SPAM FILTERING REPORT

Thi Thu Trang Do 17774

1. Project Description

Project description: Given 57 features of 4601 emails and their labels, classify them into 2 classes: spam and non-spam.

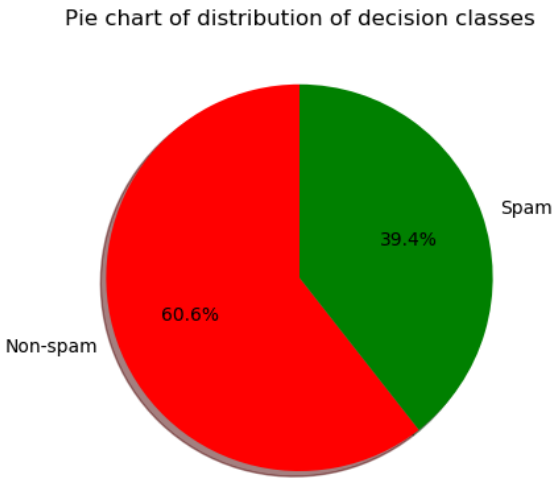
Number of records: 4601

Number of attributes: 57

Decision classes: 0 (non-spam) and 1 (spam)

Distribution of decision classes: Class 0 (non-spam) has 2788 emails (60.6%) and class 1 (spam) has 1813 emails (39.4%).

Programming language: Python 3



2. Data Preprocessing

a. Fetching data:

Using library pandas to read data from file.

X: set of features of each example (size of X: (4601, 57))

y: set of labels for each example (size of y: (4601, 1))

b. Feature selection: Mutual Information

Using library sklearn, module feature_selection.mutual_info_classif

Calculate mutual information (MI) between each feature of X and the decision classes in y. Keep only features that has MI (mutual information) greater than 0.05, which means that those features are more than 5% related y.

After feature selection, number_of_features is in range of [25, 30] (inclusive)

X_new: new set of features (size of X: (4601, number_of_features))

c. Normalization of values of features in X

For each example $X_i = (X_{i,1}, X_{i,2}, X_{i,3}, \ldots, X_{i,n})$ in X

Where: X_i is i-th example in X

$X_{i,1}, X_{i,2}, X_{i,3}, \ldots, X_{i,n}$: features 1, 2, 3, …, n of example X_i

Calculate: $||X|| = \sqrt{(X_{i,1})^2 + (X_{i,2})^2 + (X_{i,3})^2 + \cdots + (X_{i,n})^2}$

We normalize X by:

$$X\_new = \frac{X}{||X||}$$

In case ||X|| = 0, set ||X||= 1

d. Splitting data

Shuffle the data randomly and then split them into training set (80%) and test set (20%). Training set has X_train and y_train. Test set has X_test and y_test

3. Neural network – Backpropagation Algorithm

a. Network configuration

Number of hidden layers = 2
Number of neurons in input layer = number_of_features
Number of neurons in hidden layer 1 = 15
Number of neurons in hidden layer 2 = 7
Number of neurons in output layer = 1

Learning rate = 0.0001
Number of epochs = 20

Save X_train, X_test, y_train, y_test as final_X_train, final_X_test, final_y_train, final_y_test for later use.

Activation function: unipolar sigmoid function

b. Initialize weights and biases

Initialize weights and biases with standard normal distribution using numpy.random.rand. Values of weights and biases are in range of [-1, 1].
Weight matrices: W1, W2, W3
Bias vectors: B1, B2, B3

c. Train model

<u>For each epochs:</u>

Step 1. Shuffle the data, split it into X_train, X_test, y_train, y_test

Step 2. For each example in training set(X_train, y_train):

2.1. Calculate net and activation for each layer

hidden1_net = W1.X + B1
A1 = sigmoid(hidden1_net)

hidden2_net = W2.A1 + B2
A2 = sigmoid(hidden2_net)

output_net = W3.A2 + B3
A3 = sigmoid(output_net)

Then, label y_pred as 0 (non-spam) or 1 (spam).
If A3 >= 0, y_pred = 1, else y_pred = 0.

2.2. Calculate errors for each layer

a. Output layer: For each k-th neuron in output layer

$$\delta_k = f'(output\_net)(y\_desired_k - y\_pred_k)$$

Vectorized calculation:

δ = errors_output = np.multiply(f'(output_net), y_desired - y_pred)

(element-wised multiplication for np.multiply)

b. Hidden layers:

For each j-th neuron in hidden layer 2

$$\rho_j = f'(hidden2\_net) \sum_{k=1}^{K} (W3_{k,j}.\delta_k)$$

Vectorized calculation:

ρ = errors_hidden2 = np.multiply(f'(hidden2_net), np.dot(W3.T, errors_output))

For each i-th neuron in hidden layer 1

$$\vartheta_i = f'(hidden1\_net) \sum_{j=1}^{J} (W2_{j,i}.\rho_j)$$

Vectorized calculation:

ϑ = errors_hidden1 = np.multiply(f'(hidden1_net), np.dot(W2.T, errors_hidden2))

