Machine Learning & Big Data – Assignment 4

Arda Harman

Introduction

In the previous assignment, we made a linear regression model which would only classify 2 labels (or classes)

But now, we have 10 different classes

Problem is this:

A single model can only say “this result belongs to class … or not”

It can’t determine which class does it belongs to

That is why there are more than one model is used

Every model is for a every class

In this assignment, we have handwritings which can have values between 0-9 (10 different classes)

- One model will determine whether the result is 0 or not
    - But it can’t say what it is if it is not 0
- Another model will determine whether the result is 1 or not
    - But it can’t say what it is if it is not 1
- This is like that until 9

We will be training our models with 5000 example

- Each example will have 400 features
    - 1 estimation will be like this:
    - $f_{w,b}(x) = w0x0 + w1x1 + … + … w400x400$
- We will have 5000 estimations (since we have 5000 examples)

In this assignment, we have 2 parts:

- Part A) Logistic regression

- Part B) Neural Network

Part A) Logistic Regression

X: 5000x400 array

Contains 5000 examples (with 400 columns)

Models: 10x400 array

If there would be 2 classes, we would only need one model

- And that model would only have 1 row with 400 columns (features)

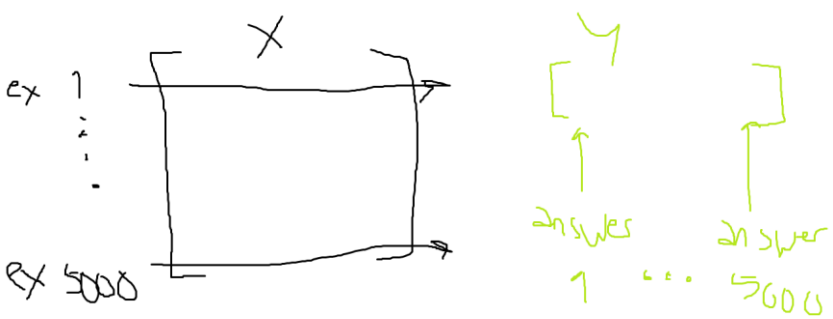
But since in this problem we have 10 classes, (and we need 10 models) we will have a matrix with 10 rows with 400 columns

- Again, 400 columns are features of the each model
- Each row is a model
    - Row 0 determines whether the output class is 0 or not
    - Row 1 determines whether the output class is 1 or not etc.

This part A) has 2 functions:

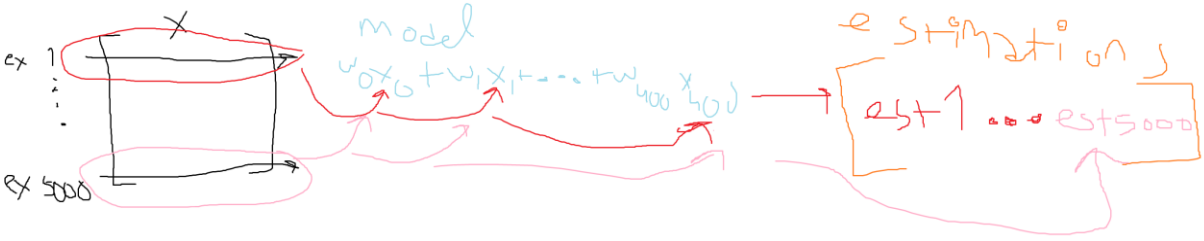
- oneVsAll
    - This function is for training all our models
        - In other words, it is for finding the best adjustments for each weight
    - After the execution of this function, we will get a trained model

- predictOneVsAll
    - This makes predictions with the derived model
        - Like “example 1 (out of 5000 examples) belongs to class 0”
            - And it has 5000 predictions for each answer



