Optimization using Machine Learning for Spam Detection Accuracy

Haaris Osman Mehmood
Imperial College London

1. Introduction

Ever since the advent of emails, spam have been a huge nuisance for email users and increase the chances of contracting malware. The methods to classifying emails as spam or not spam has been studied in great detail since the 1990s. In this study, we focus on improving spam detection accuracy by tuning popular machine learning algorithms. The most basic baseline predictor gave us a result $\tilde{8}9\%$ which we were able to improve to $\tilde{9}5\%$ using state-of-the art methods described in section 3.

1.1. Background

The Spambase data-set [1] will be used to train and evaluate models for this problem. The data-set consist of records of personal and work emails as well as emails marked by users and postmasters at HP as SPAM. The dataset was generated in 1999 and hence some of the features present might not be relevant today. Instead of presenting the full corpus of an email, each email corpus has been transformed to a set of 57 features. As such, this transformation will form the basis for training all models described below since the original corpus is non-retrievable.

1.2. Learning Setup

1.2.1 Inputs and outputs

A summary for the type of features can be found in the appendix Each output is a binary classification with a 0 or 1 label. The label '1' represents a spam email while the label '0' represents a ham (not-spam) email. The notation used for inputs and outputs are described below.

$$\textit{Input: } \vec{x}_i \in \mathbb{R}^{57}$$

$$x_{i1}, x_{i2}, ..., x_{i54} \in [0, 1]$$

$$x_{i55}, x_{i56}, x_{i57} \in [1, +\infty)$$

$$\textit{Output: } y_i \in \{0, 1\}$$

1.2.2 Data-set summary

The data-set has a total of 4601 records. Approximately 39.4% of the records are classified as spam. The data-set will initially be divided into train:test split with a 0.75:0.25 ratio. A stratified train/test splitting function will be used which will ensure that the relative percentage of each class samples remain the same in both splits. To reduce the chances of generalization error, the records will also be randomized before split but using a constant random seed for result reproducibility and consistency.

$$(\vec{x}_1, y_1), (\vec{x}_2, y_2), ..., (\vec{x}_{4601}, y_{4601})$$

1.2.3 Loss function

Given that this is primarily a binary classification problem, the best loss function to choose is one that penalizes with a cost of 1 when the prediction is not the same as the output or 0 otherwise.

$$\ell(h, y) = \mathbb{I}(h \neq y)$$

They are several other scoring metrics also available to use such as the F1-score or ROC-AUC. These scores are mainly important when the split between classes is significant and/or preference is given to a particular class. Since the dataset has a $0.4 : 0.6$ split between classes, it is not deemed essential to evaluate performance using those metrics. The hyper-parameter selection will be done based on accuracy.

$$ACC = \frac{1}{N} \sum_{n=1}^{N} (1 - \ell(h_n, y_n))$$

Although this is the main loss function which will be used for hyper-parameter tuning, not all models will be able to use it directly for training (for reasons described later) and hence the training loss function will mainly depend on the method being used.

1.3. Evaluating Performance

1.3.1 Train-test split

The data-set will initially be split into training and testing with a $0.75 : 0.25$ ratio. Like the loss function, there is no agreed ratio for the train-test split but it is often found to that $70 - 80\%$ of data is part of the test-set. To main the relative ratios of the classes in each sub-dataset, a 'stratified' split is performed. This will reduce the chances of any model from over-predicting a single class during training. Overall roughly 3100 samples will be used for training during every fold so this would not have a huge impact on simpler models but it needs to be taken into consideration for more complex models.

1.3.2 Cross-validation

The golden technique for a credible evaluation of a model's performance is k-fold cross-validation. This involves splitting the data-set (in our case the training data) into k divisions and then training is performed on $k-1$ folds while the k_{th} fold is used for testing. This way, each fold gets used for testing only once and a more accurate mean score can be generated. Using the 10-fold cross validation method, N samples (from the k_{th}) test set will be used to evaluate the accuracy of the model and then an average score ($\overline{ACC}$) will be generated along with its variance.

