ASSIGNMENT 3:- Spam v/s Ham

CS5691:Pattern Recognition and Machine

Course Instructor : Arun Rajkumar

Pragalbh Vashishtha **MM19B012**

April 2022

Due to size constraints these files are uploaded on the following drive link:-Drive Link

1 Introduction

Problem statement:- We are given a Binary Classification problem, Spam v/s Ham, where we classify emails based on whether it is harmless(ham) or spam.

We have to build a spam classifier from scratch using publicly available data-sets. We shall be using two data-sets, namely the Enron Data-set and Spam-assassin Data-set. As we don't have access to the test Data-set, we have to consider and carefully choose the pre-processing steps, the models we decide and the feature extraction we perform.

2 Preprocessing and Data

2.1 Data

Firstly, we collected all the spam and ham data together, the formats of the Spam assassin data-set was not txt, so we converted it to txt, and then sorted all the data (Enron +Spam-Assassin) into spam and ham.

Summary of the data, after duplicates and erraneous files were removed:

Dataset	# Spam	# Ham
Enron	17,157	16,545
Spam Assassin	2,398	6951

Table 1: Counts of spam and ham in the training data

80% of the data was kept as training data and the rest was used as cross-validation for the models.

2.2 Preprocessing

The emails are first loaded in 'r+' mode with 'utf-8' encoding. The email library is used with the combination of regex to clean the emails.

The following cleaning procedures were done:-

- Remove escape characters.
- Remove '\', '\n', '\r and other escape characters.
- Extra spaces are replaced by a single space ' '.
- Images, links, ip-addresses, currency signs are replaced by placeholder words.

After this basic stopwords form the nltk library were removed. However this is not enough. Hence a strategy of looking at the most used words is applied, Words are counted and a vocabulary is created, wherein the most repeating words, which contain no particular useful information, such as, emailaddr, enron, subject, etc. are removed. Also a few words, which are related to rtf formatting in the test data were removed by my discretion,

as they do not add value to discriminating between spam and non-spam.
Note stemming has not been done, due to the possibility of stem-words removing important information crucial to detecting spam. Case in point, caring and car are both stemmed to car, which looses definite information

The final step for preprocessing common to all algorithms is tokenizing the words, by breaking each email into a list of words. Hence the final data was a list of words for each email, which was nested in a list containing all emails. The spam emails were marked +1 while the ham emails were marked 0.

For algorithms like SVMs, Adaboost etc., words need to be converted into vectors, hence a bag of words feture matrix was used. Taking the 2500 most frequent words as features, this bag of words matrix was created. For each email, a vector e_i is created, if a word existed, the entry corresponding to the word in the vocabulary, say the j^{th} entry assigned the value 1, else 0, hence $e_{ij} = \mathbb{1}(word_j \text{ is in } email_i)$. Thus a vector of $\{+1, 0\}^{2500}$ is used for each email.

3 Models

3.1 Naïve Bayes

3.1.1 Formulation

Consider representing an email as the set x1, x2, . . . , xn of distinct words contained in the text. We are interested in computing:-

$$\mathbb{P}(S|x_1, x_2, ..., x_n)$$

where S is the event that the email is spam, and H is the event that the email is ham (equivalent to S). By Bayes' Theorem, this is equivalent to

$$\mathbb{P}(x_1, ..., x_n|S)\mathbb{P}(S)$$

$$\mathbb{P}(x_1, ..., x_n) = \mathbb{P}(x_1, ..., x_n|S)\mathbb{P}(S)$$

$$\mathbb{P}(x_1, ..., x_n|S)\mathbb{P}(S) + \mathbb{P}(x_1, ..., x_n|H)\mathbb{P}(H)$$

Ignoring the denominator for the moment, by definition of conditional probability,

$$\mathbb{P}(x_1, ..., x_n|S)\mathbb{P}(S) = \mathbb{P}(x_1, ..., x_n, S)$$

We take the conditional indepence assumption as disucssed in class

Let's rewrite those chain rule probabilities using the conditional independence assumptions.

