloc[i, :].astype('float64')
    except ValueError:
        rows_to_drop.append(i)

print(f"There are {len(rows_to_drop)} rows to drop")
finaldf = finaldf.drop(rows_to_drop).reset_index(drop=True)
print(f"After dropping, the df contain {finaldf.shape[0]} rows")
```

The output on screen:

```
 -----loading the files-------
first folder loaded
second folder loaded

 -----Convert the file's content into DataFrame-------
The full df has been loaded and contain 2225 rows
Example of the data:
  category  1   2   3   4   5   6   7   8  ...  92  93  94  95  96  97  98  99 100
0        1  -1  -1  -1  -1  -1  -1  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1  -1  -1
1        1  -1  -1  -1  -1   1   1   1   1  ...  -1  -1  -1  -1  -1  -1  -1  -1  -1
2        1  -1  -1  -1  -1  -1  -1  -1  -1  ...  -1  -1  -1  -1  -1  -1  -1  -1  -1

[3 rows x 101 columns]

 -----Clean the DataFrame-------
There are 267 rows to drop
After dropping, the df contain 2078 rows
```

4) Separate the data frame according to the relevant comparisons for each letter

```
print("\n -----The data contains 3 types of Hebrew letters: מ,ל,ב ------- ")
print(" -----Creating 3 tables to Comparisons: (ל,ב),(מ,ב),(מ,ל) ------- ")
condition1 = finaldf['category'] == 1.0  # 1 represent the letter ב
condition2 = finaldf['category'] == 2.0  # 2 represent the letter ל
condition3 = finaldf['category'] == 3.0  # 3 represent the letter מ
data23 = finaldf[~condition1]
data13 = finaldf[~condition2]
data12 = finaldf[~condition3]
data23 = data23.reset_index().drop('index', axis=1)
data13 = data13.reset_index().drop('index', axis=1)
data12 = data12.reset_index().drop('index', axis=1)

print(f"Shape of data12: {data12.shape}")
print(f"Shape of data23: {data23.shape}")
print(f"Shape of data13: {data13.shape}")
```

The output on screen:

```
 -----The data contains 3 types of Hebrew letters: מ,ל,ב -------
 -----Creating 3 tables to Comparisons: (ל,ב),(מ,ב),(מ,ל) -------
Shape of data12: (1391, 101)
Shape of data23: (1378, 101)
Shape of data13: (1387, 101)
```

5) Creating a class called Adaline, an object of this class is our classifier, and it's functions:

```
# Adaline class
class Adline:
    def __init__(self, lr=0.00001, n_iter=1000):
        self.lr = lr
        self.n_iter = n_iter
```

6) fit function that trains the object according to Adaline's algorithm

```python
def fit(self, X, y):
    self.weights = np.zeros(1 + X.shape[1]).reshape(-1, 1)

    # self.weights = np.random.rand(X.shape[1]+1, 1)
    # self.weights = np.full((101,1),0.001)
    self.errors = []

    for i in range(self.n_iter):
        output = self.activation_function(self.net_input(X))
        Y = y.values
        output = output.reshape(-1, 1)
        errors = Y - output

        self.weights[1:] = self.weights[1:] + self.lr * X.T.dot(errors)
        self.weights[0] = self.weights[0] + self.lr * errors.sum()
        cost = (errors ** 2).sum() / 2.0
        self.errors.append(cost)

        if i > 2 and np.isclose(self.errors[-2], self.errors[-1], rtol=1e-4):
            return i, errors
            break
    return self
```

7) The following function calculates the value received in the input of the vector according to the weights (activation function)

```python
def net_input(self, X):
    return np.dot(X.astype('float64'), self.weights[1:]) + self.weights[0]
```

8) The following function receives a vector that represents an image and returns a prediction of the letter that the vector represents

```python
def predict(self, X):
    return np.where(self.net_input(X) >= 0.0, 1, -1)
```

9) The following function returns the data members of the object

```python
def get_params(self, deep=True):
    """
    Get parameters of the Adline model.
    """
    return {
        'lr': self.lr,
        'n_iter': self.n_iter,
        'weights': self.weights
    }
```

10) Cross-validation