1.3.3 Objective

The main objective of this task is to find the best possible hypothesis g which minimizes the expected loss or equivalently maximizes the expected accuracy. As long as we can ensure that the mean cross-validation accuracy ($\overline{ACC}$) tracks the original test-set accuracy (ACC_{test}) reasonably well, it can be reasoned that with high probability, the optimum function (g) found using cross-validation will satisfy the objective below for any $\vec{x}$.

$$Find\ g \in \mathcal{H} \mid g = \arg\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(\vec{x}), y)]$$

1.4. Feature Analysis

Before trying out different models, a brief analysis on the principal components will be performed using principal component analysis [2]. PCA allows us to transform the raw features into a vector of uncorrelated variables where the first component accounting for the highest variance in the data and so on. By performing this analysis and calculating the cumulative variance ratio of all the feature we can visualize how much of the data-set cumulatively contribute to the total variance margin (normalized to 1).

The plots confirms our earlier concern that some of the features (especially last three) would highly dominate the

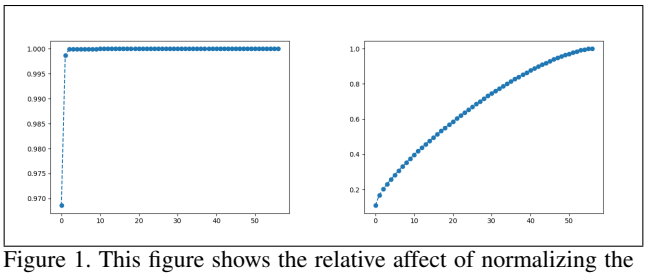

Figure 1. This figure shows the relative affect of normalizing the data on the principal component decomposition of the training dataset. The x-axis is the feature index and the y-axis is the cumulative variance

variance in the dataset because they are a continuous real number compared to all others that are normalized scores $[0, 1]$. To alleviate the affect, standardization was applied only on the training set and the second graph was obtained. The almost linear SHOW PROOF points out to the fact that when feature vectors are standardized, each component contributes roughly the same amount of variance. Thus we predict that dimensionality reduction might not improve the accuracy of our models. We will

2. Baseline Classifiers

This section evaluates the performance of various famous baseline classifiers using tuned parameters which depend upon the model being used.

2.1. Perceptron Model

To start with the simplest baseline model, we will use the perceptron model along with the pocket algorithm to measure our baseline accuracy.The single-layer perceptron is well suited for a binary classification task and has been around since the 1960s so it has been very well studied. Each trained hypothesis can be described as:

$$h(\vec{x}) = sign(\sum_{i=0}^{d} w_i \vec{x}_i) = sign(\mathbf{w}^T \mathbf{x})$$

Here $w_0 = -threshold$ for the classification boundary and $w_1, ..., w_{57}$ will be the weightings for each feature that will contribute to the overall total score. Because all our outputs are either 0 or 1 a slight modification to the hypothesis function will be needed:

$$h(\vec{x}) = \mathbb{I}(sign(\mathbf{w}^T \mathbf{x}) > 0)$$

This will now ensure that the hypothesis function now returns a 0 or 1 for any $\vec{x}$.

The perceptron algorithm was able to reach 89.98% accuracy using the default parameters. Given the very high accuracy just using a base-model we can argue that the data-set is well curated and we will not make much performance gains by doing feature transformation. The only

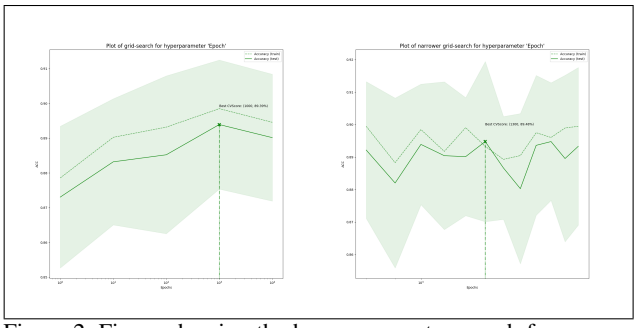

Figure 2. Figure showing the hyper-parameter search for perceptron algorithm

hyper-parameter we could tune for the perceptron algorithm was the number of complete passes or epochs through the training data and this improved the accuracy to 89.48%.

The search was performed using a GridSearchCV algorithm using cross-validation to test the performance of various hyper parameters. The general approach taken here involves testing out values in powers of 10 and then tuning it down to smaller intervals in the second pass. This will be used for the rest of the study. Figure 2 showcase the results of the hyper-parameter search in first and second pass.