- oneVsAll

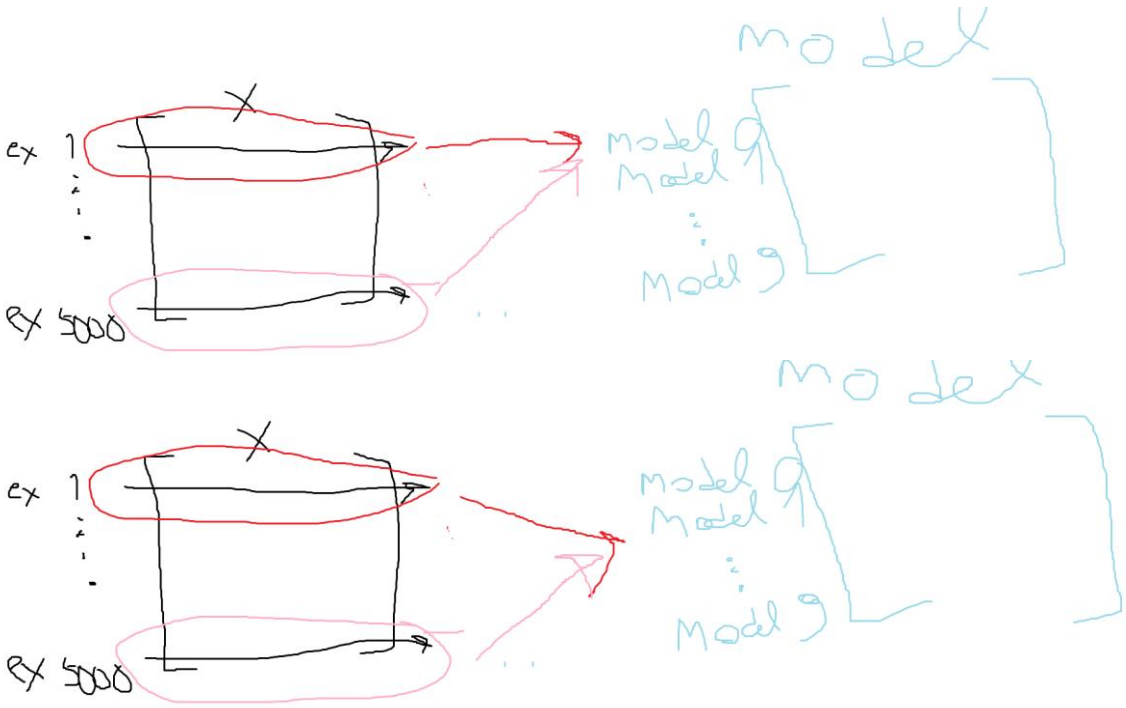
  Normally, training a “single model” is like this:

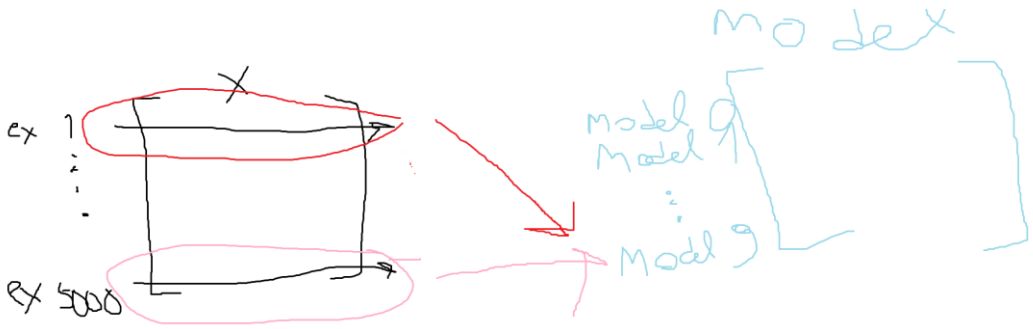


  For 5000 times, we would give (5000 examples) inputs and we would get 5000 estimations (with 1 row and 5000 column)

  With 10 models, it is actually same

  - In models array we have 10 models
    - !! EACH ROW CORRESPONDS TO A MODEL !!
      - Row 0 is only containing model for determining class is 0 or not etc.
  - In estimations, we will also have 10 rows
    - !! EACH ROW WILL CORRESPOND TO A MODEL !!
      - Row 0 will only contain estimations that model for model determining class is 0 made etc.