```python
def cross_validate(self, X, y, n_folds=10):
    """
    Perform n-fold cross-validation for the Adline model on the given data.
    """
    kf = KFold(n_splits=n_folds)
    scores = []
    iters = []
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index[0]:], X[test_index[0]:]
        y_train, y_test = y[train_index[0]:], y[test_index[0]:]
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.fit_transform(X_test)
        (i, cost) = self.fit(X_train, y_train)
        iters.append(i)
        score = self.score(X_test, y_test)
        scores.append(score)

    return scores, np.mean(scores), iters, cost
```

11) A function that calculates the accuracy of the classifier we created

```python
def score(self, X, y):
    """
    Return the accuracy score for the Adaline model on the given data.
    """
    y_pred = self.predict(X)
    accuracy = accuracy_score(y, y_pred)
    return accuracy
```

12) Activation function definition

```python
def activation_function(self, X):
    return X
```

13) A function that receives a Data Frame and returns a final result, intermediate results, average and cost

```python
def trainSubTable(df):
    X = df.drop(columns=['category'])
    y = pd.DataFrame(df.category)

    # Convert the values in the pairs' comparison tables to 1 and -1 instead of the original label
    if (df['category'] == 3.0).any():
        y = y.replace(3.0, -1.0)
    elif (df['category'] == 2.0).any():
        y = y.replace(2.0, -1.0)
    if (df['category'] == 2.0).any() and (df['category'] == 3.0).any():
        y = y.replace(2.0, 1.0)

    # Scale features
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    kf = KFold(n_splits=5, random_state=42, shuffle=True)
    adaline = Adline()

    scores, scores_mean, i, cost = adaline.cross_validate(X_train, y_train, n_folds=5)
    final_scores = adaline.score(X_test, y_test)

    d = {'score': final_scores}
    return d, scores, scores_mean, i, cost
```

14) Running the program and printing the results

```python
print("\n -----Calculating the datasets ------- ")
test_score12,scores12,scores_mean12,i12,cost12 = trainSubTable(data12)
test_score13,scores13,scores_mean13,i13,cost13 = trainSubTable(data13)
test_score23,scores23,scores_mean23,i23,cost23 = trainSubTable(data23)

# Calculate the accuracy of the model on the test set
std_dev12 = np.std(scores12)
std_dev13 = np.std(scores13)
std_dev23 = np.std(scores23)

print("\n -----Predicting by Adaline algorithm 1(ב) versus 2(ל) ------- ")
print("Std accuracy scores: ", std_dev12)
print("Average number of iterations before convergence:", i12, " Mean:", np.mean(i12))
print("accuracies in the cross validation:", scores12)
print("Average accuracy across all folds:", scores_mean12)
print("Test accuracy score:", test_score12)


print("\n -----Predicting by Adaline algorithm 1(ב) versus 3(מ) ------- ")
print("Std accuracy scores: ", std_dev13)
print("Average number of iterations before convergence:", i13, " Mean:", np.mean(i13))
print("accuracies in the cross validation:", scores13)
print("Average accuracy across all folds:", scores_mean13)
print("Test accuracy score:", test_score13)
```

Output on the screen:

```
 -----Predicting by Adaline algorithm 1(ב) versus 2(ל) -------
Std accuracy scores:  0.0037165548522343226
Average number of iterations before convergence: [344, 287, 287, 287, 287]  Mean: 298.4
accuracies in the cross validation: [0.8794964028776978, 0.8875140607424072, 0.8888888888888888, 0.8806306306306306, 0.8828828828828829]
Average accuracy across all folds: 0.8838825732045015
Test accuracy score: {'score': 0.8494623655913979}

 -----Predicting by Adaline algorithm 1(ב) versus 3(מ) -------
Std accuracy scores:  0.013758911749682095
Average number of iterations before convergence: [278, 227, 227, 227, 227]  Mean: 237.2
accuracies in the cross validation: [0.8259693417493237, 0.8387824126268321, 0.837593984962406, 0.8306997742663657, 0.8009049773755657]
Average accuracy across all folds: 0.8267900981960986
Test accuracy score: {'score': 0.8525179856115108}

 -----Predicting by Adaline algorithm 2(ל) versus 3(מ) -------
Std accuracy scores:  0.006863574212726591
Average number of iterations before convergence: [361, 303, 303, 303, 303]  Mean: 314.6
accuracies in the cross validation: [0.8838475499092558, 0.8887627695800226, 0.8863636363636364, 0.8795454545454545, 0.9]
Average accuracy across all folds: 0.8877038820796738
Test accuracy score: {'score': 0.8659420289855072}
```