MM21B030 Hiran N

Foundations of Machine Learning
Spam-Ham Classification

Data Preprocessing

In this section, the steps taken to clean and prepare the text data for analysis using the NLTK and re libraries are outlined. This preprocessing step is crucial as it helps eliminate less meaningful words and characters, thereby enhancing the performance of the machine learning model.

1. **Strip Extra Spaces and Tabs:** This involves removing any leading, trailing, or multiple spaces and tabs within the text, ensuring uniform spacing throughout the dataset.
2. **Strip Special Characters:** Special characters, such as punctuation and symbols that do not contribute to the sentiment or meaning of the text, are removed.
3. **Replace Numbers with 'number'**: Numeric values are replaced with the placeholder 'number'. This is done because the actual value of the numbers is often not significant for spam detection; rather, it is the presence of numbers that might be indicative of spam content.
4. **Strip English Stop Words:** Commonly used words (e.g., 'the', 'is', 'and') that do not add meaningful context are removed from the dataset. This helps to reduce noise and focuses the model on more informative words.
5. **Replace Links with 'link':** URLs or hyperlinks in the text are replaced with the word 'link'. This prevents the model from being influenced by potentially irrelevant web addresses.
6. **Lowercase All Characters:** All text is converted to lowercase to ensure that the model treats words like "Spam" and "spam" as identical, reducing redundancy in the feature set.

Model_1 Naïve Bayes algorithm:

Feature Extraction

The preprocessing steps mentioned above result in a cleaner dataset, which is then transformed into features using the CountVectorizer(). CountVectorizer converts text data into a matrix of token counts, where each unique word in the dataset becomes a feature. It represents each document as a vector, with values showing the frequency of each word in that document.

Parameter Extraction

The following lines of code are used to calculate the necessary parameters for the Naïve Bayes model:

```
# Calculate the probability of spam (P(spam))
self.p_spam = X_spam.shape[0] / (X_spam.shape[0] + X_ham.shape[0])
```

P(spam): This line calculates the prior probability of a message being spam. It is determined by dividing the number of spam messages (X_spam.shape[0]) by the total number of messages (both spam and ham). This gives a proportion of how many messages in the dataset are classified as spam.

 # Calculate the probability of each word given spam (P(word|spam)) and ham (P(word|ham))
self.p_spam_words = (X_spam.sum(axis=0) + 1) / (X_spam.sum() + X_vectorized.shape[1])
self.p_ham_words = (X_ham.sum(axis=0) + 1) / (X_ham.sum() + X_vectorized.shape[1])

P(word|spam): This computes the conditional probability of each word occurring in a spam message. The numerator counts the occurrences of each word in spam messages (X_spam.sum(axis=0) + 1 for Laplace smoothing), while the denominator adds the total count of words in spam messages plus the number of unique words in the vocabulary (X_vectorized.shape[1]). This helps avoid the zero probability issue for words that do not appear in spam messages.

P(word|ham): Similarly, this computes the conditional probability of each word occurring in a ham message using the same logic as above.

Prediction

The following code snippet calculates the log probabilities for both classes, spam and ham:

 # Calculate log probabilities for numerical stability
log_prob_spam = np.log(self.p_spam) + x_vec.dot(np.log(self.p_spam_words.T)) + (1 - x_vec).dot(np.log(1 - self.p_spam_words.T))
log_prob_ham = np.log(1 - self.p_spam) + x_vec.dot(np.log(self.p_ham_words.T)) + (1 - x_vec).dot(np.log(1 - self.p_ham_words.T))

Log Probabilities: The use of log probabilities helps with numerical stability, especially when dealing with very small probability values.

1. Log Probability of Spam:

$$\log P(\text{spam}) + \sum x_i \log P(\text{word}_i \mid \text{spam}) + (1 - x_i) \log(1 - P(\text{word}_i \mid \text{spam}))$$

The first term, log P(spam), is the prior probability of the message being spam.
The second term calculates the contribution of each word that is present in the message (x_i), considering its conditional probability given spam. The third term considers the words that are absent, applying the probability of not seeing the word given spam.

2. Log Probability of Ham:

$$\log P(\text{ham}) + \sum x_i \log P(\text{word}_i \mid \text{ham}) + (1 - x_i) \log(1 - P(\text{word}_i \mid \text{ham}))$$

The first term, log P(ham), is the prior probability of the message being ham.
The second term calculates the contribution of each word that is present in the message (x_i), considering its conditional probability given ham. The third term considers the words that are absent, applying the probability of not seeing the word given ham.Similar to the spam probability, but it uses the prior and conditional probabilities for ham messages.

Final Decision: The prediction returns '1' if the calculated log probability for spam is greater than that for ham, indicating that the message is classified as spam. Otherwise, it returns '0', indicating ham.

Result

Final 5-fold cross validation accuracy was around 96%.

Model_2 Ensemble Learning

Feature Extraction

The preprocessing steps mentioned above result in a cleaner dataset, which is then transformed into features using the CountVectorizer(). CountVectorizer converts text data into a matrix of token counts, where each unique word in the dataset becomes a feature. It represents each document as a vector, with values showing the frequency of each word in that document.

Design of a Weak Classifier

For ensemble learning, we use a weak classifier that can perform slightly better than random guessing, typically with accuracy just above 50%. In this case, the weak learner identifies the top 25 "fishy" words—words that are highly indicative of spam. We calculate the "fishiness" by finding the words with the highest ratio of P(word|spam) to P(word|ham). The classifier makes predictions based on whether any of these top fishy words are present in a message.

The average 5-fold accuracy was around 70% roughly. The top 25 fishy words were found out to be:

['td', 'nbsp', 'width', 'pills', 'computron', 'href', 'height', 'viagra', 'font', 'xp', 'src', 'cialis', 'soft', 'meds', 'paliourg', 'php', 'bgcolor', 'oo', 'voip', 'drugs', 'biz', 'mx', 'img', 'photoshop', 'valign']

Final Ensemble Model

In this model, AdaBoost is used to create an ensemble of weak classifiers. The algorithm iteratively improves the model by focusing on misclassified samples, adjusting the weights of each sample based on prediction accuracy.

1. **Initialization**:
   Each training instance is assigned an initial weight w[i]. Initially, all instances have equal weight.
2. **Iterative Training**:
   For each classifier in the ensemble:

- Calculate predictions.
- Determine errors, and compute the classifier's weight (alpha) based on accuracy.
- Increase the weights of samples that were misclassified, making them more likely to be selected in the next iteration.
  Weight Update Formula:
  w[i] *= np.exp(-alpha * (predictions[i] == y.iloc[i]))
  Updates weights to penalize misclassifications. If a prediction is incorrect, the term (predictions[i] == y.iloc[i]) is False (0), leading to a larger weight increase. Misclassified samples end up with higher weights, ensuring that the next classifier will focus more on these instances.

3. **Prediction**
   For final predictions, each classifier votes on the class, weighted by its accuracy (alpha). The ensemble aggregates predictions:

- Each classifier's prediction is converted to 1 for spam and -1 for ham.
- clf_preds accumulates these predictions across classifiers, weighted by their confidence (alpha).
- y_pred sums these weighted votes. If the final score for an instance is positive, the prediction is "spam" ('1'); otherwise, it is "ham" ('0').
  This final ensemble approach combines the strengths of each weak classifier, providing a robust and accurate spam detection model.

Prediction

Unfortunately, the 5-fold accuracy was around 44%. Its incredibly low because only 50 classifiers could be trained due to limitation of time and computation power.

Conclusion

Since NB classifier gave better results the final model was trained on an extended using NB and saved as a pickle file naivemodel.pkl. Run the execute.py file to make the predictions for test dataset.