Team Name: Scam Detectors (Project 2 Team 8)

Team Members+GitHub Usernames: Luna Prado(lunagpprado), Joshua Yoon(joshyoon5), Cole Manning(cmanning75)

GitHub Link: https://github.com/lunagpprado/scam_detector_p2

Link to Video: https://youtu.be/h3LSdqz3zlE?si=sXWfTcNqDokjVfjS

Extended and Refined Proposal

Problem:

Spam/Scam Emails have been an issue every single email-user has encountered. Either spam/scam detection is insufficient in detecting hazardous and phishing emails, or they end up flagging emails that may be important (this occurs with verification codes often). We're trying to solve the problem of scam detection.

Motivation:

Spam emails are hard to detect by hand and are extremely detrimental to those vulnerable to scams. Elderly, children and regular individuals all fall victim to a scam at least once in their life. A scam detector would protect these individuals from the bad actors from using emails to steal personal information and get access to individuals' money. A robust scam detector would help to mitigate the damage that a scam can take on an individual.

Features:

We implemented a UI capable of receiving input sender, receiver, email subject, and email body data, and projecting the likelihood of it being a scam or not and if the chosen model is projecting that the data is a scam or not. To choose between the two models available there's two buttons to do so. We know that the problem is solved once our model gains an accuracy of around 95% for detecting scam/spam emails.

Description of data:

We used multiple csv's from https://tinyurl.com/mr3fj4c7, which amount to around 80,000 data points of emails. Some of the datasets had variables for presence of url, subject line, sender, and receiver. However all of the datasets contained body text, and a respective scam or not scam label feature.

Languages:

For the project overall we decided on using Python for its readability/accessibility, data science tools, and that we were all familiar with Python.

Libraries/APIs/Tools:

UI: tkinter, threading, os

Bayes Theorem: Numpy, Pandas, Sklearn train_test_split, Feature Extraction TfIdf, accuracy score, classification report.

Random Forest: Numpy, Pandas, Sklearn train_test_split, Sklearn accuracy_score, Sklearn TfIdfVectorizer, Sklearn Truncated SVD, Sklearn Standard Scaler.

Algorithms implemented:

Naive Bayes Theorem: Uses Bayes Theorem to calculate the probability of a scam given words within the email data.

Random Forest: Uses the Random Forest algorithm to create randomly sampled Decision Trees to come to a vote on scam classification.

Additional Data Structures/Algorithms used:

Naive Bayes Theorem: We used the Naive Bayes algorithm to help calculate the probability of spam emails based on the words being used. This was a good choice as it is a very light and fast algorithm, it outputs a clear spam probability percentage, and gives us good probability readings based on certain word patterns.

Random Forest: We used three Sklearn algorithms to be able to process the input text data into numerical data: **TfIdf:**Used to change text data into a numerical vector. The resulting vector represents the importance of each unique word within the corpus of documents (in this case email text). **SVD:** Used to lower the high dimensionality sparse matrix from tfidf. **Standard Scaler:** Standardized the distribution of the data into a standard normal distribution for easy interpretation.

Distribution of Responsibility and Roles:

1. **UI/Main:** Cole Manning
2. **Naive Bayes:** Cole Manning (Josh Yoon Floating)
3. **Random Forest:** Luna Prado (Josh Yoon Floating)

Analysis

Any changes the group made after the proposal? The rationale behind the changes:

1. **Distribution of Roles:** Initially we wanted to try have everyone participating in every algorithm however we ended up going for a more efficient approach with Cole leading Main/UI and Naive Bayes, and Luna leading Random Forest with Joshua Yoon going between the algorithms to help and work on them with the respective team members.
2. **Decision Tree to Random Forest:** Originally Decision tree was going to be one of the algorithms however during the testing process it ended up being very slow to run for one big reason: it'd need to cycle through all the data points and all of the features each time. Which ended up being very inefficient and not that accurate. Random Forest however only takes a random sample of data points and a sample of variables, and is able to do multiple trees as a result providing efficiency and accuracy in one algorithm.