2.3. Update weights and normalize weights

a. Output layer: For each k-th neuron in output layer

$$W3_k = W3_k + \eta.\delta_k.A2$$

$$B3_k = B3_k + \eta.\delta_k$$

Vectorized calculation:

$$W3 = W3 + \eta * np.dot(errors\_output, A2.T)$$

$$B3 = B3 + \eta * errors\_output$$

b. Hidden layers:

For each j-th neuron in hidden layer 2

$$W2_j = W2_j + \eta.\rho_j.A1$$

$$B2_j = B2_j + \eta.\rho_j$$

Vectorized calculation:

$$W2 = W2 + \eta * np.dot(errors\_hidden2, A1.T)$$

$$B2 = B2 + \eta * errors\_hidden2$$

For each i-th neuron in hidden layer 1

$$W1_i = W1_i + \eta.\vartheta_i.X$$

$$B1_i = B1_i + \eta.\vartheta_i$$

Vectorized calculation:

$$W1 = W1 + \eta * np.dot(errors\_hidden1, X.T)$$

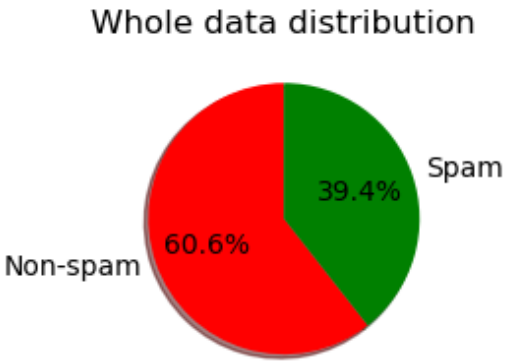
$$B1 = B1 + \eta * errors\_hidden1$$

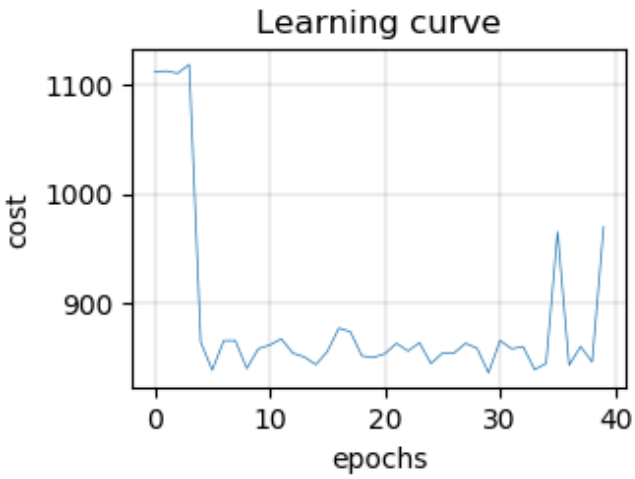
Step 3. Calculate the cost and add it to costs list.

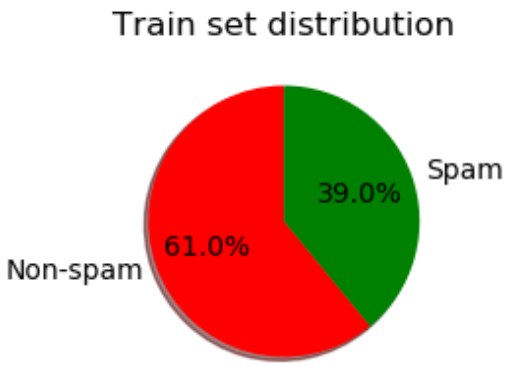
Compare it with min_cost, save the min_cost and its corresponding W1, W2, W3, B1, B2, B3.

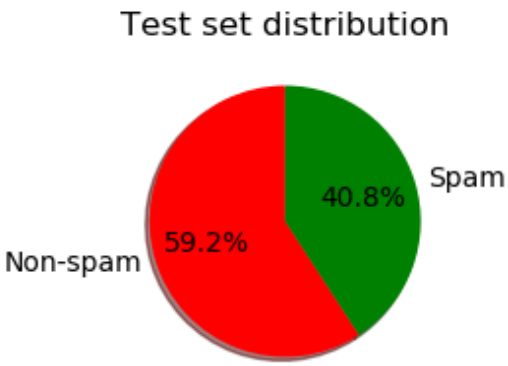
Step 4. The algorithm stops when we run through all epochs.

4. Accuracy









Accuracy:

- On training set: 53.23%
- On test set: 88.72%

The neuron works better on test set.

The distributions of decision classes in train set and test set are similar to each other and similar to distribution of the whole data. The classifier works better on small scale.

Confusion matrix:

- On training set

  $$\begin{bmatrix} 1437 & 819 \\ 902 & 522 \end{bmatrix}$$

  Class non-spam accuracy: 63.7%

  Class spam accuracy: 63.34%

- On test set

  $$\begin{bmatrix} 363 & 169 \\ 246 & 143 \end{bmatrix}$$

  Class non-spam accuracy: 68.23%

  Class spam accuracy: 63.24%

  ➔ The accuracies of class spam and non-spam are similar.