Machine Learning Report

Dirk Brink
CID - 01065264
Imperial College London
djb15@ic.ac.uk

I, Dirk Brink, pledge that this assignment is completely my own work, and that I did not take, borrow or steal work from any other person, and that I did not allow any other person to use, have, borrow or steal portions of my work. I understand that if I violate this honesty pledge, I am subject to disciplinary action pursuant to the appropriate sections of Imperial College London.

1. The Problem

This project is concerned with email spam classification (using dataset a). The goal of this project is to create a machine learning model that correctly categorizes emails as spam or not spam, with as high an accuracy as possible. The input data includes a set of 57 features that can be used by the model. This machine learning problem is a binary classification problem, and hence I will be utilizing classification based methods.

1.1. Loss function

Classifying emails, since this is a binary problem, has 4 possible outcomes. These 4 outcomes can easily be seen from a confusion matrix.

Two of the quadrants are correct classification either as spam or not spam. The other 2 quadrants correspond to either false positive or false negative (where positive indicates a spam email). In the problem of email classification, false positive is more undesirable than false negative. This is due to the fact that a false positive means you may never see a completely legitimate email since it goes straight into your spam folder. Although frustrating, it is better for a few spam emails to reach your inbox than lose legitimate emails, and as such the false positive case should have a higher error weighting.

2. Baseline Classifiers

I started this project by implementing a few basic classifiers in order to obtain a base level upon which I should try to improve. The simplest algorithm I could think of was the perceptron algorithm, so this was the first one I implemented.

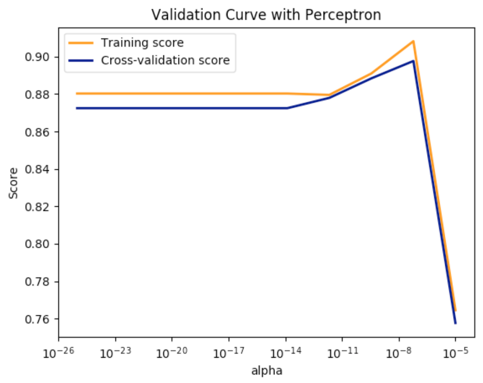


Figure 1. Perceptron model validation curve

2.1. Perceptron Classifier

The perceptron algorithm is very simple and only includes a couple of tunable parameters. The parameters that I chose to edit and control were `penalty` (the penalty function to use), `alpha` (the constant that multiplies the regularization term), `max_iter` (the maximum number of epochs when training) and `tol` (the stopping criterion).

Through a lot of trial and error I discovered that changing `max_iter` was not very influential and so I settled upon a fairly large value of 10000 which I have used throughout the project. Since `tol` is correlated to `max_iter`, changing this value also had no real effect on training and validation. As a result, I varied `alpha` having picked an L2 penalty (because I wanted a loss function that tended slowly to 0 as the input data cannot be completely separated by a linear method). As can be seen from the validation curve in Fig.1, training and validation were both fairly high when `alpha` was small enough. Also, the validation curve shows that there is no real problem of over-fitting at any value of alpha.

The confusion matrix in Fig.2 shows the classification per-

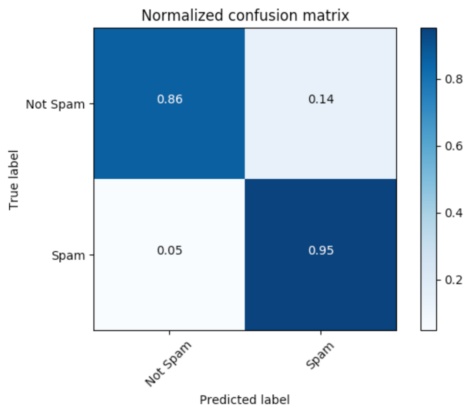
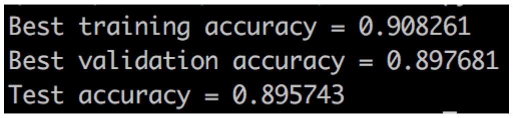


