Spam Filtering Techniques for Short Message Service

Adrien Besson and Dimitris Perdios

***Abstract*—We study various short message service spam filtering techniques based on a Kaggle dataset composed of 5572 messages, whose 4825 are legitimate and 747 are spam. The Bag-of-Words models followed by term-frequency-inverse-document-frequency transformation is employed for feature extraction. Several state-of-the-art classifiers are compared, e.g. logistic regression, regularized logistic regression, linear and kernel support vector machine (SVM), k-nearest neighbours, multinomial Bayes, decision trees, random forests, AdaBoost and neural networks, where the best hyper-parameters are identified using 10-fold cross validation. We demonstrate that all the classifiers perform remarkably well in terms of misclassification error and that even simple linear methods, such as logistic regression leads to less than 2 % of misclassification error. We study two reseampling methods that can be used to counter the class imbalance present in the training set, e.g. downsampling of the majority class and upsampling of the minority class. We show that both lead to an increase of the sensitivity at the cost of a lower specificity. Online learning strategies are finally investigated, where the algorithms sequentially update with a new batch of messages, mimicking a more realistic example. The supporting code is avilable at https://github.com/dperdios/adaptation-and-learning-project.**

***Index Terms*—Spam Filtering, Classification, Machine Learning**

I. Introduction and Data Exploration

Short Message Servive (SMS) is one of the most popular telecommunication service with approximately 15 millions SMS sent each minute around the world in 2017[1].

Spam can be described as unwanted messages sent in bulk to a group of recipients. SMS spamming has become a major nuisance to mobile users because of their intrusive nature and their waste of money, network bandwidth and time.

However, SMS spam filtering techniques are still at a relatively early stage due to the limited amount of publicly available datasets [1]. Most of the existing methods inherit from email spam filtering techniques which do not always perform well on SMS spam [2].

In this work, we are interested in addressing the problem of spam filtering by designing a feature extraction process and comparing various state-of-the-art classifiers, e.g. k-nearest-neighbours classifier, logistic regression, multinomial Bayes classifier, support vector machines, decision trees, random forests, neural networks and AdaBoost.

The study is based on the "SMS Spam Collection Dataset" available on Kaggle[2]. The dataset is composed of 5572 SMS tagged according to being *ham* (legitimate) or *spam*. It contains 4825 ham messages and 747 spam messages representing 86 % and 14 % of the dataset respectively. Thus, the problem is quite imbalanced which may have impact on the sensitivity and specificity of the classifiers, as discussed in Section IV.

[1]https://www.domo.com/learn/data-never-sleeps-5

[2]https://www.kaggle.com/uciml/sms-spam-collection-dataset

In a first data exploratory step, we display a bar plot of the 20 most frequent words in the dataset in Figure 1. It can be noticed they are all commonly used english words, usually called stop words.

The main idea behind SMS filtering relies on the fact that spam and ham messages are composed of different words (or groups of words). More precisely, some words are very likely to occur more frequently in ham messages than in spam messages and vice-versa. To illustrate such a fact, Figs. 2(a) and 2(b) display word clouds for ham and spam messages respectively, where the most frequent words appear the biggest.

Interestingly, it can be seen that most frequent words in spam messages are related to money ("free", "account", "winner", "win", "subscription", "reward") and urgency ("urgence", "last", "week"), which must be familiar to anybody that have already received spam messages. In ham messages, the words are the usual ones used in discussion like "anything", "home", "go", "wait", "remember" and "joke". We also notice the presence of onomatopoeia and slang words that are characteristics of text messages like "rofl", "dat", "hahaha", "lol" and "yup".

The above mentioned plots are informative in terms of two necessary preprocessing steps:

- Convert the words to lowercase;
- Remove the most frequent english words;

