**Microsoft Word Author Guidelines**

|  |  |
| --- | --- |
| Koral HassanImperial College LondonKensington, London SW7 2AZkbh15@ic.ac.uk |  |

**Abstract**

*This paper discusses my process of solving a particular machine learning project. The project of choice was a predictor for detecting SMS text messages that were perceived to be spam, the definition of spam in this context being “irrelevant or unsolicited messages, typically sent to a large number of users, for the purposes of advertising, phishing, spreading malware, Et cetera.*

# Introduction

For solving this problem, I had access to a data set of 5574 data points labeled as “spam” or “ham”. The label “ham” denotes that a message should not be considered spam, this term will be used throughout the report for this purpose. The data set consisted of real messages that were sent to people. The only information contained in each data point was the actual text contents of each SMS message. This was a binary classification problem.

## Submission

I, Koral Hassan, 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.

# Discussing design choices

## Use cases

The predictor’s likely use case is for filtering spam messages out. It might also be used for warning the recipient of the message’s harmful intentions, but currently this not a mainstream feature in phones.

Most phone users care dearly about reliably receiving the text messages sent to them, since nowadays text messages are frequently used for conveying important or urgent information. Moreover, most phone users are technologically literate enough to recognise spam and ignore it, or at the very least to not trust messages from strangers.

This brings us to the deduction that misclassifying SMS messages as spam causes a greater inconvenience than failing to identify an actual spam SMS messages. This should be accounted for by false positives (i.e. misclassifying “ham” as “spam”) contributing more to our loss function than false negatives (i.e. misclassifying “spam” as “ham”).

Figure 1 shows the contribution of all prediction vs label scenarios. The ratio of 10:1 was chosen from personal preferences (i.e. tolerance to spam vs tolerance to missing messages). This loss function should ideally filter only one ham message out for every 10 spam messages it allows through.

|  |  |  |
| --- | --- | --- |
|  | **Label: ham** | **Label: spam** |
| **Prediction: ham** | 0 | 0.1 |
| **Prediction: spam** | 1 | 0 |

*Figure 1*

## Language and Libraries

All practical activities (including gathering data for demonstration and justification in the report) were performed using the programming language Python. The libraries used included sklearn, nltk, re and scipy.

## Train-test split

The best method for getting an estimation our out-of-sample error is to leave out some of the data set for the purpose of emulating previously unencountered situations. Reserving more of the data results in higher confidence in our out-of-sample performance, but reduces our number of data points for training, resulting in a poorer model. While there is no perfect answer to how much of the data set should be reserved, the industry standard is 20-30%. This range has come to be favoured through trial-and-error.

Our data set of 5574 messages is relatively small for the purposes of machine learning applications, so I chose to keep as much data as reasonably acceptable, reserving only 21% of the data for testing.

This method is called splitting the data set. From here onwards, I shall refer to the data set we use for training as the training data/set and the reserved set as the testing data/set.

It is strictly forbidden to use any information gained from the test set to improve our models, since this clearly violates the concept of previously unencountered data.

## Feature extraction

All mainstream machine learning algorithms require numbers as inputs since their operation is mathematical in nature. In all supervised machine learning projects, you are given a data set you can use for training and testing. Some problems lend themselves well to machine learning because the real-world properties you need to consider are either intuitively quantifiable or are literal numbers to begin with. The data set available for this project consisted entirely of text, and there was no single obvious way of selecting the features to be used.