Figure 2. Perceptron model confusion matrix



```
Best training accuracy = 0.908261
Best validation accuracy = 0.897681
Test accuracy = 0.895743
```

Figure 3. Perceptron model accuracy

C	Loss	Dual	Penalty
0.01,0.1,1,10,100	squared_hinge	True, False	L2
0.01,0.1,1,10,100		False	L1

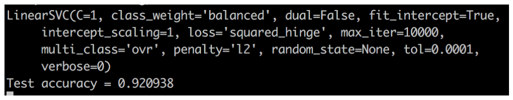
Table 1. Linear SVM grid search parameters

formance split, and overall performance of this perceptron model is around 89.6% accurate on a test set (as can be seen in Fig.3). As a note, the train-test split throughout this project is 70-30.

2.2. Linear SVM

The other baseline classifier I decided to use was a linear support vector machine. Linear methods are the simplest machine learning models and as such they were used as a baseline. A SVM has more parameters than a perceptron model and so in order to select the best parameters, I performed an exhaustive grid search instead of plotting a validation curve and manually selecting the best parameters. The parameters I varied were C and the penalty function (L1 or L2), and the ranges can be seen in Table.1. Not all the possible combinations are grid searched, since restrictions are placed on some of the permutations.

After performing a grid search, the best parameters for the model were L2 loss function with squared-hinge loss. The penalty C was optimum at 1. The optimum model can be seen in Fig.4 along with the test set performance of 92.1%. The distribution of correct and incorrect classification can be seen from the confusion matrix in Fig.5.



```
LinearSVC(C=1, class_weight='balanced', dual=False, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=10000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)
Test accuracy = 0.920938
```

Figure 4. Linear SVM model parameters and test set accuracy

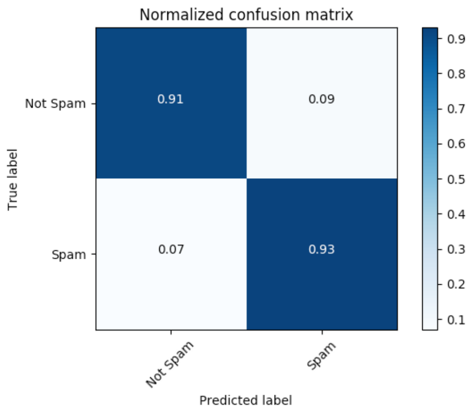


Figure 5. Linear SVM confusion matrix

2.3. Evaluation of baseline models

Overall, the linear baseline models performed a lot better than expected on this dataset. Since linear models are generally the simplest I expected to have some issues obtaining high percentage accuracy. However, so far they seem to generalize very well across the data and it will likely be hard to beat with other more complex models. As expected the SVM performed better than the perceptron since the algorithm is more complicated and elegant.

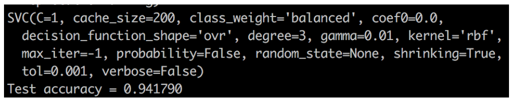
3. Proposing more advanced methods

My baseline classifiers were intentionally as simple as possible. It is not a difficult problem to use more advanced algorithms to classify the same dataset. When considering more advance models I generally believed that some non-linear models would provide better results than the baseline predictors. Of the available non-linear methods, SVM seemed like the logical choice since it can implement both RBF, polynomial and linear classifiers. By performing a grid search across these 3 parameters I will be able to directly compared the performance of linear models against non-linear. Generally I expect the non-linear models to perform better than their linear counterparts since they should provide a better fit to the complex nature of the problem. I do however expect high order polynomials to perform badly due to the over-fitting that is likely to occur.