2.2. Logistic Regression Model

2.2.1 Baseline

Logistic regression is another popular algorithm for classifying features into two or more classes. It is a type of linear model which associates probability (for the likelihood) to each output class (assuming each y_i is a Bernoulli distribution) given one or more input features. The hypothesis estimating this probability can be modeled as passing a linear combination of weights through a sigmoid function:

$$h(\vec{x}) = \theta(\mathbf{w}^T\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

Logistic regression has no close form solution but weights can be optimized using an iterative algorithm as long as the loss function is differentiable. Maximum likelihood estimation can be used to derive the piecewise loss function [3] which is equivalent to log-loss.

$$\ell(h, y) = \log(1 + e^{-\mathbf{w}^T\mathbf{x}}) - y\mathbf{w}^T\mathbf{x}$$

Results for the linear regression with default parameters were at 92%.

2.2.2 Regularization

We also tested both L1 and L2 type of regularization parameters which try to shift non-important co-efficients of the weights towards zero. They are defined as:

$$||\mathbf{w}||_1 < C \; and \; ||\mathbf{w}||_2^2 < C$$

Even after trying out different range of values for C, the overall accuracy still remained in the 92% range. Thus the model does not benefit from regularization and this can be inferred from earlier when we observed that all of the input features had a similar variance value during PCA. This means that any constraint that minimizes particular weights will actually cause the model to perform worse. While choosing hyper-parameters the model gave preference to L2 or elastic-net. Without looking closely at the data, we can say that most fields have 0 or small values and hence the dataset is sparse so doing it further wouldn't reap many benefits.

2.2.3 Quantile Transformation

We were able to see some significant improvements with quantile transformation as a pre-processing step before training the model. Instead of standardizing the data as usual, we transformed them into a uniform distribution (after CV scores showed preference to uniform distribution over normal distribution). This helps removes outliers and reduces the affect of limited repeated values [4]. The final accuracy after hyper-parameter optimization was 94.87% on test-data set.

3. Advance Classifiers

3.1. Gradient Boosting

In this section we will focus on advance models to train our dataset and evaluate their performances. We will start with gradient boosting which an algorithm from a family of ensemble methods. In simple terms, gradient boosting starts with a weak learner which improve itself by creating a new learner iteratively with the objective of reducing the error of the loss function [5].

Given a weak model $F_m(x)$ trying to predict y, each $F_{m+1}(x)$ is produced by adding a correction term $h(x)$ such that:

$$h(x) = y - F_m(x)$$

$$F_{m+1}(x) = F_m(x) + \gamma_m h_m(x) \; \textit{with} \; \gamma_m = \; \textit{step-size}$$

Given $h(x)$ is of a similar form for the gradient of least-squares-error, there is motivation to use this algorithm iteratively just like gradient descent. The general model for each gradient boosted weak learner is a single-class decision tree where each leaf node providing $P(y|\vec{x})$.

We first tried with directly using the model with default parameters and it gave 94.75% ACC. After trying with and

