SPAM-HAM TEXT MESSAGE CLASSFIER

Mrinal Kumar

Introduction:

Text messages have provided us a form of communication channel which enables us to communicate across the globe in asynchronous and very succinct way. This helps to convey information in a timeframe that is convenient to both, sender and the receiver. While text messages have several legitimate applications, it is often abused in terms of use, such as unsolicited advertisement and sometimes inappropriate messages. Based on content of message they are classified into two categories namely: *'Ham'- Genuine message*; *'Spam'- Unsolicited messages*. In order to save user from Spam we use language processing technique of text classification which enables us to filter out 'Spam'. Here in this report I talk about implementation of such classifier called '**Perceptron Classifier**' further improved as '**Averaged Perceptron**'. It uses supervised technique for learning difference between different classes.

Building feature set:

To develop an efficient classifier we need to develop rich feature set from the training data that helps classifier to learn and assign weights precisely for further classification. If we closely observe spam and ham SMS, we can see various distinctive features between them. Generally, most prominent difference between these class of messages are used words. For example let us consider following spam and ham message:

Spam: Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005.
Ham: Nah I don't think he goes to usf, he lives around here though

By observing various spam and ham messages we can conclude that following can be considered while building feature set:

- Use of words like 'Free', 'win' are frequently observed in spam messages, as the case in above example, which acts as a classification feature.
- Words like 'San Francisco', 'CALL ME', 'Free entry' frequently occurs together so using bigram features increases efficiency.
- **URL**: whenever we find text in the form of /something/something it is replaced by word "#URL#" and counted as feature. Spam will more likely have a url than a normal text.
- **Offset**: This feature is introduced to tackle ties while classification.

These features are generated after data preprocessing like stop words, Lemmatization with the help of nltk libraries like WordNetLemmatizer(), SnowballStemmer() etc.

Perceptron Classifier:

Perceptron classifier works in error driven technique where weight update or learning takes place only when classifier makes mistake. This gives Perceptron advantage over naïve bayes which is a probabilistic learning model having very strong assumption of independence between features. Perceptron converges for linearly separable data i.e. data where we can find line plane or hyper-plane that separates data into two classes. It is not always possible to find a separator that separates the data completely and Perceptron trashes in this case. Solution to this problem is averaged Perceptron model where we average the weights across all iteration. Both of these are online learning classifier so there is no need to have whole dataset into memory while learning. This is a great advantage over traditional naïve bayes model.

Implementation:

To develop a feature set we need to preprocess the data. In this case I built a 'Bag of words (BOW)' model from the given data. While building BOWs we need to identify words and symbols that are redundant or can misguide the classifier and remove them. This can be identified by observing the data classes, trying out various data pre processing followed by result validation. Based on the observation and trial I came up with following:

- Removal of numbers, punctuation and irrelevant characters: These are generally observed in both the classes so can be removed from feature set. e.g. Ham and spam both can have telephone numbers, some date, smiley. So counting on results from these data can sometimes mislead the classifier.
- Removal of determiners and other stop words: These are also common in both classes and including those to feature set may confuse the classifier in the similar manner.
- Lemmatization of words: Words like 'go', 'goes', 'going' have same root and essentially same context in most of the cases. So, having different features for all of them may lead to very few occurrence of each of them while learning. e.g. If we never see 'goes' in our training data then it may reduce accuracy while evaluating our classifier on unknown data(test data).

After these data pre-processing I create BOW which contains a row corresponding to every text message. Each row contains words with its count in that text message. Single row in the BOWs looks like:

Actual data: ham Nah I don't think he goes to usf, he lives around here though
BOWs row: usf:1 around:1 though:1 i:1 nah:1 goe:1 live:1 think:1

The classifier is initially trained in supervised manner so that it can learn weights corresponding to feature set. We provide feature instances, weights, and all the labels which in this case is 'Ham' and 'Spam'. Algorithm proceeds in following manner:

Training Algorithm:

```
for each feature, feature_count, label in all_instance
  predict label of single instance based on weights.
  if predicted_label not equal to actual_label
        update the weight
        increase training error
return weights
```

After classifier is trained weights are applied on test data set to evaluate the performance. To improve this algorithm I used averaged perceptron where I keep a weighted sum which is averaged after every iteration.

$$\underset{\emptyset}{\rightarrow} = \frac{1}{T} \sum \emptyset_t$$

Given dataset was partitioned into 10 sub samples using k-fold technique, k=10. So, 90% of the sampled data was used to train the classifier and 10% for the evaluation. This process was repeated k times and in each iteration one random sample out of k samples was selected as test data.