Other than that, there is nothing much different than the previous assignment and it is just logistic regression with 10 models at the same time

```
def oneVsAll(X, y, n_labels, lambda_):
    Trains n_labels logistic regression classifiers and returns
    each of these classifiers in a matrix all_theta, where the i-th
    row of all_theta corresponds to the classifier for label i.

    Parameters
    ----------
    X : array_like
        The input dataset of shape (m x n). m is the number of
        data points, and n is the number of features.

    y : array_like
        The data labels. A vector of shape (m, ).

    n_labels : int
        Number of possible labels.

    lambda_ : float
        The logistic regularization parameter.

    Returns
    -------
    all_theta : array_like
        The trained parameters for logistic regression for each class.
        This is a matrix of shape (K x n+1) where K is number of classes
        (ie. `n_labels`) and n is number of features without the bias.
    """

    # for each model, I am initializing w and b values...
    w_matrix = np.zeros((n_labels, len(X[0])))  # row 1 -> model 1    row 2 -> model 2
    b_matrix = np.zeros(n_labels) # column 1 -> model 1    column 2 -> model 2

    y = np.array([np.where(y == c, 1, 0) for c in range(10)])

    for i in range(1000):

        estimations = np.dot(w_matrix, X.T) + b_matrix[:, np.newaxis]
        estimations = 1 / (1 + np.exp(-estimations))

        #print(len(estimations), len(estimations[0]))

        dj_db = np.sum(estimations - y) / len(y)
        dj_dw = np.dot(estimations - y, X) / len(y) + lambda_* (1 / len(y))*w_matrix

        w_matrix = w_matrix - np.multiply(0.006, dj_dw)
        b_matrix = b_matrix - np.multiply(0.006, dj_db)

        loss = ( (np.dot(np.log(estimations), -y.T)) - np.dot(np.log(1-estimations), 1 - y.T) ) * (1 / len(y) )
        loss = np.sum(loss)
        lambda_part = np.sum((lambda_ / (2*len(y)) ) * np.square(w_matrix))
        total_cost =  loss + lambda_part
        print(total_cost)

    all_theta = np.hstack((b_matrix.reshape(-1, 1), w_matrix))


    '''
    print("b_matrix:")
    print(len(b_matrix))
    print("w_mattix")
    print(len(w_matrix), len(w_matrix[0]))
    print("all_theta")
    print(len(all_theta), len(all_theta[0]))
    '''

    return all_theta
```

I just do X.T to being able to obtain a model array where every row corresponds to a specific model

I use npwhere to produce an array for an answer key for all models

- Let's say our target array is like [0, 2, 1, 4]
  - When training a model which either says "label is 0 or not" target array should be like [1, 0, 0, 0]
    - Because estimation should also say (first element) 0 is 0. (Second element) 2 is not 0 etc
  - For model which either says "label is 1 or not" it should be [0, 0, 1, 0]

This way, I can train all different models

- predictOneVsAll

```
def predictOneVsAll(all_theta, X):
    """
    Return a vector of predictions for each example in the matrix X.
    Note that X contains the examples in rows. all_theta is a matrix where
    the i-th row is a trained logistic regression theta vector for the
    i-th class. You should set p to a vector of values from 0..K-1
    (e.g., p = [0, 2, 0, 1] predicts classes 0, 2, 0, 1 for 4 examples) .

    Parameters
    ----------
    all_theta : array_like
        The trained parameters for logistic regression for each class.
        This is a matrix of shape (K x n+1) where K is number of classes
        and n is number of features without the bias.

    X : array_like
        Data points to predict their labels. This is a matrix of shape
        (m x n) where m is number of data points to predict, and n is number
        of features without the bias term. Note we add the bias term for X in
        this function.

    Returns
    -------
    p : array_like
        The predictions for each data point in X. This is a vector of shape (m, ).
    """

    '''print("all_theat: ")
    print(all_theta)
'''
    X_with_bias = np.insert(X, 0, 1, axis=1)

    probabilities = np.dot(all_theta, X_with_bias.T)

    p = np.argmax(probabilities, axis=0)
```

With "probabilities" I store the estimation of each (our trained) model

And then with the argmax, I take indices which has the highest value

- If we have value 2 for example:
  - Model which looks whether label is 0 or not would find a low value
  - Model which looks whether label is 1 or not would find a low value
  - Model which looks whether label is 2 or not would find a high value
- So,taking the indices of highest value, we technically choose a label as our estimation

Part B) Neural Networks

We only have “predict” function because weights are already trained

- predict

  Interestingly, neural networks are extremely similiar (maybe even exactly same) things

  - At logistic regression, we have multiple models where input comes and they generate an output
  - At neural networks, we have multiple “activations” where input comes and they generate an output

  What made neural networks more successful than logistic regression here is that “logistic regression” had “2 hidden layers”