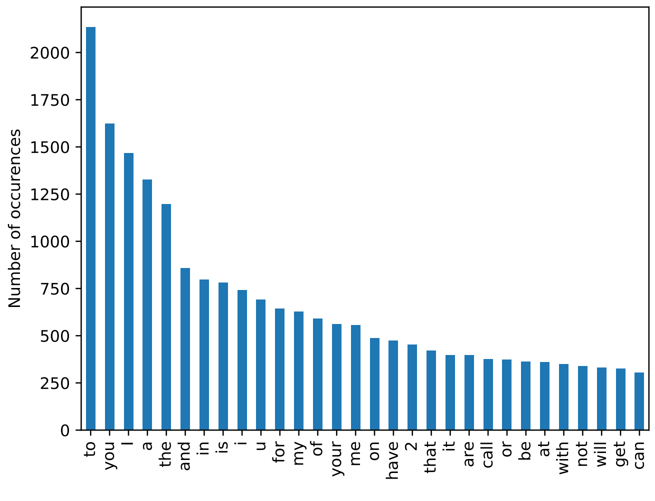
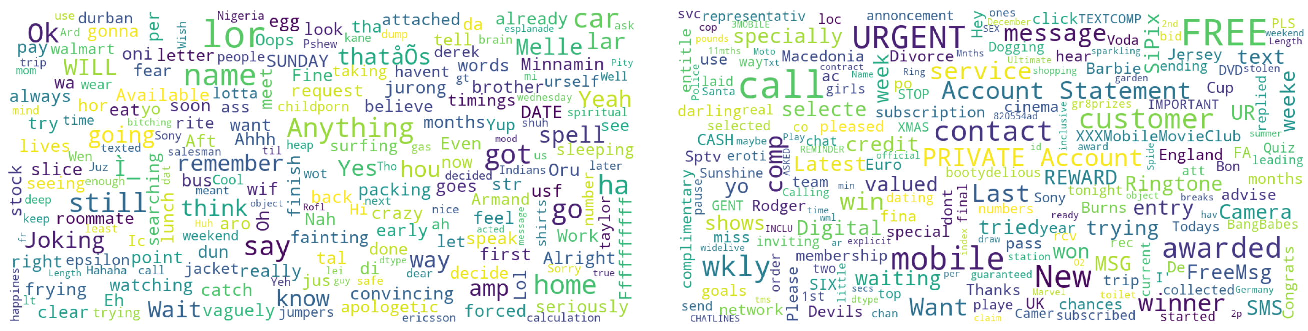


Fig. 1. 20 most frequent words in the dataset.

The remainder of the paper is organized as follows. Section II details the feature extraction step. Section III proposes a



(a) Ham (b) Spam

Fig. 2. Wordcloud of words in (a)-ham messages and (b)-spam messages. The bigger the words, the more frequent.

comparative study of the classifiers based on the misclassification error and Section IV describes two methods to increase the sensitivity and counter the class imbalance. Section V suggests an online learning strategy for classification and concluding remarks are given in Section VI.

II. Feature Extraction: From Messages to Word Frequencies

A. The Bag-of-Words Representation

It is evident that a message, considered as a sequence of symbols, cannot be used directly as input of a machine learning algorithm. Indeed, operations inside such algorithms require algebraic quantities, i.e. real- or complex-valued vectors in a given dimensional space.

The Bag-of-Words (BoW) representation [3], also called vector space model, is a well known representation in natural language processing, which describes the occurrence of words within a given document.

Formally, we consider a document composed of D words indexed as d_i, $i = 1, \dots, D$. We also consider a known vocabulary V which contains M reference words, where M can be very large. From the vocabulary, we build the representation vector $\boldsymbol{x} \in \mathbb{R}^M$ associated with the document as follows:

$$\boldsymbol{x} = \sum_{i=1}^{D} \boldsymbol{x}_{d_i}, \tag{1}$$

where $\boldsymbol{x}_{d_i} \in \mathbb{R}^M$ is a one-hot vector in which all entries are zero except the single entry corresponding to the location of the word d_i in the vocabulary.

Thus, given a dataset of N documents, we represent a matrix of occurrences $\boldsymbol{X} \in \mathbb{R}^{N\times M}$, where each row contains the representation vector associated with a given document.

In the specific case of our dataset of SMS, we build the vocabulary based on all the messages of the considered training set. In order to avoid duplicates and mismatches, we preprocess the dataset by removing all the capital letters. In addition, we remove the stop-words that are uninformative. These procedures result in a vocabulary of 7495 words.

B. Normalization of the BoW Representation: The tf-idf Weighting

It is preferable to work with frequencies rather than occurrences. It is also very important have normalized features since methods based on distances such as k-NN classifiers may be biased with unnormalized feature vectors.

A common way to perform such a task is based on the term-frequency (tf) inverse-document-frequency (idf) weighting [4], where the term frequency is calculated for the document n as follows:

$$tf_n = \frac{1}{D_n}, \tag{2}$$

where D_n designates the number of words. The tf weighting amounts to transforming occurrences in frequencies inside each document.

It can be seen that the tf weighting does not hold any information to assess relevancy on a message. Indeed, words that appear very often among the messages have little or no discriminating power in determining whether a message is a spam or not. To address this problem, we introduce the document frequency df_{d_i} of a term d_i as the number of documents in the collection that contain the term d_i. The idf is then defined as follows:

$$idf_{d_i} = \log \frac{N}{df_{d_i}}, \tag{3}$$

from which we can deduce the tf-idf weighting coefficient for term d_i in document n as follows:

$$T_{n,d_i} = tf_n \times idf_{d_i}. \tag{4}$$

The matrix of occurrences weighted by the tf-idf coefficient is used as input of the proposed classifiers. The feature extraction process is implemented on Python using scikit-learn feature extraction package.

III. Comparison of Several Classifiers

A. Considered Classifiers

The goal of the project is to design a classifier $c : \mathbb{R}^M \mapsto \{0, 1\}$, which takes a feature vector $\boldsymbol{h} \in \mathbb{R}^M$ containing the normalized word frequencies of a given message and assign it to either *ham* (class 0) or *spam* (class 1).

We implement the following statistical learning methods to predict the class of a message:

- **k-nearest neighbors (kNN):** kNN is a non-parametric approach which is highly flexible and uses non-linear decision boundaries. Before the implementation of kNN, it is crucial to scale the data since this method relies on the euclidean distance between observations. The number of neighbours is selected by cross-validation;

- **Multinomial naive Bayes (MNB):** The multinomial naive Bayes classifier models the conditional probability of a feature $\boldsymbol{h}$ given its class by a multinomial distribution. The additive smoothing parameter is tuned by cross-validation;
- **Logistic regression (LR):** Logistic regression models the posterior probability of each class given a data point using the logistic function;
- **ℓ_2-regularized logistic regression (LR-ℓ_2):** Logistic regression with ℓ_2 penalization on the weight vector. It is used when feature vectors are not sparse and to avoid overfitting. The regularization parameter $C \in \mathbb{R}_+$ is tuned by cross-validation;
- **ℓ_1-regularized logistic regression (LR-ℓ_1):** Logistic regression with ℓ_1 penalization on the weight vector used for feature selection (cf. (4.31) of [5]). The regularization parameter $\lambda \in \mathbb{R}_+$ is tuned by cross-validation;
- **Soft-margin support vector machine (SVM):** SVM is a separating hyperplane classifier which models the boundary between classes as an hyperplane with maximal margin. Three kernels are tested, e.g. linear (SVM-L), radial basis functions (SVM-R) and sigmoid (SVM-S). The regularization parameter $C \in \mathbb{R}_+$ applied on the slack variables (cf. (12.8) of [5]) is tuned by cross-validation;
- **Decision trees (DT):** Decision trees stratify the feature space recursively into simple regions and assign the label *ham* or *spam* using the majority vote. The CT are fitted using a cost-complexity pruning based on the Gini index and with a minimum number of leaves estimated by cross-validation;
- **Random forests (RF):** Random forests are based on averaging DT grown from random subset of features. The DT are fitted using a cost-complexity pruning based on the Gini index and with a minimum number of leaves estimated by cross-validation. Bootstrap of the samples are used to grow the trees;
- **AdaBoost (AB):** AdaBoost classifier is based on a linear combination of a high number of short DT. The number of DT used in the AB classifier as well as the associated learning are tuned by cross-validation;
- **Multilayer perceptron (MLP):** Multilayer perceptron is a fully connected neural network composed of N hidden layers of dimension D. The considered non-linearity is ReLu. The posterior probabilities of each class are computed on the output of the last hidden layer using softmax. The parameters D and N are optimized by cross-validation.

B. Experimental settings

We are interested in determining not only the model with the best predictive performance, but also the most significant features. Therefore, we decided not to pre-process our data with a dimensionality reduction technique, such as principal component analysis (PCA). Even though the data are imbalanced, we use a threshold of 0.5 for our Bayes plug-in estimator. Other values are discussed in Section IV.

