Machine Learning Coursework

Sze Tyng Lee
Imperial College London
sze.lee15@imperial.ac.uk

March 23, 2018

1 Introduction

The dataset chosen for this experiment is an email spambase data set [1]. Each email is represented by 57 input features, which are the frequencies of particular words or characters' occurrences, average and maximum lengths of uninterrupted sequences of capital letters and the total number of capital letters in the email.

This dataset is sparse; there are missing values for many of the features. The emails are classified as either 1 or 0, which represent spam and ham respectively. The dataset consists of 4601 points, and was split into 3680 training and 921 testing points (80% and 20% respectively).

1.1 Problem definition

The task of predicting if new emails are spam or ham is a binary classification problem, since the target is a discrete number with only two possible values. Classification given in the dataset is 1 for spam and 0 for ham. In order for us to formulate the problem, we will convert the 0's into -1's.

Input: $\boldsymbol{x}_1, \boldsymbol{x}_2, ...\boldsymbol{x}_n, where\ \boldsymbol{x} = x_1, x_2, ...x_{57}$
Output: $y_1, y_2, ...y_n, where\ y = \{-1, 1\}$
Target function: $f : x \rightarrow y$
Hypothesis: $h(\boldsymbol{x}) = \hat{y} = sign(\boldsymbol{w}^T\boldsymbol{x})$

Thus, the appropriate loss function to be used by the machine learning algorithm is the binary error. The goal of the problem is to maximise the probability of the event that the machine learning model correctly predicts new emails. Mathematically, this is equivalent to minimising the test error of the model. The test error is obtained when evaluating the model on unseen data that is not made available during the training period. Hence, empirical risk minimisation methods are employed during training in order to minimise the training error instead.

2 Methodology

Employing the empirical risk minimisation method alone without any regularisation would result in overfitting, where the model tries to fit the noise in the data. When that happens, the test error would not generalise well to the training error, resulting in poor performances. Regularisation methods such as the L1 and L2 methods should be used, with the parameters in the penalty terms be chosen using proper validation methods. The penalty terms are introduced in the loss function being minimised.

The constant multiplying the regularisation term is called the *lambda*, λ; it must be chosen carefully through a methodological approach. In this experiment, the λ term is selected using a 10-fold cross-validation approach. The cross-validation approach is chosen so that a small validation set can be used with the tradeoff of a longer training time - as the training has to be repeated 10 times for each parameter value being tried. This is not too big of a problem as the number of training points in this experiment is not that large. The best value of λ is the one with the lowest cross-validation error. The model using this value is then re-trained with the full training set (by putting the validation points back in), which should theoretically improve its performance since it was trained on more points.

3 Baseline predictors

A linear predictor suitable for this problem is the perceptron learning algorithm. It is a baseline model that updates the weight vector whenever a mistake is made. The algorithm minimises the training error through the updates which aim to decrease the error on every iteration.

It is clear that the algorithm will stop after some time if the data is linearly separable. If the data is not linearly separable, the perceptron will never converge so it must be forced to stop after a certain criteria is met. The training data was put through 5000 iterations and could not achieve a training error of zero - it is not linearly separable. Thus, a stopping criterion was set such that the maximum number of epochs the perceptron is allowed to go through is 5000.

To prevent overfitting, the LASSO or L1 method of regularisation was implemented, where the penalty term is the L1 norm of the weight vector. The LASSO regularisation was chosen over ridge regression because it has a feature selection property which greatly helps this dataset as there are many missing values in the input features. The λ parameter in the regularisation term is renamed as *alpha* in the library of choice [2], and was chosen using a 10-fold cross-validation method. The value 0.0001 of *alpha* corresponds to the highest cross-validation accuracy of 0.9046, and is chosen as the final parameter for the model.

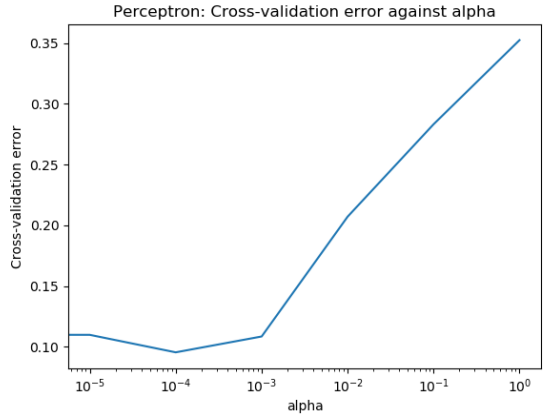

Figure 1: Cross-validation error for LASSO regularisation on perceptron

α	0.00001	0.0001	0.001	0.01	0.1	1
L1	0.8902	0.9046	0.8916	0.7929	0.7168	0.6476

Table 1: 10-fold cross-validation accuracy scores for LASSO regularisation on the perceptron

Having selected the parameters, the perceptron was retrained with the entire set of training data (the 80% of the original data available, split earlier as explained in the introduction), in order to get the best hypothesis possible.