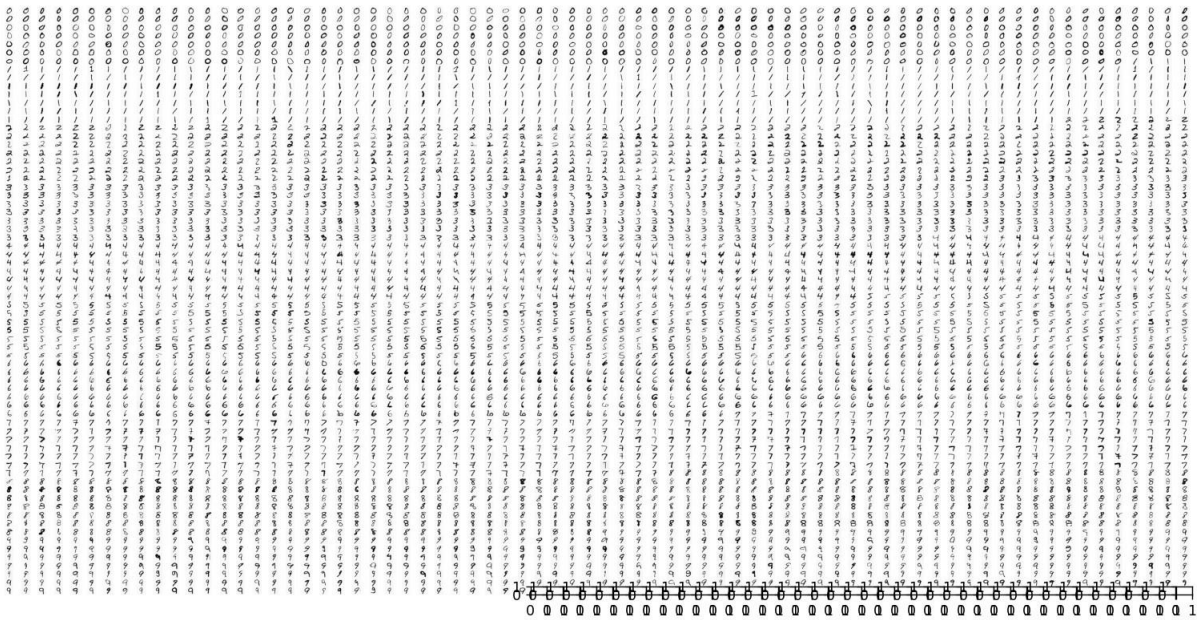
  - It first uses a layer with 25 activations and then with 10 activation
  - If the first layer wouldn’t exist, it wouldn’t be any different than logistic regression

Proof everything Works:

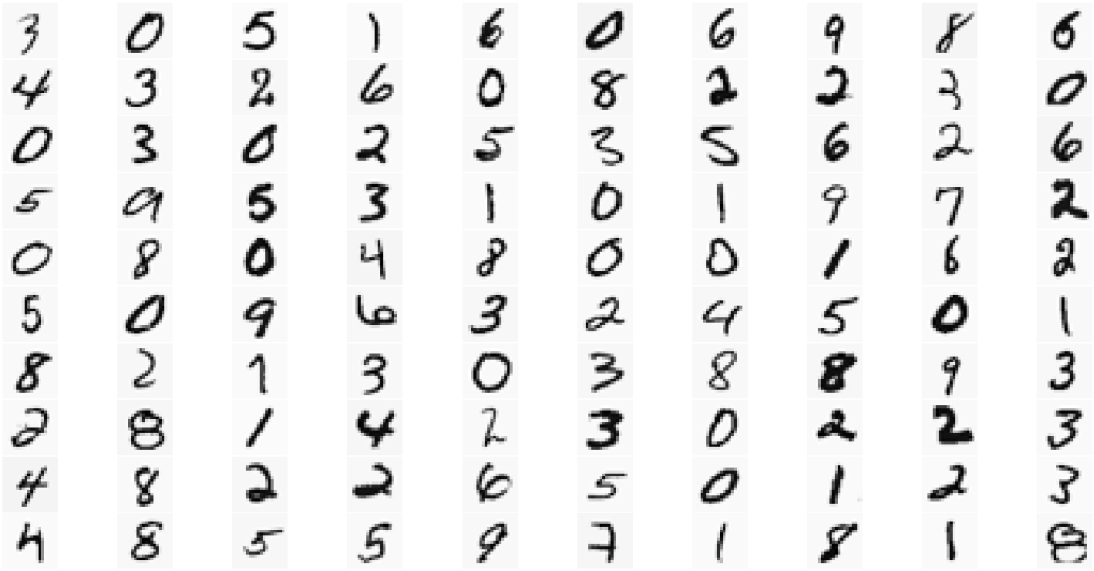
```
print("y : " ,  len(y))
print("log reg: " + str(len(predict_log_reg)) + ", ")
print("neural net: " + str(len(predict_neural_netw)) + ", ")


# Count the number of matching elements
predictions_log_reg = np.sum(y == predict_log_reg)
predictions_neural_netw = np.sum(y == predict_neural_netw)

# Calculate the total number of elements
total_examples = len(y)

# Calculate the accuracy
accuracy_log_reg = (predictions_log_reg / total_examples) * 100.0
accuracy_neural_netw = (predictions_neural_netw / total_examples) * 100.0

print("Accuracy - logistic regression: ", accuracy_log_reg, "%")
print("Accuracy - neural network: " , accuracy_neural_netw, "%")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\ardah\OneDrive\Masaüstü\p4> python -u "c:\Users\ardah\OneDrive\Masaüstü\p4\multi_class.py"
y :  5000
log reg: 5000,
neural net: 5000,
Accuracy - logistic regression:  93.96 %
Accuracy - neural network:  97.52 %
PS C:\Users\ardah\OneDrive\Masaüstü\p4>
```