Evaluation and Analysis:

To train and evaluate the performance of the classifier it was required to partition the given dataset into training data and test data. To ensure a fair partition I used K-fold partitioning where sampled data was partitioned into k-partition out of which (k-1) were used for training and remaining one for testing. This

was repeated k number of times and the result of every iteration was combined to report the final accuracy.

In test phase given a text message classifier returned label for the text. The classification has following state:

True Positive (TP) : Spam classified as Spam	True negative (TN) : Ham classified as Ham
False Positive (FP) : Ham classified as Spam	False negative(FN) : Spam classified as Ham

These states were used to evaluate performance of the classifier by calculating precision, recall and accuracy.

Precision(p) = TP / (TP + FP)	Recall(r) = TP / (TP + FN)	F1 measure = (2 * p * r) / (p + r)	Accuracy = (TP + TN)/ (TP + TN + FP + FN)

After fine tuning on data pre-processing, k-fold and feature classifier gives following result. Fig.1 shows the number of tagging error and median error that averaged Perceptron classifier makes for different values of k.

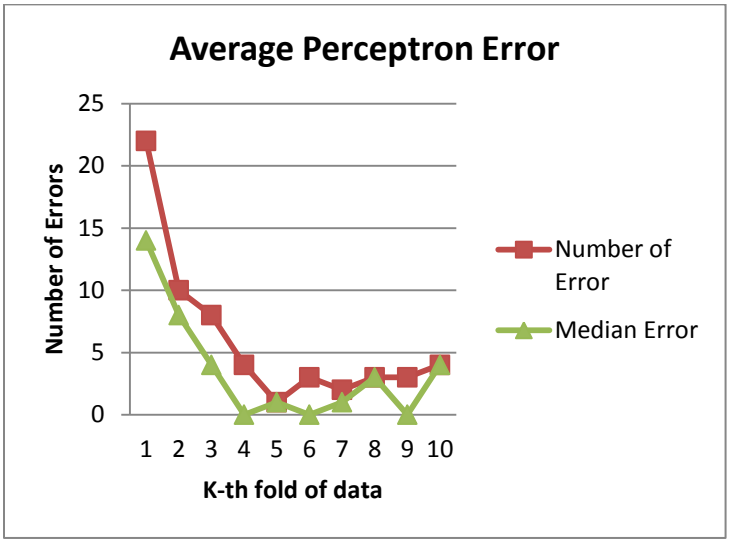

Fig.1

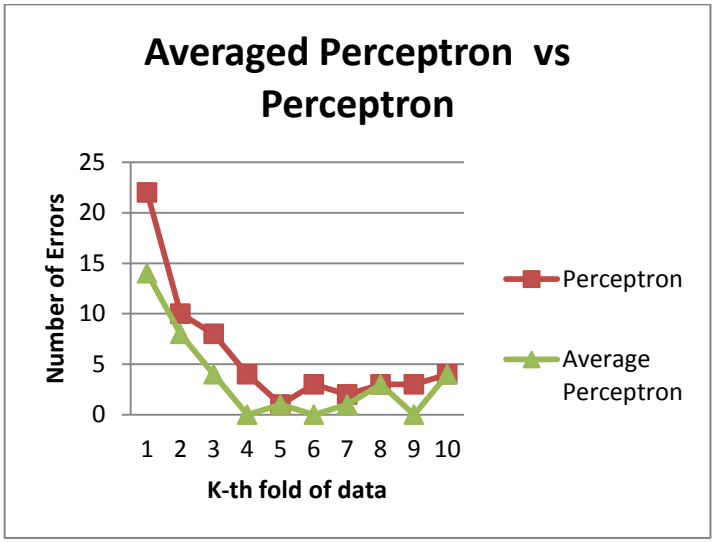

Fig.2.

Fig.2 shows how averaged Perceptron performs better than traditional Perceptron in terms of number of tagging error. In Table.2 we can see detailed performance evaluation in terms of precision, recall, accuracy and F1 measure (up to two decimal places) for averaged Perceptron model.

Table.1: Performace evaluation measure for different values of k for averaged perceptron

Kth fold	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Avg.**
Precision	0.90	0.96	0.95	1	1	1	1	0.97	1	0.97	0.975
Recall	0.90	0.94	0.98	1	0.98	1	0.98	0.98	1	0.97	0.973
F1 measure	0.9	0.95	0.96	1	0.99	1	0.99	0.97	1	0.97	0.97
Accuracy	0.97	0.98	0.99	1	0.99	1	0.99	0.99	1	0.99	0.99