Report 7

1. Data Preparation
    a. Storage strategy using ElasticSearch or an alternative

- In my code, Elasticsearch serves as a central component for managing email data, which involves processing, storing, and querying vast amounts of emails for spam detection.
- I start by initializing an Elasticsearch instance and creating an index called "emails." This index is configured with specific mappings that are tailored to optimize the handling of email data. The fields specified in the index include a unique identifier, the email text, spam classification, and data split (training or testing). This structured indexing supports efficient querying and data retrieval. I use the helpers.bulk function for efficient bulk data ingestion into Elasticsearch. This approach is effective for loading large datasets, which are common in email processing applications. Each email's content, along with its spam label and whether it is part of the training or testing set, is indexed for rapid access.
- I employ the mtermvectors API to retrieve term vectors for the emails. Term vectors are essential for text analysis as they provide detailed information about the frequency of terms within each document.
- Using the term vectors, I extract features based on predefined spam-related words. This step involves scanning through the indexed data to identify and quantify occurrences of these terms across the dataset

2. Feature Extraction and Model Training
    a. Manual Spam Features

- Load a list of "spam-related words" from a file. These words are treated individually as features, similar to unigrams. The file is read, and each word is converted to lowercase and stored in a list. This list represents the set of features (i.e., unigrams) that are specifically looked for within the emails indexed in Elasticsearch.
- Features are fetched from Elasticsearch based on the presence of these spam-related words within the email texts. A search query is executed for each word in the spam-related word list, and the response includes aggregation results that count occurrences of these words by document ID (_id). This provides a sparse representation where each document's features are essentially the count of how many times each spam-related word appears.

    b. All Unigrams as Features

- Utilizes a CountVectorizer to transform the email texts into a matrix of token counts. This vectorizer handles the creation of a full unigram model from the email text data. The CountVectorizer is configured to recognize words (tokens) separated by word boundaries, effectively creating an unigram feature set from the entire corpus of email texts.
- After vectorization, the email texts are transformed into a sparse matrix representation where each row corresponds to an email and each column to a unigram. This matrix is highly sparse, meaning most entries are zeros, which is typical in text data where only a small subset of the possible words (features) appear in each document. The sparse

matrices are stored using SciPy's sparse matrix facilities, ensuring that the data consumes less memory and can be processed more efficiently.

c. Description of the machine learning algorithms used for training

- Linear Regression
- In the SpamClassifier class, LinearRegression is utilized presumably for its simplicity and as a baseline. Given that linear regression is not typically suitable for classification tasks, it's configured to threshold predictions to classify emails into spam or non-spam.
- Naive Bayes
- Both classifiers use MultinomialNB, a version of Naive Bayes suitable for feature vectors that represent counts or frequency data. This algorithm is particularly effective for text classification tasks involving discrete feature sets like word counts.
- Decision Tree
- DecisionTreeClassifier is used in both classes, providing a model that can handle non-linear relationships between features and the target variable. Decision trees are interpretable and work well for binary classification tasks such as spam detection.
- Logistic Regression
- In TextClassifier, LogisticRegression is used instead of Linear Regression for the binary classification task. It models the probability that an email is spam, providing a direct probability score that can be thresholded to make classification decisions.

3. Testing and Evaluation

a. Approach taken for testing the model on the test dataset

- Both the SpamClassifier and TextClassifier classes divide the data into training and testing datasets. In the SpamClassifier, the data frame is explicitly split using the sample method for 80% of the data as the training set and the remaining 20% as the test set. In the TextClassifier, a similar split is performed where each entry is labeled as 'train' or 'test' depending on its position in the dataset.
- Each model (Linear Regression, Naive Bayes, and Decision Tree) is trained on the training dataset. This involves fitting the model to the training data features and the corresponding labels.
- After training, predictions are made on the test dataset. For SpamClassifier, the predictions are generated using the predict method of each trained model. If Linear Regression is used, the continuous output is thresholded to convert it to binary labels (1 for spam, 0 for not spam).
- In TextClassifier, each model predicts the labels for the test dataset, and for models supporting probability outputs, the predict_proba method is used to obtain the probability scores for each instance being classified as spam.