Cross-validation score	**Training score**	**Test score**
0.9046	0.8940	0.8806

Table 2: Accuracies using the chosen perceptron model

It is found that the training error was in fact worse than the cross-validation error. Given that the model of choice is the perceptron algorithm and that the data is not linearly separable, this discrepancy could be due to the fact that in the final iteration, the perceptron was not at the boundary with the lowest training error. The algorithm allows for changes in the random state of the perceptron, which is the random way that the model shuffles the data after each epoch. A simple way to improve the performance of the perceptron and reduce the ambiguity of the training error is by implementing the pocket algorithm discussed in the lectures. Otherwise, a different sequence of shuffling could result in a different final boundary, as shown below:

Seed	0	1	2	3
Training score	0.8940	0.8984	0.9242	0.9125

Table 3: Training scores of the same perceptron model with different seeds for the random number generator

4 Advanced algorithms

4.1 Proposal

While the linear model has been shown to deliver a good method of classifying emails, there are better models with more robust classifiers than the simple perceptron. The perceptron updates its model whenever a mistake in classification has been made; it is an online iterative process

where the classifier result depends on the starting point of the algorithm (i.e. the initial weight vector) and also the way the data is shuffled after each iteration.

A more advanced algorithm that can be used is the Support Vector Machine (SVM). An SVM aims to maximise the margin between the hyperplane and the nearest data points to it, which are called the support vectors. The difference between the SVM and the perceptron is that the perceptron only aims to correctly classify the data points, while the SVM does that in addition to choosing the classifier with the widest margin, which leads to better performances on out-of-sample data points.

To avoid overfitting, the regularisation penalty and its constant, C are introduced in the SVM. The penalty symbolises the amount of violations (i.e. the data points violating the margin) allowed by the hyperplane. A larger C is similar to decreasing the amount of regularisation and allows less violations - the classification is more accurate at the expense of a smaller margin and more support vectors.

4.2 Implementation

SVMs use the kernel trick to transform the input features of the dataset into a higher dimension in order for it to be able to construct a hyperplane boundary between the two classes. It is found by inspecting the training data that some of the input features are in a numerical range much larger than the rest. This could cause several problems in kernelling [3], which is essentially computing the inner products of the attributes and would face difficulty with these large values. The larger values could also dominate the features with smaller values, leading to a worse performance overall.

Hence, it is important to scale the data before training the model; the test data underwent the same scaling, using parameters defined only by the training set. The many benefits of scaling the dataset were found during implementation as the SVM training was run on both scaled and unscaled data for comparison. The results displayed in the following sections are based on scaled data unless specified otherwise, while the full results are available in the appendix.

The kernel used in this experiment is the radial basis function (RBF), due to its ability to model a bigger number of functions while only having one important parameter to tune: *gamma*, γ. In comparison, the polynomial kernel requires tuning both the γ and the degree of the polynomial. This is all in addition to finding an appropriate value of C to prevent overfitting, a parameter required regardless of the type of kernel chosen.

The γ coefficient determines the level of influence of a single point. By extension, it defines how much influence any of the support vector has on the rest of the data points. The larger the value of γ, the less the effect of a single support vector. Thus, the number of support vector increases, which could classify the points more accurately but runs the risk of overfitting.

The parameters γ and C clearly have a large influence on the performance of the SVM model. They were chosen using the 10-fold cross-validation method. The best cross-validation score is the model with a C value of 10^5 and γ of 10^{-4}.

C \ γ	10^{-6}	10^{-5}	10^{-4}	0.001	0.01
10^7	0.9163	0.9318	0.9212	0.9033	0.8897
10^6	0.9215	0.9351	0.9315	0.9168	0.8981
10^5	0.9280	0.9342	0.9375	0.9247	0.8989
10^4	0.9223	0.9326	0.9345	0.9315	0.9155
10^3	0.9062	0.9242	0.9329	0.9370	0.9261

Table 4: 10-fold cross-validation accuracy scores for γ and C

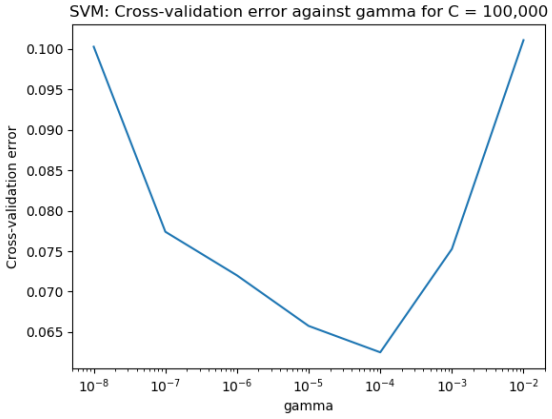

Figure 2: Cross-validation error for different values of the kernel coefficient for SVM when C is fixed

