SMS Spam Detection using Machine Learning (Python)

Mariam Sarfraz
Imperial College London, EEE Department
ms5715@ic.ac.uk

Abstract

SMS Spam Detection is one of the most prominent Machine Learning applications. Using supervised learning for binary (0/ham, 1/spam) test classification, two baseline predictors have been trained and used: Naïve Bayes Algorithm and Perceptron Learning Algorithm. The more advanced algorithms then proposed are SVMs (Support Vector Machines) and Neural Networks.

1. Introduction

SMS (Short Message Service) is one of the most popular means of communications. With its growing popularity and the general mass's dependency on mobile phones, the problem of receiving spam SMS is a prominent problem. *Spam* messages are defined as unwanted text messages from dodgy firms/individuals aiming to grab attention. The solution to this is using spam detection systems using Machine Learning techniques. In this report, two baseline classifiers are evaluated and their performance is analyzed followed by two advanced algorithms which aim to give us better results. The test data set was 20% of the whole data given and the training data set was 80%. Scikit-learn was the library mainly used for this module.[i]

2. Feature Extraction

The biggest challenge in SMS spam detection is converting the words in these SMS into features for our classifiers. The feature extraction process is divided into three steps:[ii]

2.1. Manual Feature Extraction using characters and capital letters

First, 18 features are extracted manually. The following attributes of SMS are used:

i) Capital letters (one letter must be followed by another capital letter to make it more efficient)
ii) Exclamation marks (!) (one exclamation mark must be followed by another exclamation mark)
iii) Question marks (?) (one question mark must be followed by another question mark)
iv) Currency signs (dollar, euro, Britain pound)
v) Star marks (*)
vi) Percentage marks (%)

Three attributes are extracted per attribute listed above:

i) Total number of attribute occurrences, relative to the total length of the SMS
ii) Average number of attribute occurrences, relative to the total length of the SMS
iii) Longest length of attribute occurrences, relative to the total length of the SMS

2.2. Preprocessing text

In order to extract other features like word count, preprocessing of data is required. We do three steps in preprocessing:

i) Using a tokenizer to split the SMS "sentences" into list of tokens (words) by removing spaces and punctuation marks. We do not need punctuation marks anymore as we have already extracted features related to them.
ii) Using a lemmatizer to change similar forms of the same word into one word (e.g. go, going, gone into go). This would help in word count.
iii) Making all words lowercase so that they can be considered the same (e.g. ORGAN and organ). We do not need our data to be case sensitive as we have already extracted features regarding capital letters.

2.3. Word count feature extraction

As the last step, raw text data is converted into numbers using CountVectorizer()[iii]. This creates a vocabulary/dictionary which includes all words from our entire data set. It then gives a count of the occurrence of each word (representing a feature/column) in our dictionary per SMS (per row).

As our data set has a size of 4459 SMS, not all word counts in the SMS can be used as features. Using the rule of thumb:

$$N \geq 10d_{vc}$$

where N is the size of data set and d(vc) is the VC dimensions. This shows us that at max, 426 (444-18) word counts can be utilized as features.
The features are reduced to 400 by using 2 parameters in the CountVectorizer():

i) stop_words: words such as "the", "or" and "a" do not contribute much as they do not give out information, being present in both spam and ham SMS. Such words are removed using stop_words. We remove all English stop words.
ii) min_df: all words that appear in less than the integer value min_df is assigned to (in our case 17) are ignored. This parameter helps get rid of infrequent words which do not give out much information as they occur less frequently.

These 400-word count features are then combined with the features we extracted in step 1 to give us a feature matrix that will be used for testing.

3. Baseline Classifiers

The baseline classifiers chosen are perceptron learning algorithm and Naïve Bayes Theorem.