Apart from non-linear methods such as SVM, the other very powerful option is a neural network made up of multiple layers of perceptrons. Neural networks are modeled around

Kernel	C	Degree	Gamma
linear	0.01,0.1,1,10,100		
poly	0.01,0.1,1,10,100	2,3,4,5	
rbf	0.01,0.1,1,10,100		0.1,0.01,0.001,0.0001

Table 2. SVM grid search parameters



```
SVC(C=1, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Test accuracy = 0.941790
```

Figure 6. SVM model parameters and test set accuracy

the human brain and are able to perform very well at classification as the number of layers increases (up to a point - a more complex neural network may perform a lot worse than a simple one as a result of over-fitting for example). As the most advanced method I'll be using to classify emails, I expect this model to perform the best. As with the SVM, I'll be implementing a grid search to select the best parameters to perform classification on the test set. This will by far take the longest out of all the models since a high number of parameters can be varied, and a multilayer network generally has a longer training time.

Overall I believe both of these methods will beat my baseline classifiers, although probably not by much since the linear models already have quite a high accuracy.

4. Training the advanced methods

4.1. SVM

As outlined in the previous section, for the selection of model parameters for SVM, I performed a comprehensive grid search. I decided to include the linear kernel in the grid search for a direct comparison between the linear and non-linear methods. If it turned out that linear was in fact better than non-linear for this dataset, then the grid search would make that apparent. During the first attempt at training it became apparent that nothing was happening and there was something wrong with the training setup. After consulting the Scikit Learn documentation [2], I discovered that data normalization was key for the SVC method. The raw email data has numbers that range from 0 - 100 with a non unitary standard deviation. Initially I tried to normalize the data between 0 and 1, however this proved not to work very well with some of the binary features. Normalizing the data from -1 to +1 seemed to work well and actually allowed the training to occur properly with my desired setup.

Table 2 shows the parameters I used for the SVM grid search. I chose each of the ranges equally spaced around the default value in Scikit Learn. From past experience and the docs, the default values generally give good performance

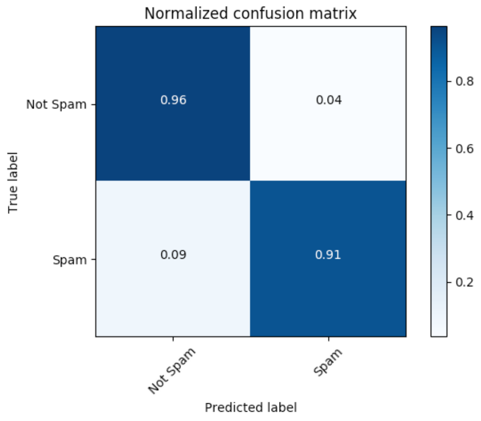


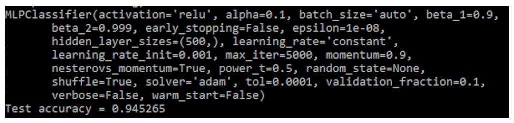
Figure 7. SVM confusion matrix (with RBF kernel)

and are a good starting point. It is wise to vary the parameters though since each dataset has different characteristics and will respond to different parameters in different ways. After performing a comprehensive grid search across all the combinations shown in the table, the model with the best parameters was selected as shown in Table 2 and Fig.6. Using these parameters I ran the model on the test set and obtained an accuracy of 94.2%. As expected this is better than the 2 baseline predictors, although actually not by much. This highlights that the 2 baseline predictors (especially the linear SVM) already model the dataset very well and that my final model (an MLP network) is unlikely to beat this, if at all.

