Spam Mail Perceptron Classification

May 14, 2020

Benjamin Pepper

Data 882: Statistical Learning II

Final

1 Introduction:

The goal of this project was to train a neural network to classify emails as either “spam” or “non-spam”. This was done on the provided “spam.tsv” dataset which contains 57 features representing the frequency of their occurence within 4601 emails by their numeric values. The response colume “type” is the only categorical variable with “spam” and “nonspam” as the two factors.

2 Methods:

For the response variable “type”, “spam” was encoded as 1 for the positive class and “nonspam” was encoded as 0 for the negative class.

A perceptron was then trained and tested via **4-fold cross validation** on the “spam.tsv” dataset with the **batch size** set high enough to ensure that all of the samples are propagated through the network in a single pass for each of 10,100 **epochs**, resulting in only a single update to the perceptron’s parameters for each epoch.

The structure of the perceptron consisted of an **input layer** with 57 nodes to correspond to each of the 57 features, a **hidden layer** with 12 nodes, and an **output layer** with 2 nodes: the first can be interpreted as the probability of an email being “nonspam” and the second as the probability of “spam”. The output node with the highest probability determines the classification of an email.

The **sigmoid function** was chosen as the activation function for each of the three layers, **binary cross entropy** as the loss function, and the **Adam optimizer algorithm** for its adaptive learning rate and momentum.

To **prevent overfitting**, generous **dropout layers** of 0.5 were implemented between the three layers. This randomly set 50% of the input elements from a previous layer to 0 which prevented nodes from relying on the output of other nodes, requiring them to be independently useful.

To ensure **reproducible results**, all random seeds where set to 1: tf.random.set_seed() from tensorflow and np.random.seed() from numpy.

Finally, an **early stopping** measure was implemented whereby the network would cease training after failing to lower the loss across 1000 epochs.

```
[ ]: import numpy as np
     import importlib

     import spam_functions
     import spam_nested_cv

     root = '../Data'

     dat = spam_functions.get_spam_dat(root)
     dat = dat.dropna()

     X = dat.drop('type', axis=1)
     y = np.array(dat['type'])

     importlib.reload(spam_functions)
     res_nn = spam_nested_cv.nested_cv(spam_functions.nn_mod, X, y,
                                       spam_functions.nn_score,
                                       spam_functions.nn_outer_perf,
                                       cv_k1 = 4, seed = 1)
```

3 Results:

The accuracy for the 4-folds of the cross validation:

```
[5]: res_nn['Perf']
```

```
[5]: [0.9539530842745438,
      0.9460869565217391,
      0.9556521739130435,
      0.9547826086956521]
```

The mean accuracy across the 4-folds:

```
[6]: np.mean(res_nn['Perf'])
```

```
[6]: 0.9526187058512445
```

4 Analysis and Discussion:

Despite the accuracy for fold 2 being slightly lower than the other three folds, all performed similarily. The mean accuracy of 95.26% is high; however, it should be noted that the classifier weighted the cost of making a false positive equal to that of a false negative. This may not be the case if for example we view filtering out genuine emails into the spam folder to be worse than leaving spam in the main folder. Then, we would want to make false positives more costly.