$$\mathbb{P}(x_1, ..., x_n, S) = \mathbb{P}(x_1|x_2, x_3, ..., x_n, S)\mathbb{P}(x_2|x_3, ..., x_n, S)...\mathbb{P}(x_{n-1}|x_n, S)\mathbb{P}(x_n|S)\mathbb{P}(S)$$

$$\approx \mathbb{P}(x_1|S)\mathbb{P}(x_2|S)...\mathbb{P}(x_{n-1}|S)\mathbb{P}(x_n|S)\mathbb{P}(S)$$

$$= \mathbb{P}(S)\prod_{i=1}^{n}\mathbb{P}(xi|S)$$

Similarly for ham,

$$\mathbb{P}(x1, ..., xn, H) \approx \mathbb{P}\prod_{i=1}^{n}\mathbb{P}(xi|H)$$

Finally we get,

$$\mathbb{P}(S|x1, ..., xn) \approx \frac{\mathbb{P}(S)\prod_{i=1}^{n}\mathbb{P}(xi|S)}{\mathbb{P}(S)\prod_{i=1}^{n}\mathbb{P}(xi|S)\mathbb{P}(S)\prod_{i=1}^{n}\mathbb{P}(xi|S) + \mathbb{P}(H)\prod_{i=1}^{n}\mathbb{P}(xi|H)}$$

3.1.2 Implementation

- Iterate over the labelled spam emails and, for each word w in the entire training set, count how many of the spam emails contain w. Compute $\mathbb{P}(w|S) = \frac{|spam\ emails\ containing w|+1}{|spam\ mails|+V}$. Compute $\mathbb{P}(w|H)$ the same way for ham emails
- To handle words which don't occur in the data, Laplacian smoothing is done, hence the added 1 and V in the numerator and denominator respectively. Note, V is the size of the vocabulary.
- Compute $\mathbb{P}(S) = \frac{|spam\ emails|}{|spam\ emails|+|ham\ emails|}$ and $\mathbb{P}(H) = \frac{|ham\ emails|}{|spam\ emails|+|ham\ emails|}$
- Given a set of unlabelled test emails, iterate over each:
  - Create a set x1, . . . , xn of the distinct words in the email. Ignore the words that you haven't seen in the labelled training data
  - Compute

$$\mathbb{P}(S|x1, ..., xn) \approx \frac{\mathbb{P}(S)\prod_{i=1}^{n}\mathbb{P}(xi|S)}{\mathbb{P}(S)\prod_{i=1}^{n}\mathbb{P}(xi|S)\mathbb{P}(S)\prod_{i=1}^{n}\mathbb{P}(xi|S) + \mathbb{P}(H)\prod_{i=1}^{n}\mathbb{P}(xi|H)}$$

  - If $\mathbb{P}(S|x1, ..., xn) > 0.5$, output spam (+1), else output ham(0).

3.1.3 An interesting Conundrum :- Underflow!

When we multiply a set of small probabilities together, (as we are considering even low frequency words for the sake of accuracy) while calculating probability of ham and spam above, a floating point underflow results, where the product will become too small to represent and will be replaced by 0.
So an idea of using logarithms for probabilities was devised.

$$log(\mathbb{P}(S)\prod_{i=1}^{n}\mathbb{P}(xi|S))$$

which can be written equivalently as, $log(\mathbb{P}(S)) + \sum_{i=1}^{n} log(\mathbb{P}(xi|S))$ which does not not underflow. Then, realize if

$$log(\mathbb{P}(S)) + \sum_{i=1}^{n} log(\mathbb{P}(xi|S)) > log(\mathbb{P}(H)) + \sum_{i=1}^{n} log(\mathbb{P}(xi|H)$$

we classify it as spam, else ham

3.1.4 Performance

Naive Bayes has an accuracy of 97.77003484320558% on the cross-validation data.