The dataset is divided into a training and a test set with a 80/20 split. Hyper-parameter tuning is achieved by 10-fold cross validation, based on the misclassification error, on the training set. Once the best estimators are identified, the following metrics are computed on the test set:

- Misclassification error (ME): The misclassification error is calculated as follows:
$$ME = \frac{1}{N_t}\sum_{i=1}^{N_t} \mathbb{I}\left(c\left(\boldsymbol{h}_i\right) \neq \gamma\left(i\right)\right), \tag{5}$$
where $\{(\boldsymbol{h}_i, \gamma(i))\}_{i=1}^{N_t}$ designates the features and labels of the test set, c is a classifier and $\mathbb{I}$ is the indicator function;
- Sensitivity (SE): The sensitivity designates the probability of predicting *spam* given that the true class is *spam*. It is calculated as follows:
$$SE = \sum_{i=1}^{N_t} \frac{\mathbb{I}\left(c\left(\boldsymbol{h}_i\right) = \gamma\left(i\right)\right) \times \mathbb{I}\left(\gamma\left(i\right) = 1\right)}{\mathbb{I}\left(\gamma\left(i\right) = 1\right)}. \tag{6}$$
- Specificity (SP): The specificity designates the probability of predicting *ham* given that the true class is *ham*. It is calculated as follows:
$$SP = \sum_{i=1}^{N_t} \frac{\mathbb{I}\left(c\left(\boldsymbol{h}_i\right) = \gamma\left(i\right)\right) \times \mathbb{I}\left(\gamma\left(i\right) = 0\right)}{\mathbb{I}\left(\gamma\left(i\right) = 0\right)}. \tag{7}$$

In this section, we assume that sensitivity and specificity are of equal importance in our setting. Thus, our objective is to minimize the total misclassification error.

The algorithms and methods are implemented on Python, with scikit-learn library and supporting code is available at https://github.com/dperdios/adaptation-and-learning-project.

C. Results

Figure 3 displays the confusion matrices as well as the misclassification error for the proposed classifiers. It can be noticed that all the classifiers perform remarkably well with a maximum misclassification error of 4.6 % for the k-NN classifiers. It can also be observed that LR-based classifiers as well as kernel SVM and MLP perform slightly better than the other classifiers, with a similar misclassification error of 1.6 %.

The reason why tree-based methods do not perform as well as others may be due to the sparseness of the feature vectors, which makes the stratification of the feature space rather difficult.

The classifiers have a very high specificity but suffer from a lower specificity which may be preferable. Indeed, that means that nearly all the ham messages are classified as *ham* while some messages are misclassified. A deeper investigation of sensitivity and specificity will be achieved in Section IV.

Regarding the robustness of the proposed training and comparison strategies, the fact that we use 10-fold cross-validation makes the proposed comparison relatively robust to hyper-parameter changes (provided that the range of parameters tested in the training is set accurately). However, we do not run the experiments for various random distributions of the training/test set, which is necessary to complete the comparison.