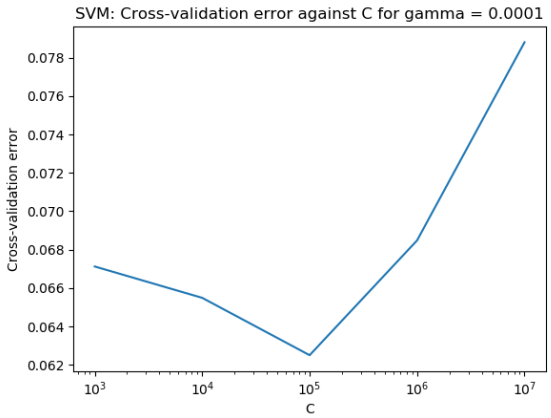

Figure 3: Cross-validation error for different amounts of regularisation on SVM when γ is fixed

4.3 Performance

The SVM model with the best-performing parameters was re-trained on the full training set before it was evaluated on the test set. The bound on the expected test error depends on the proportion of support vectors to training points, regardless of the type of transformation used on the features [4]. The number of support vectors used in this model is 607 out of 3680 training points, so the test score is expected to be at least 0.835.

$$\mathbb{E}[R(h)] \leq \frac{\mathbb{E}[\#\ support\ vectors]}{N-1} \quad (1)$$

$$\leq 0.1650 \quad (2)$$

$$\mathbb{E}[Test\ score] \geq 0.835 \quad (3)$$

Cross-validation score	Training score	Test score
0.9375	0.9611	0.9381

Table 5: Accuracies using the chosen SVM model

As mentioned in the previous section, the performance of the SVM was vastly improved after scaling the dataset. One of the most drastic changes was in the training time of the model. The scaled data was found to be almost 40 times faster to train than the unscaled data.

	Scaled data	Unscaled data
Training time(s)	2.79	109.71

Table 6: Training times for SVM

5 Conclusion

Using α = 0.0001 for the LASSO regularisation in the perceptron, and C = 10^5 and $\gamma = 10^{-4}$ in the SVM, the training performances are as follows:

	Perceptron	SVM
Training accuracy	89%	96%
Training times(s)	6.23	2.79

Table 7: Performance of different prediction models during training

The SVM clearly outperforms the perceptron in all the metrics being evaluated. The final proposed solution to the machine learning problem presented in this report is to apply the Support Vector Machine algorithm using the parameters stated. When using the SVM, the training data must be scaled and the same scaling parameters must be used again on the test data before making the predictions. In doing so, a 94% accuracy of correct predictions of spam or ham emails was measured when the model was applied on the test data set aside in the beginning of the experiment.

Pledge

I, Sze Tyng Lee, 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] Mark Hopkins et al. *Spambase Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/spambase. (accessed: 05.03.2018).

[2] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[3] Chih-Chung Chang Chih-Wei Hsu and Chih-Jen Lin. "A Practical Guide to Support Vector Classification". In: (2007). URL: https://www.csie.ntu.edu.tw/~r95162/guide.pdf.

[4] Yaser S. Abu-Mostafa. *Learning From Data, [Lecture 14: Support Vector Machines]*. URL: http://work.caltech.edu/slides/slides14.pdf. (accessed: 20.03.2018).

Appendix

C\Gamma	1e-08	1e-07	0.000001	0.00001	0.0001	0.001
10000000	0.914397	0.928257	0.932602	0.914394	0.893753	0.860611
1000 000	0.917119	0.932058	**0.938855**	0.924722	0.908963	0.866043
100000	0.883692	0.922279	0.934780	0.933695	0.912769	0.875008
10000	0.775004	0.886137	0.926083	0.931519	0.917659	0.886146
1000	0.735058	0.785334	0.89048913	0.92065217	0.91902174	0.89592391
100	0.711684	0.732884	0.78423913	0.87961957	0.90842391	0.90081522
10	0.679075	0.719831	0.73423913	0.76222826	0.85923913	0.87690217
1	0.658423	0.680428	0.705972	0.727993	0.757069	0.809503

Figure 4: 10-fold cross-validation accuracy scores for SVM on unscaled data

Training score	Cross-validation score	Test score
0.9548913043478261	0.938855	0.9163952225841476

Figure 5: Accuracy of the SVM with $\gamma = 10^{-6}$ and $C = 10^{6}$ on unscaled data

C\gamma	1e-8	1e-7	1e-6	1e-5	0.0001	0.001	0.01
10000000	0.907875	0.918201	0.916299	0.931786	0.921196	0.903256	0.889673
1000000	0.914939	0.913580	0.921465	0.935055	0.931519	0.916844	0.898095
100000	0.899722	0.922594	0.927988	0.934235	**0.937498**	0.924723	0.898905
10000	0.811679	0.905430	0.922277	0.932606	0.934510	0.931517	0.915482
1000	0.608424	0.814669	0.906246	0.924179	0.932877	0.936954	0.926079

Figure 6: Full 10-fold cross-validation accuracy scores for SVM on scaled data