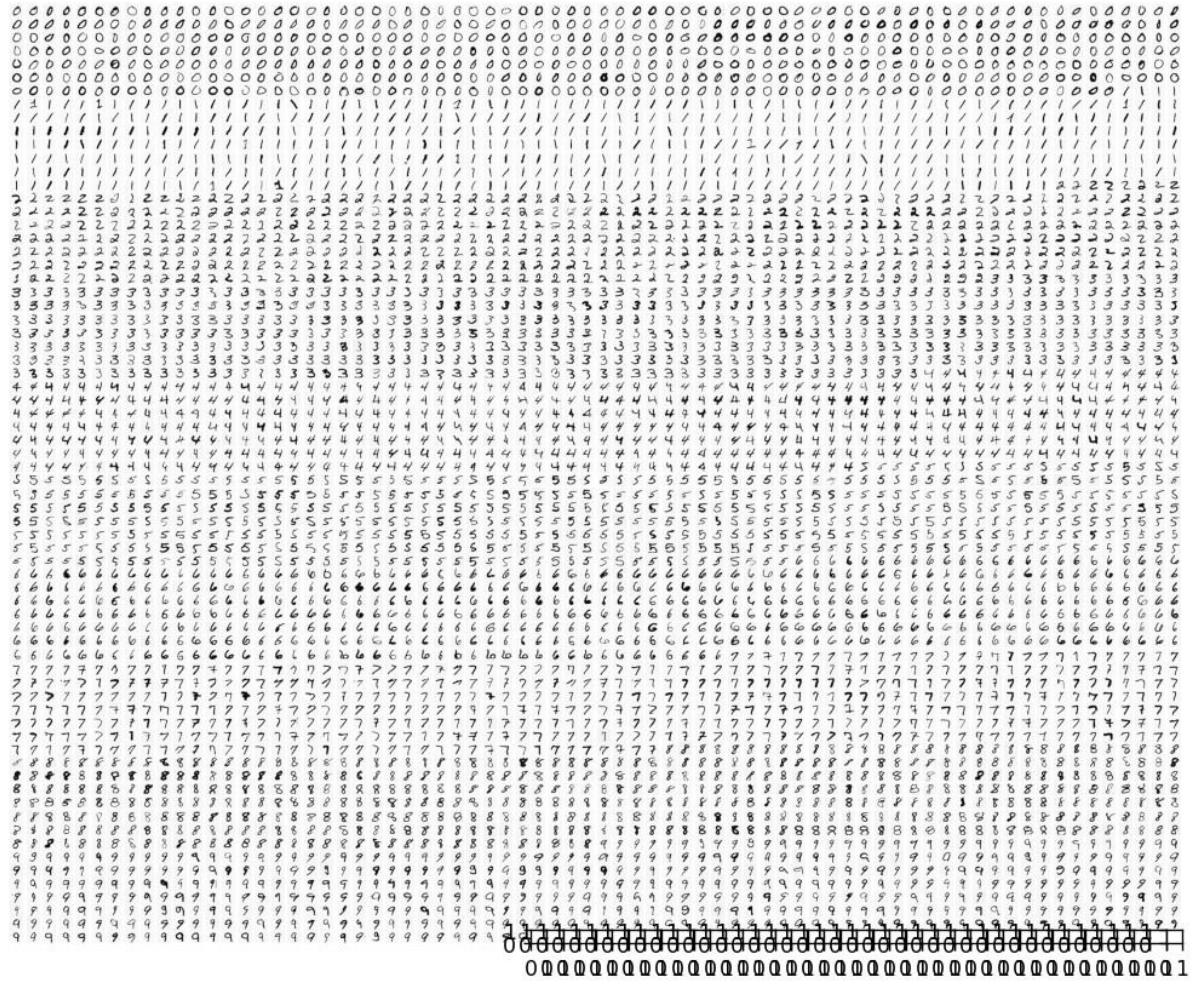
target array graph:



Input array graph:



Logistic reg predictions graph:



Neural network graph: