ED21B018

Foundations of Machine Learning
Assignment 2 Report

SPAM-HAM CLASSIFER

November 3, 2024

Dataset

A combination of 2 available datasets- Enron and Spam email dataset on Kaggle.

1. Enron Dataset- Contains already preprocessed emails in txt for each mail. Model was trained on both these folders
    - Enron 1: Spam-1500, Ham - 3672
    - Enron 3: Spam-1500, Ham- 4012

    Validation set chosen was Enron 2: Spam-1496, Ham-4361

2. Spam email dataset was tried on Adaboost+ Descision tree model only, saved as a csv file with a column containing mail content and another column with 0/1

Algorithms used

1. **Naive Bayes:** According to its name, being very Naive, as it considers every word indpedently and calculated the probability of occurence according to Bayes rule, lot of preprocessing had to be done.

Parameters to estimate:

$$p, \{p_1^1, \ldots, p_d^1\}, \{p_1^0, \ldots, p_d^0\}$$

Maximum Likelihood estimators

$$\hat{p} = \frac{1}{n}\sum_{i=1}^{n} y_i \leftarrow \text{[Fraction of Spam email in data]}$$

$$\hat{p}_j^y = \frac{\sum_{i=1}^{n} 1(f_j^i = 1,\ y_i = y)}{\sum_{i=1}^{n} 1(y_i = y)}$$

label: y; j^{th} word

[Fraction of y-labeled emails that contain j^{th} word]

Given $x_{test} \in \{0,1\}^d$, [test email]

What is y_{test}?

Predict 1 if $P(y_{test} = 1 / x_{test}) > P(y_{test} = 0 / x_{test})$

Predict 0 otherwise.

BAYES RULE

$$P(y^{test} = 1 / x^{test}) = \frac{P(x^{test} / y^{test} = 1) \cdot P(y^{test} = 1)}{P(x^{test})} \leftarrow$$

Preprocessing pipeline:

a. Removal of stop words and punctuation- This was done with `sklearn_feature _extraction.text ENGLISH_STOP_WORDS` library and punctuation removal and lower casing from `Regex.`
b. Lemmatization: Removal of tenses formats of each word, reduces the word/vector space that we should work with. This retains meaning, but reduces the dimension of problem statement.
c. Custom Stop words: As we know the nature of mail data, there are the filler formats of email writing that I utilised to remove unnecessary words that doesnt give any information. `Ex: Please, Regards, Good Morning, Thanks, email.` All these words arent included in the extracted mail text.
d. URL, Mail id removal: Directly moving to the Subject of the mail is what we humans subconsciously do. Hence I removed unnecssary information that are saved before the “Subject”, ex: email ids of sender, cc, bcc and some URLs in the body of the mail.

Feature Engineering:

a. Domain specifc spam_words_list- Adding domain knowledge of nature/sentiment of spam messages always help. Spam_word_list which when found in the test said, have a condition of doubling the count, increasing the probability of mail being spam.

```
domain_specifc_spam_words= ["urgent", "bonus", "unsubscribe", "winner", "claim", "discount", "buy now", "free"] etc

domain_specifc_ham_words= ["meeting", "schedule", "project", "client", "presentation", "follow-up", "update"] etc
```

b. TF-IDF vectorisation- Calculates importance of a word in document according to term frequency and inverse document frequency. Again, I removed the words that appear in less than 0.001 fraction of words, and more than 90% of the document frequency as they wont add distinquishing value/information.
c. Punctuation_percentage: Another peculiar observation is that spam mails hav multiple ‘!!!!!’ and such unprofessional attention gaining mechanisms. Hence I added a punctuation_percentage to with respect to the number of words, which is a give-away feature to detect spam.

Trained data: P(spam/word) andd P(ham/word) are saved to json files after training with counts `P_word_spam= {word: ((spam_dict[word])/(total_spam_count)) for word in column_names}`

Validation: is_spam function does the classification. To handle class-imbalance, initiliased according to the distribution. Finally whichever probability is higher is classified.

```
spam_score, ham_score= np.log(P_spam), np.log(P_ham)

spam_score += word_count * np.log(P_word_spam[word])
```

Final results on Enron 2 comes out to be-

Accuracy	**96.82%**
F1 score	**96.82%**
Balanced Accuracy	**96.35%**

2. **Adaboost:** With Decision tree weak classifier.
    a. Word2Vec embedding- converted using gensim model to 100 length vector, after lowercasing and punctuation removal
    b. Adaboost Code:

    The below algorithm is implemented on Decision Stump weak learner. This is chosen as Decision trees and stumps perform much better as weak learners for adaboost than Linear models like LDA, Logistic regression,etc as they lead to a linear pieces fit (not so smooth) boundary.

    According to the error, the email set diffcult to classify is sampled in a greater affinity, and all the clfs and parameter alpha are saved.