After careful consideration the following methods were used to transform the text:

1. 3 features were extracted by;
	* counting the number of times each of the following symbols appeared:
		+ “#”,
		+ “%”,
		+ “$” or “£” or “€”.
	* for each criterion, calculating the following as a percentage of the length of the entire SMS;
		+ total number of occurrences.
2. 9 features were extracted by;
	* counting the number of times each of the following criteria were satisfied:
		+ 2 or more consecutive capital letters,
		+ 2 or more consecutive exclamation marks,
		+ 2 or more consecutive question marks.
	* for each criterion, calculating the following as a percentage of the number of characters in the entire SMS;
		+ total number of occurrences,
		+ average length of occurrences,
		+ length of the longest occurrence.
3. The remaining ~405 features were extracted by;
	* making all characters lowercase.
	* tokenising the message (i.e. splitting messages into a list of words consisting of alphanumeric characters).
	* lemmatising the list of words (i.e. simplifying words down to their root words and getting rid of plural forms).
	* making a dictionary containing every word that;
		+ makes up at least 0.0035% of our lemmatised training set (note that you cannot include any of the extra words in the test set since this leaks information about the test set to our model),
		+ is not an English stop word (e.g. “and”, “or”, “but”, Et cetera).
	* counting the number of times each word in the dictionary appears in the message.

## Motivation for feature selection

When our features, we are limited by the fact that having a higher number of features per training data point results in a higher probability of overfitting. The empirical industry standard is to try and have 10 times as many training data points as our features. In practice increasing the number of data points is difficult so we limit our number of features. We have 4403 training data points. We should have at most 440 features.

The features were selected with the following properties of ham and spam messages in mind:

1. Spam messages are often sent by scammers trying to convince people that they won a lot of money through some unlikely event or that they are about to lose a lot of money. They also often contain fake reference numbers. Taking this into account, it makes sense to search for symbols related to money, probability and serial numbers.
2. Advertisements try their best to grab a reader’s attention. This means their messages are filled with emotion and excitement. This is why they can be expected to contain lots of capital letters with repeated exclamation and question marks for emphasis. Matches are only made with 2 or more consecutive matches to ignore the contributions of normal capitalisation and punctuation adhering to grammar rules.
3. The dictionary is a catch-all solution for words that distinguish spam from ham that I as the designer of the model might fail to recognize. It is a system that simply looks for common features in messages.

## Baseline predictors

Baseline predictors are simpler models. Although they are sufficient for a wide range of problems, they are usually expected to not give us an optimal solution. Their main purpose is to give us a benchmark to compare the results of more advanced models against.

1. Random guessing:

Our primary baseline above all is what you would get by predicting completely at random. It is mathematically trivial to prove that in a binary classification problem, this will give an out-of-sample error of 50%.

1. Zero rule:

There is a second theoretical model we can use without implementing because of its simplicity. It performs better than the random guesser. It simply predicts the majority class in your dataset. In our case, this would equate to always predicting “ham”. Roughly 80% of the data set is ham. Assuming this generalises, this model would give an out-of-sample error of 20%. For any model to be worthwhile it must surpass the performance of the zero rule model.

1. Single layer perceptron:

My first implementation was the perceptron model. Figure 2 shows my results.

|  |  |  |
| --- | --- | --- |
|  | **In-sample** | **Out-of-sample** |
| **Data points** | 4403 | 1171 |
| **No. of errors** | 413 | 118 |
| **Error percentage** | 9.38 | 10.1 |
| **Loss function** | 293 | 82 |

*Figure 2*

1. Multinomial naïve bayes:

Figure 3 shows my results with the multinomial naïve bayes model.

|  |  |  |
| --- | --- | --- |
|  | **In-sample** | **Out-of-sample** |
| **Data points** | 4403 | 1171 |
| **No. of errors** | 198 | 53 |
| **Error percentage** | 4.50 | 4.53 |
| **Loss function** | 159 | 47 |

*Figure 3*

## Discussion of advanced algorithms

For advanced models, we will try using support vector machines and neural networks.

## Implementing advanced algorithms

We will use 10-fold cross validation in order to leverage our training set as much as possible but also get reliable parameter values.

We will use L1 regularisation to use even less features than we used for our baseline models, to be certain that our model will generalise well out-of-sample.

# Conclusion

Overall SVC seems to be the best solution to the problem but multinomial naïve bayes performs acceptably with much less computational cost.