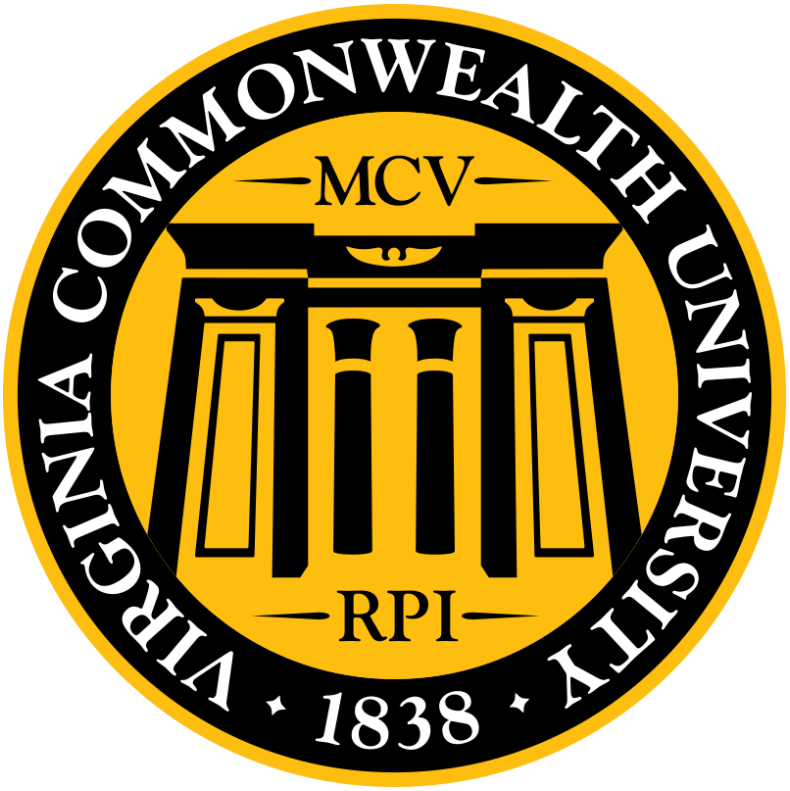
EMAIL CONTENT SENSITIVITY DETECTION

CAPSTONE SENIOR DESIGN 2019 - 2020



PREPARED BY

Bharat Venkat

Jeremy Bailey

Liam Fernando

Mohammad Malik

TABLE OF CONTENTS

Executive Summary

The exfiltration of sensitive information is when an authorized individual extracts data from a secured system and either shares it with unauthorized third parties or moves to an insecure system. Organizations like Capital One contain large amounts of high-value data that outside threats would be eager to obtain. To reduce the risk of data exfiltration, organizations must integrate security awareness and implement tools to prevent exfiltration from happening. The Virginia Commonwealth University Senior Design group proposes to create a tool that can scan the contents of an email to detect potential data exfiltration attempts.

The email anomaly scanning tool will be able to detect business related sensitive data in email and email attachments. Utilizing machine learning concepts such as Natural Language Processing (NLP), a field focusing on computer interactions with human language, the tool will be able to detect data exfiltration attempts more quickly and efficiently. Attachments like PDFs that are contained in emails will also be scanned to ensure that they do not contain any protected information. Image attachments specifically will be scanned using image scanning machine learning models that the tool utilizes.

To establish a baseline for further modeling, we construct a rule-based system using Regex to identify the PII that was injected. To build on this, we conduct further modeling by using popular feature extraction methods like Word2Vec and TF-IDF. We then transition to using a different version of the Enron dataset that has labels corresponding to business significance. We propose two labeling schemes for this dataset and use them for further modeling. We build multiple feature extraction, sampling, and modeling pipelines, and compare them across datasets. The best performing pipeline consists of counting extracted unigrams and bigrams then training a SVM classifier on these counts

Technical Volume

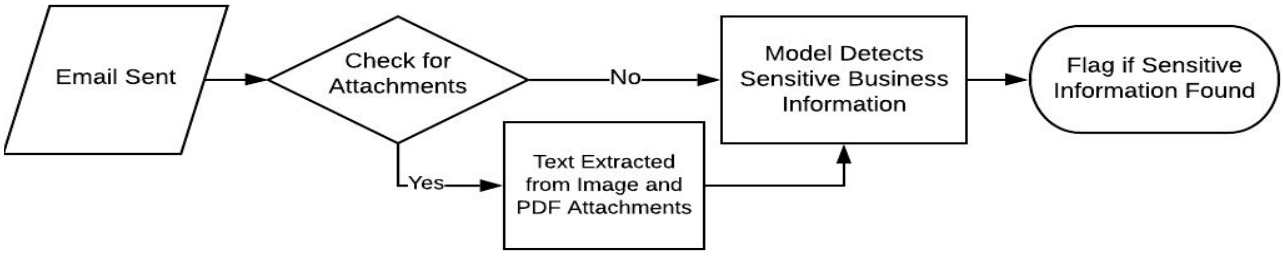
This Section covers the setup and detailed description of the tool. Each feature implemented in the tool is described in detail, providing code level understanding of how it was created. Please contact **Dr. Robert A. Dahlberg** - **dahlbergra@vcu.edu** for access to the project github that contains all the files.