3.1. Loss Function

For both baseline classifiers, a binary error loss function was implemented using the penalty table below:

	Actual Spam (1)	*Actual Ham (0)*
Predicted Spam (1)	0	10
Predicted Ham (0)	1	0

Figure 1

Using this penalty function, the number of errors were counted. If an actual spam was predicted as ham, the error count only increased by one. However, if an actual ham was predicted as spam, the error count increased by 10. This is because a Ham SMS could contain very useful information and that we wouldn't want to classify as spam. However, if we get a spam SMS in our text folder, we could simply ignore/delete it. The total error was then given by:

$$Error\ (In\ Sample) = \frac{error\ count\ (from\ training\ dataset)}{length\ of\ training\ data\ set}$$

$$Error\ (Out\ of\ Sample) = \frac{error\ count\ (from\ test\ dataset)}{length\ of\ test\ data\ set}$$

3.2. Cross-Validation

In all algorithms analyzed in the report, 10-fold cross validation is used as it is arbitrary and highly recommended.

3.3. Single Layer Perceptron Learning Algorithm (PLA)

PLA is a standard baseline algorithm used for binary classification. It maps the input to the output by calculating a weight vector. One of the parameters, tol, is kept to its default value 'None' as this implements the pocket algorithm which is used for linearly inseparable data. Below are the results obtained by keeping max_iter (parameter: epochs) as 10, 100 and 1000.

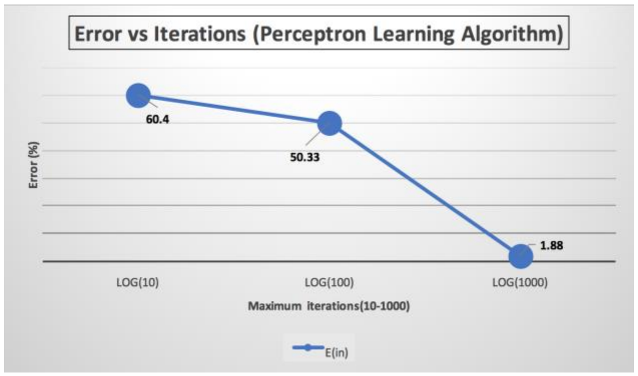

Figure 2

The results show us that the best training error (1.88%) is obtained using 1000 epochs. Using these optimized parameters, we get our test/true error to be 26.01%.

3.4. Multinomial Naïve Bayes Algorithm

Naïve Bayes Algorithm is a simple yet highly effective classifier as it predicts the probability that the input belongs to a class (in our case, spam or ham) depending on each of the feature value probabilities. The aspect that makes this algorithm "naïve" is that it is presumed that the occurrence of a certain feature is independent of the occurrence of the other features. It is suitable for high dimensional training sets. The algorithm is based on Bayes Theorem:

$$P(class|features) = \frac{P(features|class)P(class)}{P(features)}$$

As our data set is multinomially distributed, we use Multinomial Naïve Bayes Algorithm.

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i {p_{ki}}^{x_i}$$

where **x** is the feature vector, C(k) is the class, p(ki) is the probability that event i occurs (in the class).

In multinomial Naïve Bayes, the key parameter is alpha, which is a smoothing hyperparameter. Optimum value for alpha is determined using grid search over reasonable values of alpha. GridSearchCV() goes through an exhaustive search process over a range of parameter values given to it to give us the optimal parameter values. The results for different alpha values are shown below:

Alpha	E(in) /%	Accuracy
1e-10	23.84	0.963
1e-5	25.63	0.961
1e-3	27.88	0.960
0.01	33.46	0.955
1	35.01	0.954

Figure 3

The results show that for alpha = 1e-10 (Lidstone smoothing), the performance of the classifier is the best, giving us the best accuracy and the lowest training error (23.84%). Using these optimized parameters, the test/true error obtained is 26.82%.

3.5. Conclusion

Looking at the results, we can conclude that both baseline classifiers have approximately the same performance level. In the next section, advanced algorithms will be selected and evaluated to increase the accuracy and reduce the true error of our spam detector.