b. Analysis of results from different algorithms

- **Results for Text Classifier**

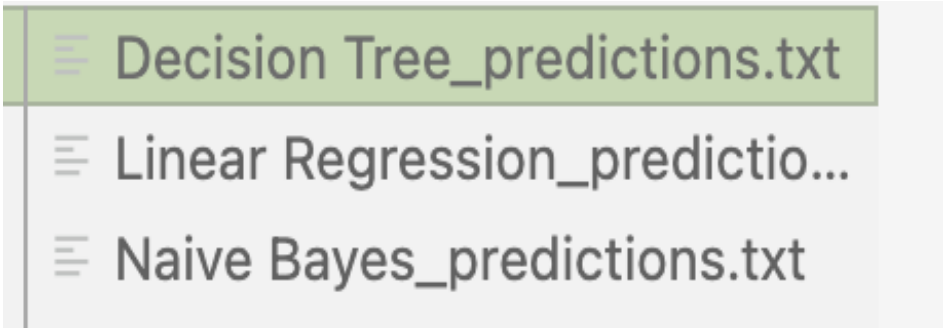
- The list of the top 10 important features, with negative coefficients, suggests these words are strong predictors for non-spam (ham) emails. Words like "mailing," "unsubscribe," and "wrote" typically appear in legitimate communications and newsletters, which the model has effectively learned as indicators of ham. The model has excellent performance metrics, with a precision of 0.99 and a recall of 0.98 for spam, and slightly lower for ham at 0.93 and 0.95, respectively. This indicates a high accuracy in identifying both spam and ham emails without many false positives or false negatives.

```
Top 10 important features:
mailing: -1.5012938740113257
wrote: -1.4571078128669122
unsubscribe: -1.1683741628050148
perl: -0.9699124178243719
ticket: -0.8734954513935276
applied: -0.8603514459986001
list: -0.8513926322068552
cmquzxhllg0kvghligf0dgfjag1lbnqgy29udgfpbmvkihrozsbucm9qyw4uugfja2vkljez: -0.7852925139380983
ihrocmvhdc4: -0.7852925139380983
tm9ydg9uieludgvybmv0ifnly3vyaxr5ihjlbw92zwqgdghligf0dgfjag1lbnqgug9zdgnh: -0.7852925139380983
Classification Report:
              precision    recall  f1-score   support

         ham       0.93      0.95      0.94       453
        spam       0.99      0.98      0.98      1547

    accuracy                           0.97      2000
   macro avg       0.96      0.97      0.96      2000
weighted avg       0.97      0.97      0.97      2000

ROC AUC Score: 0.9867806521487863
```



```
id,actual,predict,probability
8001,spam,spam,1.00000
23870,spam,spam,1.00000
23868,spam,spam,1.00000
23865,spam,spam,1.00000
23864,spam,spam,1.00000
23863,spam,spam,1.00000
23860,spam,spam,1.00000
23859,spam,spam,1.00000
23858,spam,spam,1.00000
23856,spam,spam,1.00000
23855,spam,spam,1.00000
23854,spam,spam,1.00000
23853,spam,spam,1.00000
23852,spam,spam,1.00000
23850,ham,spam,1.00000
23849,spam,spam,1.00000
23848,spam,spam,1.00000
23847,spam,spam,1.00000
```

- **Results for Email Spam**

- Linear Regression shows the poorest performance among the models with an overall accuracy of 82%.
- Naive Bayes similar to Linear Regression, Naive Bayes shows a high overall accuracy (81%)

- Decision Tree shows a better balance in classifying both spam and ham with an accuracy of 83%.

```
Results for Linear Regression:
              precision    recall  f1-score   support

           0       0.74      0.16      0.26       397
           1       0.83      0.99      0.90      1603

    accuracy                           0.82      2000
   macro avg       0.78      0.57      0.58      2000
weighted avg       0.81      0.82      0.77      2000

Results for Naive Bayes:
              precision    recall  f1-score   support

           0       0.57      0.15      0.24       397
           1       0.82      0.97      0.89      1603

    accuracy                           0.81      2000
   macro avg       0.70      0.56      0.57      2000
weighted avg       0.77      0.81      0.76      2000

Results for Decision Tree:
              precision    recall  f1-score   support

           0       0.62      0.34      0.44       397
           1       0.85      0.95      0.90      1603

    accuracy                           0.83      2000
   macro avg       0.74      0.64      0.67      2000
weighted avg       0.81      0.83      0.81      2000
```

4. Results and Discussion
   - Further investigate the feature importances, particularly those contributing negatively to the predictions, and assess whether additional features (e.g., email headers, time of day sent) could enhance the model's predictive power. Refine the preprocessing of text to possibly include bi-grams or tri-grams, which might capture more context than unigrams alone

-