Big O worst case time complexity analysis of the implementation:

Naive Bayes:

The Naive Bayes algorithm analyzes the subject and body of emails through calculating the posterior provability. This is done by

1. **Feature Extraction:** The text dats is first transformed into a numerical TF-IDF matrix, similar to the random forest algorithm. The time complexity for this would be O(n * f) and the space complexity would be O(n * f).
2. **Probability Computation:** For each word, and each class, the algorithm computes conditional probabilities: P(word|class). The time complexity of this is simply O(n * f) since every feature of each email must be processed once to compute counts.
3. Prior Probability Calculation: The prior probability of each class is simply the fraction of samples in each class, so the time complexity for this would be O(n)
4. **Overall:** The overall time complexity would be O(n * f), making the Naive Bayes very efficient.

Random Forest:

The major function is the fit function on the Random forest which takes in training data and randomly samples from it to build a number of trees.

1. **Bootstrapping:** Randomly selects n rows from the input data with replacement. With n representing the number of rows in the data. It does this for t trees, with t representing the number of trees being used in the forest. O(t*n)
2. **Subspace:** Randomly selects sqrt(f) features to be used in each tree. With f representing the number of features(columns) in the input data. O(sqrt(f))
3. **Tree Building:** Each tree recursively builds a binary tree with decision, and leaf nodes. The recursion depth takes O(n) time to construct the tree in the worst case, but the inner best split calculations of each node take O(n). O(Sqrt(f)) is then multiplied as well because each tree is a subspace of the original columns. D representing maximum depth is also multiplied because in the worst case maximum depth(max height of the tree) is achieved. Which results in a total O(sqrt(f)*n^2*D) per tree.
4. **Overall Forest:** The forest has a total time complexity of O(t*sqrt(f)*n^2*D) with t representing the number of trees, f representing the number of features(columns) in the data, D representing the maximum depth(height) of each tree, and n representing the number of rows in the data.

Reflection

Group Experience:

At first we had some trouble organizing, but getting to learn from each other and do independent research on our algorithms before implementation helped us bring our ideas together in a knowledgeable way.

Group Challenges:

A significant challenge we faced was simply the placement of this project within the semester. Having it as a midterm project, but with a final-project-level scope brought difficulty with a barrage of higher level assignments and exams from other classes occurring concurrently.

Would you change anything in the workflow if done again:

Yes, specifically how we distributed the work. We should've had one person per role to simplify the workflow instead of having a floating role that could've been applied to a more dedicated role.

What Each Member Learned:

Cole - I was given an introduction and understanding on how to use, and implement a machine learning algorithm when developing Baye's algorithm to calculate the probability of certain words appearing in spam emails. Also developed a deeper understanding on how to create a cleaner UI with the Tkinter library in python.

Luna - I gained new experience in the basics of natural language processing from learning and applying tools such as Tf-Idf to interpret text data into something usable for the Random Forest. I also learned the importance of meetings matching the scope of a project. Where at first we had sparse shorter meetings before getting into longer, more frequent meetings as the project grew in complexity in the final days before the deadline.

Joshua - I learned some principles of machine learning and gained some experience on how to implement a new non trivial algorithm such as the decision tree. Researching and finding resources online about something completely new without going over it in class wasn't something I had to do a lot before, so I thought it was good for me to get that experience.

References

IBM Decision Tree: https://www.ibm.com/think/topics/decision-trees

IBM Random Forest: https://www.ibm.com/think/topics/random-forest

Krish Naik TfIdf Explanation: https://www.youtube.com/watch?v=D2V1okCEsiE&t=2s

Steve Brunton SVD Truncation: https://www.youtube.com/watch?v=9vJDjkx825k

Machine Learning for Everybody: https://www.youtube.com/watch?v=i_LwzRVP7bg