With:

- Samples $x_1 \dots x_n$
- Desired outputs $y_1 \dots y_n, y \in \{-1, 1\}$
- Initial weights $w_{1,1} \dots w_{n,1}$ set to $\frac{1}{n}$
- Error function $E(f(x), y_i) = e^{-y_i f(x_i)}$
- Weak learners $h: x \to \{-1, 1\}$

For t in $1 \dots T$:

- Choose $h_t(x)$:
  - Find weak learner $h_t(x)$ that minimizes ϵ_t, the weighted sum error for misclassified points $\epsilon_t = \sum_{\substack{i=1 \\ h_t(x_i) \neq y_i}}^{n} w_{i,t}$
  - Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$
- Add to ensemble:
  - $F_t(x) = F_{t-1}(x) + \alpha_t h_t(x)$
- Update weights:
  - $w_{i,t+1} = w_{i,t} e^{-y_i \alpha_t h_t(x_i)}$ for i in $1 \dots n$
  - Renormalize $w_{i,t+1}$ such that $\sum_i w_{i,t+1} = 1$
  - (Note: It can be shown that $\frac{\sum_{h_t(x_i)=y_i} w_{i,t+1}}{\sum_{h_t(x_i)\neq y_i} w_{i,t+1}} = \frac{\sum_{h_t(x_i)=y_i} w_{i,t}}{\sum_{h_t(x_i)\neq y_i} w_{i,t}}$ at every step, which can simplify the calculation of the new weights.)

c. Decision Stump code: It's a simple binary classifier that makes decisions based on a single feature threshold. During fitting, it:

- Tries every feature and every possible threshold value
- For each split, tries both possible polarities (predicting +1 or -1 above threshold)
- Keeps track of the configuration that gives the lowest weighted error

For prediction, it simply checks if the chosen feature is above/below the threshold and returns +1/-1 accordingly.

But the results turn out to be just better than 50% for stumps, as I have a 100 length vector and need a heavier model to choose the best features, and their dependencies.

Algorithm 1 DecisionStump Training

procedure FIT(X, y, sample_weights)
  if sample_weights = null **then**
    sample_weights ← ones(length(y))
  end if
  min_error ← ∞
  for feature ← 1 to n_features **do**
    feature_values ← X[:, feature]
    thresholds ← unique(feature_values)
    for threshold in thresholds **do**
      predictions1 ← ones(n_samples)
      predictions1[feature_values ¡ threshold] ← -1
      error1 ← $\frac{\sum \text{sample\_weights} \cdot (predictions1 \neq y)}{\sum \text{sample\_weights}}$
      predictions2 ← ones(n_samples)
      predictions2[feature_values $\geq$ threshold] ← -1
      error2 ← $\frac{\sum \text{sample\_weights} \cdot (predictions2 \neq y)}{\sum \text{sample\_weights}}$
      if error1 ¡ min_error **then**
        min_error ← error1
        feature_idx ← feature
        threshold_value ← threshold
        polarity ← 1
      end if
      if error2 ¡ min_error **then**
        min_error ← error2
        feature_idx ← feature
        threshold_value ← threshold
        polarity ← -1
      end if
    end for
  end for
end procedure
procedure PREDICT(X)
  predictions ← ones(length(X))
  if polarity = 1 **then**
    predictions[X[:, feature_idx] < threshold_value] ← -1
  else
    predictions[X[:, feature_idx] $\geq$ threshold_value] ← -1
  end if
  return predictions
end procedure

Tried methods:

Added a feature- Count of ‘meaningless english words’ instead of adding every meaningless word as a feature to be counted. I saw many gibberish words and removed them using `wordnet.synsets()` that checks with an english dictionary. But turns out that there are more

Proper Nouns in ham mails as well, like, name of clients which are also not recognised, and wasnt a differentiating factor.

Inference

Data preprocessing along with domain knowledge of the data can change a Naive model with 80% accuracy to 97% accuracy and more. Feature engineering is a very important step, along with dimension reduction.

Naive Bayes is the the most intuitive model for this problem statement, and can accomlosh good results with additonal sentiment analysis, n_grams etc., yet it can be very slow

Robustness of model can depend on diversity of dataset, along with finetuning of hyperparametrs.

More complex models like SVM, Perceptron or Adaboost needs careful parameters, but not necessarily as much preprocessing, ccan help when we dont know much about the data.

Code files guide:

1. FINAL_TEST

   Takes in the pkl and json files that are saved after training to give final prediction from the test folder in directory

2. Naive_Bayes_Training

   Full training final implementation as explained in the report

3. spam_words_probab3, ham_words_probab3

   JSON files that are saved after training on final model, containing probabilities of spam and ham given a word

4. tfidf3.pkl

   Saved vectorizer to be used during testing

5. Adaboost_training

   Notebook with from-scratch Adaboost implementation

6. English_meaning_trial

   A trial notebook with the featue engineering approach of linguisticss and english meanings

7. archive

   Folder containing dataset