4.2. MLP

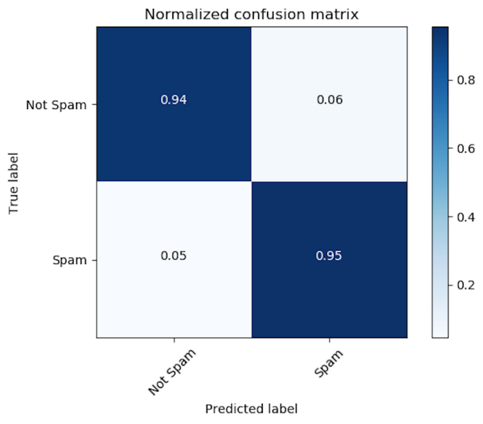
My final model for email classification is a MLP network. As with my previous models, I employed a grid search in order to find the best parameters for the model. An MLP network has the largest number of tunable parameters due to the various ways in which an MLP can be constructed. There can be a varying number of layers, a varying number of neurons in each layer, different loss functions, loss/dropout at each layer, etc. So, as expected, training the MLP network took by far the longest at a couple of hours to perform the comprehensive grid search. This is the reason grid searches are not performed in industry on neural networks, especially with large datasets. My grid search parameters can be seen in the `mlp.py` file (too many parameters to include in a table). Having finished the grid search, my results were marginally better than the SVM with an RBF kernel. The best parameters can be seen in Fig.8.

The test set accuracy was 94.5% as shown in Fig.8. This is a marginal increase compared to the SVM and could just be a fluke. As a result I ran a few trains on a randomly selected test set using the same model parameters, and the accuracy fluctuated around 93.8% to 94.7%. To give an equal com-



```
MLPClassifier(activation='relu', alpha=0.1, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(500,), learning_rate='constant',
       learning_rate_init=0.001, max_iter=5000, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
Test accuracy = 0.945265
```

Figure 8. MLP model parameters and accuracy



	Not Spam	Spam
Not Spam	0.94	0.06
Spam	0.05	0.95

Figure 9. MLP confusion matrix

parison, I also ran the SVM with the same parameters a few additional times, getting an accuracy range of 93.5% to 94.6%. You could argue that the MLP is therefore slightly better performing than the SVM, however personally I believe they give roughly identical performance.

4.3. Proposed solution

As discussed in the previous section, the performance of the MLP and the SVM are practically identical. With some more time and beefier compute resources, it is very likely that I could design a better neural network that would beat the SVM. However, I think the potential gains do not warrant the cost. Between the SVM and MLP I'm inclined to pick the SVM as my proposed solution. Although it arguably performs slightly worse than the MLP, the training time for the SVM is much faster and the model is simpler. For the application of email spam detection, I believe this is the best choice, since the possible marginal gains from an MLP would not be worth it, especially if you wanted to update the model parameters over time (because the MLP would take a lot longer to train and update and use more compute resources). If I had more training data/more features available, I would probably choose the MLP network since they generally perform better on such larger, more complicated datasets.

5. Conclusion

Overall, all of my models have performed fairly well at classifying emails as spam/not spam. The initial goal was to ensure that as much email was correctly classified as legitimate, with a low false positive percentage (where positive means the email is spam). This is due to the fact that losing legitimate email is more costly than seeing one or two spam emails. This was defined at the start of the report with the loss function. Without having to enforce the loss function, the models generally performed well in this regard. This is exactly the performance I was after so overall I am happy with the outcome of my models. In terms of accuracy, I believe my models all performed very well, especially the RBF kerneled SVC and the multiple layer perceptron. I can compare my performance with the results obtained in [1]. This article used exactly the same dataset and obtained a best performance of 92.3% correct classification. In comparison, my best result was 94.5%. I think it is possible however to beat my best model. I could have been more thorough with the grid search and used a wider range of parameters, however this would have increased the training time exponentially. There is almost certainly a better combination (you could say the optimum combination) of parameters that would provide a better test result. Nonetheless, the performance improvement would be so marginal that it may not be worth the extra training time and computing time, depending on the application.

References

[1] Comparison of machine learning methods in email spam detection. `https://www.matchilling.com/comparison-of-machine-learning-methods-in-/email-spam-detection/`. Accessed: 20/03/2018.

[2] Scikit learn. `http://scikit-learn.org/stable/index.html`. Accessed: 10/03/2018.