Environment setup

The first task in building the email anomaly detection tool was to create a virtual environment and set up some programs that would be used to create our tool. Listed below are the programs that are required to be set up before creating the tool.

- Anaconda version 4.7.12
- Spark specifically PySpark inside Anaconda version 2.4.3
- Scala version 2.11.12
- numpy version 1.17.2
- python version 3.7.4
- pip version19.2.3
- pandas version 0.25.1
- jupyter
- scikit-learn
- flask=1.1.2
- tensorflow
- flask-dropzone version 1.5.4
- flask-uploads version 0.2.1
- werkzeug version 0.16
- nltk
- imblearn

Architecture Diagram



Dataset Creation

Initial Attempt

The dataset we built our model from was the Enron Email dataset. This dataset was widely used because it was a very large repository of real emails. An issue that we had with this dataset is that it has already been cleansed of sensitive information: what we wanted to predict with our model.

To process the dataset, we parsed each email into a DataFrame. DataFrames (both Pandas and PySpark) are easy to work with and process. Sensitive data was injected (using the Python Faker library) into selected emails. These emails were then labeled as Anomaly emails. We also prescreened the emails before injecting data to see if they already have sensitive content. If they do they were also labeled as Anomaly emails. Our dataset was balanced (even number of positive and negative samples). The relevant data that we decided to work with was data from the body of the emails (necessary because our injection of data was random: using other info like sender would have created false correlations that the model might pick up on).

Modified Approach

After consultation with Professor McInnes, we realized that our initial approach would not result in a dataset that we would be able to extract meaningful information from. Artificially injecting PII into data randomly does not provide learning algorithms with the patterns they need to make meaningful models.

We reevaluated our approach and decided that because datasets containing sensitive information as we had previously defined (containing PII) would not be publicly available that we would need to change our definition of sensitive. We changed our definition of sensitive to mean business sensitivity (discussing things such as meetings, etc), and were able to find a version of the Enron dataset with preexisting labels that aligned with our needs. Emails were grouped into the following categories:

1 Coarse genre

1.1 Company Business, Strategy, etc. SENSITIVE
1.2 Purely Personal NON
1.3 Personal but in professional context (e.g., it was good working with you) NON
1.4 Logistic Arrangements (meeting scheduling, technical support, etc) SENSITIVE
1.5 Employment arrangements (job seeking, hiring, recommendations, etc) SENSITIVE
1.6 Document editing/checking (collaboration) NON
1.7 Empty message (due to missing attachment) NON
1.8 Empty message NON

2 Included/forwarded information

2.1 Includes new text in addition to forwarded material NON
2.2 Forwarded email(s) including replies NON
2.3 Business letter(s) / document(s) SENSITIVE
2.4 News article(s) NON
2.5 Government / academic report(s) NON
2.6 Government action(s) (such as results of a hearing, etc) SENSITIVE
2.7 Press release(s) NON
2.8 Legal documents (complaints, lawsuits, advice) SENSITIVE
2.9 Pointers to url(s) NON
2.10 Newsletters NON
2.11 Jokes, humor (related to business) NON
2.12 Jokes, humor (unrelated to business) NON
2.13 Attachment(s) (assumed missing) NON

Additionally, each label assigned in this Enron dataset carried a weight. This weight described how many of the people that labeled a given email assigned it a given label.

We went through this list of labels and grouped each label into one of two categories: sensitive or non-sensitive. Labels above that are followed by SENSITIVE were grouped into the sensitive category while those that are followed by NON were grouped into the non-sensitive category.

Each labeled email had at least one category label assigned to it but also could possibly have more. For example, a given email may just have label 1.1 (Purely Personal), but another email may have labels 1.1 (Company Business, Strategy, etc.) and 1.2 (Purely Personal).

We realized that because emails may have multiple conflicting labels we had to come up with our own schemes to map these to a singular sensitive or non-sensitive label for each email. The first scheme we developed (Scheme 1) labeled each email based on the sum of sensitive versus non sensitive category weights. The category with the greater total weight was assigned to the email. The second scheme we developed (Scheme 2) labeled each email based on the presence of any label that belonged to the sensitive category. A short example is provided below to provide clarification.