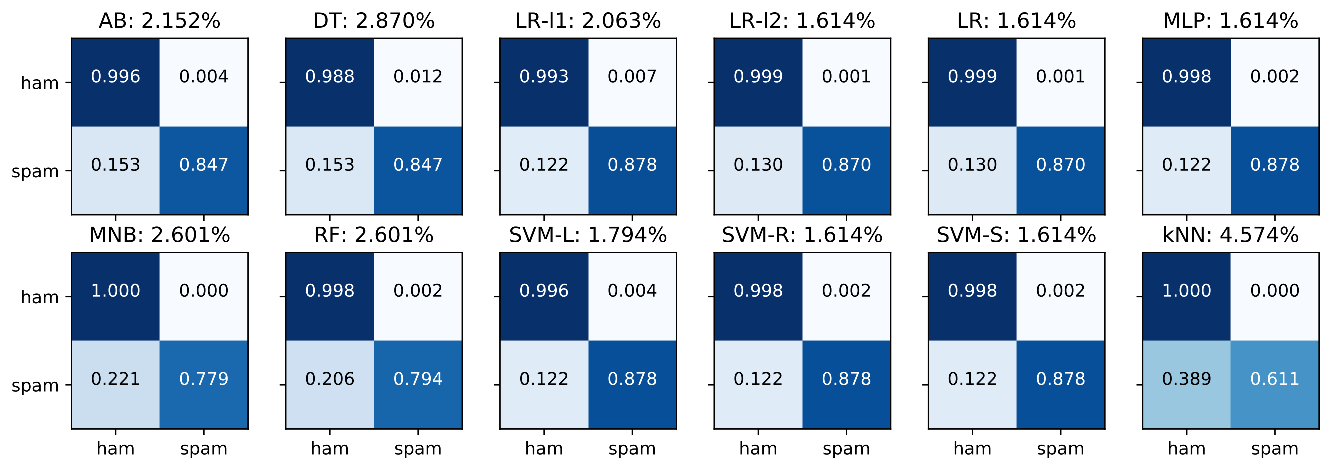


Fig. 3. Confusion matrices of the proposed classifiers.

IV. On Sensitivity And Specificity

A. Increasing the Sensitivity of the Classifiers

The class imbalance coupled with the minimization of the misclassification error induce significant differences in the sensitivity and specificity as it can be seen on Figure 3. Indeed, the misclassification error is a symmetric loss. Hence classifiers focus more on the prediction accuracy of the most common class, which often results in poor accuracy for the other class. In some applications, one may want to increase the sensitivity of the classifiers. That may be the case if the user really wants to avoid spam.

To increase the sensitivity, one may tune the threshold of the Bayes classifier associated with methods that compute posterior probabilities, e.g. logistic regression, neural networks, linear discriminant analysis etc. A higher threshold results in a higher specificity and a lower one to a higher sensitivity.

In this study, we try an alternative based on resampling methods where the idea is to artificially balance the classes. Two methods are tested:

- **Downsampling of the majority class:** we randomly choose a subset of the ham messages to obtain a balanced training set. Intuitively, it may not be ideal since it impacts the predictive performance of the classifiers;
- **Upsampling of the minority class:** we resample with replacement the spam messages in order to balance the training set. Intuitively, it should be better than downsampling but may lead to overfitting in case of highly imbalanced datasets.

B. Results

Figures 4 and 5 display the confusion matrices of the proposed classifiers associated with an upsampled and a downsampled training set, respectively.

First, it can be noticed that the downsampled dataset leads to lower predictive performance as expected, since all the methods suffer from a higher misclassification error than with a normal dataset. However, the sensitivity has been significantly increased, by more than several percent for all the methods. k-NN even gets a 30 % increase of its sensitivity. But evidently, the price to pay is a lower specificity.

Regarding the upsampled training set, the performance of the classifiers are slightly better than with the normal dataset. Such an increase can be explained by a slightly higher sensitivity. However, it can be observed that k-NN may overfit resulting in a lower sensitivity.

Thus, it is clear that upsampling is dangerous in terms of overfitting and has a relatively small impact on the sensitivity, while downsampling works significantly better on the sensitivity but is sub-optimal in terms of misclassification error.

V. Online Learning Strategies

A. Motivations and Strategies

The content of spam messages is changing with time mainly because spammers try to adapt their strategies to fool spam filtering algorithms. So it may be desirable to design online learning strategies where the algorithm is able to adapt to new content labelled by users.

The main difficulty in designing an online strategy for spam filtering is the management of the vocabulary size. Formally, we consider that we have trained a classification model at time t, with a training set of size N_t, from which we have extracted a vocabulary of size V_t. At time $t+1$, due to the new incoming messages, the size of the training set is $V_{t+1} > V_t$. If we consider a naive strategy where the model is trained on the whole training set, feature extraction leads to a vocabulary of size $N_{t+1} > N_t$ which is not viable in in a long-term perspective due to memory and computational issues and because of the curse of dimensionality for several methods. In addition, this strategy may be questionable since vocabulary is evolving. Indeed, it is likely that the most frequent words used at a given time instant may not be the same at a different time instant.

To address such a problem, we can fix the vocabulary size and use only the most frequent words of the training set as features for the classification. Associated with a training set windowed in time, it should be efficient but requires to be updated thus retrained quite often.