3.2 Adaboost

3.2.1 Formluation & Implementation

For Adaboost, we first looked into the bag of words model. As discussed previously, each email is converted into a $\{+1, 0\}^{2500}$ vector. After this is done, we implement Adaboost as follows:-

- Convert the predictions from $\{0, 1\}$ to $\{-1, +1\}$
- Initialize uniform probabilities for each sample, i.e, $D_1 = \frac{1}{n}$
- for iterations 1,2,... T:-
  - Get a weak hypothesis h_t by inputting the data-set
  - Choose $\alpha_t = 1/2 * ln(1 - \epsilon_t/\epsilon_t)$, where ϵ_t is the error of h_t at that round.
  - $D_{t+1(i)} = D_{t(i)}exp(-\alpha * t * y_i * h_t(x_i)/Zt$, where Zt is the normalisation factor equal to sum of all $D_{t(j)}$ for all j.
- The final output is $H(x) = \sum_{t=1}^{T} \alpha_t * h_t(x)$

The Decision Stumps/ Weak classifiers were decision trees with small number of nodes and depth (depth¡5).

3.2.2 Performance

Adaboost has a accuracy of 97.53774680603949% on the cross-validation Data-set. This is slightly worse than Naive Bayes

3.3 SVM

3.3.1 Formluation & Implementation

Taking the Dual of the Soft Margin SVM,

$$\text{maximize } \alpha^T 1 - \frac{1}{2}\alpha^T y^T (\mathbf{x}^T \mathbf{x}) y\alpha,$$

$$\text{subject to } 0 \leq \alpha \leq C \text{ for all } i.$$

This dual can also be kernelized, to get,

$$\text{maximize } \alpha^T 1 - \frac{1}{2}\alpha^T y^T (\mathbb{K}) y\alpha,$$

$$\text{subject to } 0 \leq \alpha \leq C \text{ for all } i.$$

$$\text{where } \mathbb{K} = \phi(\mathbf{x})^{\mathbf{T}} \phi(\mathbf{x})$$

So based on kernel choice and the C parameter, we can get various results and tune accordingly

3.3.2 Performance

Kernel	C/degree	Accuracye
rbf	C=1	98.14169570267132 %
rbf	C=2	98.19976771196284 %
rbf	C=3	98.28106852497096 %
rbf	C=4	98.25783972125436 %
rbf	C=5	98.21138211382113 %
rbf	C=6	98.24622531939605 %
linear	degree=2	97.06155632984901 %
linear	degree=3	97.06155632984901 %

Table 2: SVM Performance Table

From the above performance table we note that the Radial Basis Function kernel with C value of 3.0 is optimal

3.4 Logistic Regression

3.4.1 Formluation & Implementation

The final model we looked at is the logistic regression model. The loss is defined as

$$\sum_{i=1}^{n} [y_i \log(h(x_i) + (1 - y_i) \log(1 - h(x_i)]$$

where

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

Now plugging this into gradient descent we get the gradient descent step as:-

$$w_{t+1} = w_t - \eta_t (\sum_{i=1}^{n} (x_i (h(x_i) - y_i))$$

When w add regularization, the cost function becomes:-

$$\sum_{i=1}^{n} [y_i \log(h(x_i) + (1 - y_i) \log(1 - h(x_i)] + \frac{\lambda}{2} ||w||^2$$

The gradient descent becomes:-

$$w_{t+1} = w_t - \eta_t (\sum_{i=1}^{n} (x_i (h(x_i) - y_i)) + \lambda ||w||$$

3.4.2 Performance

λ	Performance
0	97.25900116144018 %
0.4	97.23577235772358 %
0.8	97.21254355400697 %
1.0	97.20092915214866 %
1.4	97.16608594657375 %
1.8	97.15447154471545 %

Table 3: Performance of Logistic regression w.r.t λ values

3.5 Conclusion

By comparing the various accuracies of the Models it is noted that SVM with RBF kernel performs the best, followed by Naive Bayes, which is followed by Adaboost, Logistic regression, and Linear kernel SVM in that order.

The C parameter of the SVM kernel is 3.0 with an accuracy of 98.28% on the cross-validation data.

4 Running the classifier

To run the classifier please run 'predict.py'. Replace the test folder in the directory with the test folder you want to test on.
the 'predict.py' imports the functions from utility.py, the vocabulary from Vocab.picklr, and finally the model from model_pkl.
The output is given on the console and then is saved to 'output.txt' file

Due to size constraints these files are uploaded on the following drive link:-Drive Link