4. Advanced Algorithms

Treating Naïve Bayes Algorithm and Perceptron Learning Algorithm as our baseline classifiers, we will now propose two advanced algorithms suitable for SMS spam detection: Neural Networks (Multi-layer perceptron learning algorithm) and SVMs (Support Vector Machines).

4.1. Neural Networks

Neural Networks are powerful and highly efficient learning algorithms. They are a suitable choice for spam detection as they are used to translate features and create non-linear, "soft" decision boundaries. They are mainly used for classification problems. They "soften" the threshold, followed by the application of the all-general gradient descent to find an effective solution. Once a solution is formed, we "harden" the threshold to get our final solution.

4.1.1 Loss Function

As the output of the neural network is a "soft threshold" i.e. the probability value between 0 and 1, the loss function of our choice will be the Cross-Entropy loss function:

$$H_{y'}(y) := -\sum_i y'_i \log(y_i)$$

y_i is the predicted probability distribution of the class i (?) and y'_i is the true probability for that class.

The error increases with the predicted probability going further away from the actual result/class.

4.1.2 Parameter Selection

There are four main features in MLP (multi-layer perceptron classifier).

i) Hidden_Layer_Sizes: this is the number of hidden layers used in the neural network. Using GridSearchCV(), a comparison was made using parameter values 50, 100 and 200. The optimal value for this parameter was 100 (which is the default value as well).

ii) Activation: is the activation function used for the hidden layers. We choose "relu" as it does not face the problem of gradient vanishing like tanh or sigmoid function.

iii) Solver: is the method used for solving for the weights. We use "adam" as it is ideal for large data sets like ours.

iv) Alpha: alpha is the regularization term for our Neural Network. We use GridSearchCV() to solve for the optimal alpha value. The optimal alpha value obtained is: 0.001

4.1.3 Performance Evaluation

Using these parameter values and the cross-entropy loss function, we obtain the training error as 0.65, accuracy as 0.982 and finally the test/true error as 4.72.

4.2. Support Vector Machines

Support Vector Machines are a suitable advanced algorithm for spam detection as it is robust and handles large feature spaces. SVM separates these features in a very high dimensional space.

4.2.1 Loss Function

The loss function suitable for SVMs is Hinge Loss Function.

$$\ell(y) = \max(0, 1 - t \cdot y)$$

where t is the actual result (-1, +1) and y is the predicted result. We can observe that cumulated the hinge loss is an upper bound for the number of misclassified points. [iv]

4.2.2 Parameter Selection

i) Choice of Kernel:

We use the Linear SVC for our spam detector as solving the optimization problem using linear kernel is much faster. As we have a large number of features, it is not necessary to map data to a high-dimensional space. Linear kernel is, hence, good enough for our case. [v]

ii) Choice of C:

Apart from the choice of kernel, there is one other crucial factor in parameter selection: C. C is the penalty parameter of the error term. Using GridSearchCV(), we obtain the optimal value of C from the set [0.001, 0.01, 0.1, 1, 10] to be 0.1.

4.2.3 Performance Evaluation

Using the optimal parameter values, we obtain the following results:

Training error is 2.69, accuracy is 0.979 and test/true error is 2.87.

4.3. Conclusion

Comparing the results between our advanced algorithm, the Support Vector Machines have better performance with only a true error of 2.87 compared to the neural network error of 4.72.

5. <u>Over-all Conclusion</u>

Looking at the results, we can conclude that the performance (best to worst) for our algorithms is as follows:

1. Support Vector Machines using Linear Kernel
2. Neural Networks
3. Perceptron (Pocket) Learning Algorithm
3. Naïve Bayes Theorem

6. <u>Pledge</u>

I, Mariam Sarfraz, 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.

7. <u>References</u>

[i] http://scikit-learn.org/stable/

[ii] https://cambridgespark.com/content/tutorials/implementing-your-own-spam-filter/index.html

[iii] https://pythonprogramminglanguage.com/bag-of-words/

[iv] https://en.wikipedia.org/wiki/Hinge_loss

[v] http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html