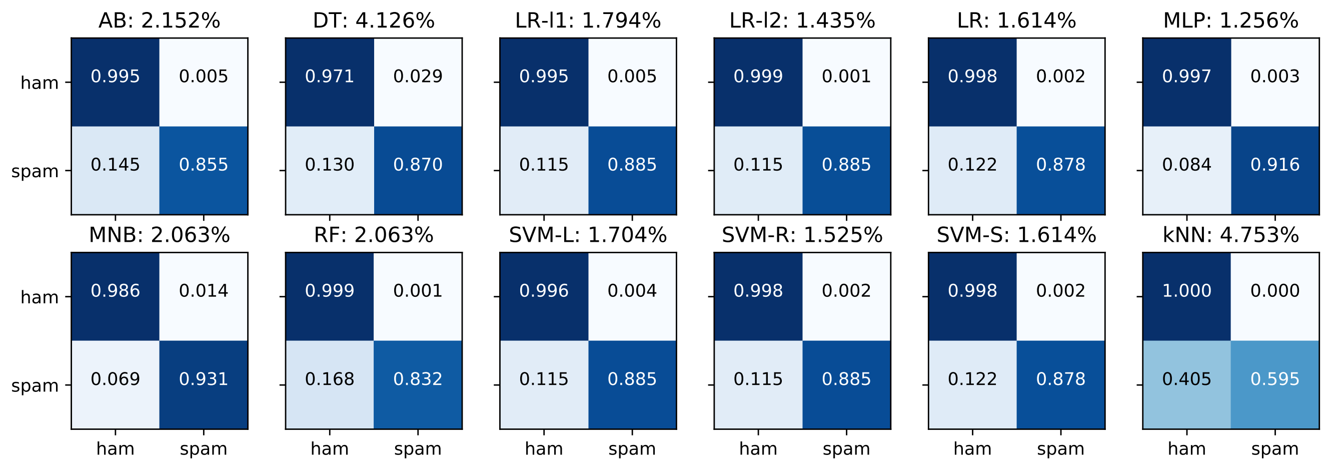


Fig. 4. Confusion matrices of the proposed classifiers fitted on an upsampled training set.

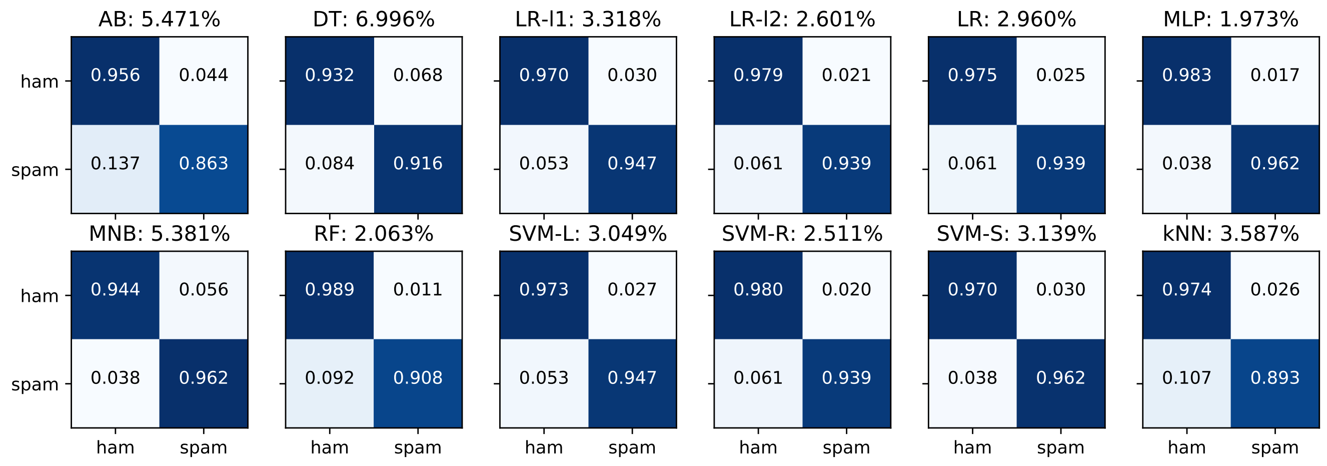


Fig. 5. Confusion matrices of the proposed classifiers fitted on a downsampled training set.

B. Experiments

In the considered dataset, we do not have access to time information related to the messages. Thus, it is impossible to analyze the evolution of the vocabulary with time. In addition, the size of the dataset is relatively small and the models are trained within several minutes for the slowest ones.