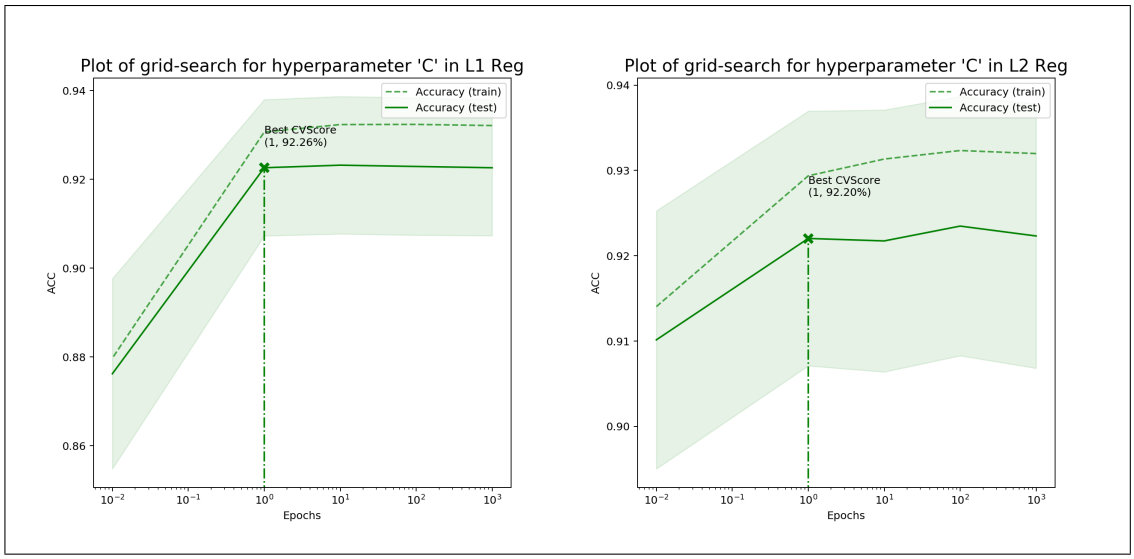

Figure 3. L1 regularization produces a smaller generalization error and also a more accurate result.

without standardization and quantization we concluded that this model is robust enough to be used with dataset that has unbalanced feature variance such as ours. This makes the algorithm flexible enough to be used in a variety of cases without requiring any modifications.

A thorough hyper-parameter search for this model result in an increase to accuracy to about 95.01%. It was observed that this model in particular takes a considerable time to inference (when weights are loaded from file). This is most likely due to the fact that a large number of individual classifiers make decisions in a sequential fashion before the final output is inferred.

3.2. Neural Network

Thanks to the advent of high-speed computation, neural networks have become the go-to model to solve any machine learning task. Their highly scalable nature means that they can be structured in a wide variety of ways to increase their complexity and thus their ability to learn non-linear features. The second advance algorithm that we tested was a simple neural network with an input layer, a hidden layer and an output layer.

3.2.1 Architecture

We started off with the simplest architecture of a single hidden layer and then iteratively adjusted hyper-parameters to improve our score. our baseline scores started around 92% and then using different activation function we were able to increase that to about 95%. Given the amount of hyper-parameters that can be optimized with deep-learning networks and the lack of time to run these networks on local machine, we decided to tweak the model manually, adding and subtracting layers and neurons until we get a good solution.

We also studied the effects of normalizing the activations and found in particular the lecun-normal initializer [6] which initializes weights in each layer in a special way allowed the activations to have a self-normalizing property and thus essentially improve performance. Furthermore the quantizer transformation also helped in improving this accuracy further.

Table 1. A summary of the top performance of each classifier.

Classifier	ACC_{CV}	ACC_TEST
Neural network	97.65% (+/- 0.82%)	95.40%
Gradient Boost	95.01% (+/- 0.41%)	95.40%
Logistic Regression	93.97% (+/- 0.29%)	94.79%
Perceptron	89.48% (+/- 2.47%)	

4. Results

Table 1 provides the best results of each classifier. In conclusion we have found that various models were quite suited for this dataset and that we can have significant performance gains in certain cases from a detailed and thorough hyper-parameter tuning. We also discovered that feature transformation or pre-processing plays an important role in this process. In general we can argue that the linear model was well suited for this task and for just a small performance drop we can get a model that trains and infer much faster than both of the advanced algorithms. In terms of the advance algorithms, the gradient boost method has the least variance thus it performs less over-fitting than neural networks by a small margin.

5. Appendix

5.1. Input

The input of each record has 57 dimensions. Looking at the description provided alongside the database. The first 48 features are frequency feature for the word represented by that column. It can be thought of the normalized frequency of that particular word in the email or in other words the probability of that word being present in the email. The next 6 features are probability of special character occurrences and finally the last three features are real numbers which represent the average, longest and total length of uninterrupted capital lettered words (respectively) in the email.

5.2. Disclaimer

I, Haaris Osman Mehmood, 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.

References

[1] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[2] "Principal component analysis with linear algebra," 2012. [Online]. Available: http://www.math.union.edu/~jaureguj/PCA.pdf

[3] "Knetml/knet-the-julia-dope." [Online]. Available: https://github.com/KnetML/Knet-the-Julia-dope/blob/master/chapter02_supervised-learning/section3-logistic-regression.ipynb

[4] "Quantile transformer." [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html

[5] C. Li, "A gentle introduction to gradient boosting." [Online]. Available: http://www.chengli.io/tutorials/gradient_boosting.pdf

[6] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *CoRR*, vol. abs/1706.02515, 2017. [Online]. Available: http://arxiv.org/abs/1706.02515