In order to simulate an online learning scenario as best as possible, we assume that we receive labelled messages by batches of $N_M = 100$, at given time instants t_i, for $i = 1, \ldots, N_B$, where N_B denotes the number of batches. We consider that the test set is fixed from the beginning. It is composed of 1115 messages randomly extracted from the entire dataset, which corresponds to 20 % of the dataset.

At each time instant, we complete the training set with the new batch of messages resulting in $i \times N_m$ messages at time t_i. We achieve feature extraction on the training set, fit the classification model and compute the error on the test set.

We are therefore using the naive strategy described in Section V-A where the size of the vocabulary is growing with time. This is justified by the fact that the dataset is too small to encounter computational and memory issues.

Figure 6 displays the average misclassification error of the proposed strategy for LR-ℓ_2 and MNB classifiers over 100 random distributions of the training/test set. The choice of these two classifiers is motivated by the fact that they can be trained very fast, which is a requirement for online learning.

We observe a consistent behaviour of the error, which is decreasing when the size of the training set increases. It is characteristic of the transitional regime of the algorithm, where the size of the dataset does not prevent from taking into account the whole training set.

At this stage, it would have been interesting to work on a bigger dataset in order to address cases where we would have to limit the size of the vocabulary.

VI. Conclusion

We have studied several SMS spam filtering methods from feature extraction step to classification. In such techniques, occurrences of words in each message are extracted and features are obtained through Tf-Idf normalization, which takes into account both term frequencies across a given document and among documents. We trained several classifiers, e.g. logistic regression, regularized logistic regression, linear and kernel support vector machine (SVM), k-nearest neighbours, multinomial Bayes, decision trees, random forests, AdaBoost

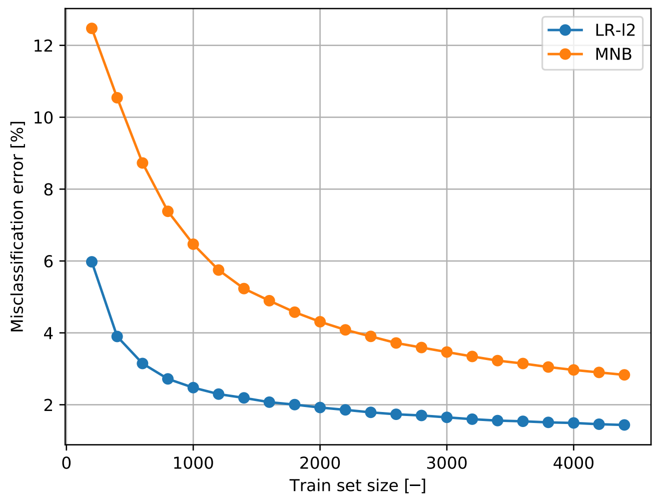


Fig. 6. Misclassification error of the logistic regression with the online learning strategy.

and neural networks, where the best hyper-parameters have been identified using 10-fold cross validation.

We showed that all the classifiers perform remarkably well with a misclassification error of at most 5 %. Among the different methods, logistic regression, SVM and neural-network-based approaches seem to give the best results. We have also studied two resampling methods to balance the values of the specificity and the sensitivity, which were significantly different due to class imbalance. We demonstrated that both of them have several drawbacks and require trade-off between target specificity and sensitivity. We have finally investigated online learning techniques where the classifier is sequentially updated with new batches of data.

In future work, extensive tests should be performed on a bigger dataset. Online learning strategies may be adapted to vocabulary of limited size. More advanced feature extraction methods such as word embeddings as well as deep-learning-based classifiers may be considered.

References

[1] D. Suleiman and G. Al-Naymat, “SMS Spam Detection using H2O Framework,” *Procedia Comput. Sci.*, vol. 113, pp. 154–161, 2017.

[2] T. A. Almeida, J. M. G. Hidalgo, and A. Yamakami, “Contributions to the study of SMS spam filtering,” in *Proc. 11th ACM Symp. Doc. Eng. - DocEng ’11*. ACM Press, 2011, p. 259.

[3] Y. Goldberg, “Neural Network Methods for Natural Language Processing,” *Synth. Lect. Hum. Lang. Technol.*, vol. 10, no. 1, pp. 1–309, apr 2017.

[4] C. D. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*. Cambridge: Cambridge University Press, 2008.

[5] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY: Springer New York, 2009.