An email has the following labels and weights:

1.1 (sensitive) with weight 2
1.6 (non-sensitive) with weight 3
2.4 (non-sensitive) with weight 1

Labeling Scheme 1:
Sum of sensitive weights = 2
Sum of non-sensitive weights = 4
Label email as non-sensitive

Labeling Scheme 2:
Contains a sensitive label (1.1)
Label email as sensitive

Both of these labeling schemes were applied to the labeled enron corpus to generate two distinct datasets to later be used for modeling.

Feature Extraction

Domain Specific

Email Details

Using PySpark we created User Defined Functions (UDFs) to extract features from our dataset such as, length of email, types of sensitive data, and feel of an email to name a few. Each feature will be described below and information on how the feature was implemented.

Length of an email using the built in len() function in python3

```
def extract_length_body(body):
    return len(body)
```

Number of lines in an email by looking for the count() function in python3

```
def get_line_counts(body):
    return body.count('\n')
```

Number of numeric characters by iterating through each character to see if it is a digit

```
def get_digit_counts(body):
    count = 0
    for char in body:
        if char.isdigit():
            count += 1
    return count
```

Number of alphanumeric characters by iterating through each character and checking with the built in isalnum() function in python3

```
def get_alphanum_counts(body):
    count = 0
    for char in body:
        if char.isalnum():
            count += 1
    return count
```

If there is a title in an email using the built in istitle() function in python3

```
def has_title(body):
    lines = body.split('\n')
    for line in lines:
        if line.istitle():
            return True
```

Regex Based Features

We also took elements from the rule based model described earlier, and incorporated each of the regex checks as a feature.

Sentiment Analysis

Sentiment Analysis is the analysis of text to determine what mood it is (positive, negative, neutral). We used the pretrained sentiment analysis model in popular library NLTK. We thought sentiment analysis would be useful to help determine if the email was sent by somebody that was particularly emotional.

Text Representations

TF-IDF

Term Frequency - Inverse Document Frequency (TF-IDF) is very popular in NLP tasks, and is a measure of how important a given word is to a document in a collection. We thought TF-IDF would be a good feature set to play with because it would give us information on specific words that possibly could indicate some type of exfiltration attempt.

Word2Vec

Word2Vec is a shallow neural network that can be used to create representations of the words in a given corpus. Word2Vec is useful because the embeddings created contain information about the relationships between words. To use for document classification the word vectors for all of the words in a document are typically averaged

Doc2Vec

Doc2Vec is another shallow neural network that can be used to create representations of text. However, instead of creating vectors that capture the relationships between words, the vectors instead capture the relationships between documents.

N-Grams

N-Grams is another common NLP technique used to extract meaningful phrases from documents. To use for document classification, sequences of length *n* are extracted and counted. An important decision when using n-grams is the choice of the stopword list used. We chose to go with Scikit-Learn's "English" stopword list.

Preprocessing

Parsing/Cleaning

For the emails in the enron dataset, many of them were forwarded emails containing header information that we thought might negatively affect the machine learning models. To deal with this we used Regex to parse and clean each of these emails.

An example of this header information below:

```
---------------------- Forwarded by Phillip K Allen/HOU/ECT on 10/09/2000 02:16
PM --------------------------- Richard Burchfield 10/06/2000 06:59 AM To:
Phillip K Allen/HOU/ECT@ECT cc: Beth Perlman/HOU/ECT@ECT Subject: Consolidated
positions: Issues & T  .......
to solve. Richard ---------------------- Forwarded by Richard Burchfield/HOU/
ECT on 10/06/2000 08:34 AM --------------------------- Allan Severude
10/05/2000 06:03 PM To: Richard Burchfield/HOU/ECT@ECT cc: Peggy Alix/HOU/
ECT@ECT, Russ Severson/HOU/ECT@ECT, Scott Mills/HOU/ECT@ECT, Kenny Ha/HOU/
ECT@ECT Subject:
```

(The "....." in the middle area was initially the entire email body that had been forwarded, but has been removed from the picture for brevity)

All the email bodies were parsed using lamda functions and regex patterns were ran to find these blocks of text in the email body, and if they were, they were removed.

SVD

Singular Value Decomposition is an algorithm used for dimensionality reduction. We experimented with SVD in multiple pipelines to see if we could shrink the space we had to model. We also used SVD to reduce our TF-IDF data to 2 dimensions before clustering using K-Means.

SMOTE

SMOTE (Synthetic Minority Oversampling-Technique) was used to oversample the minority class for each of the two datasets. SMOTE works by creating artificial samples that are on a line between two existing data points.

Removal of Tomek Links

Tomek Links occur when two points are each other's nearest neighbors and both have different class labels. The removal of Tomek Links can help limit the noise in data.

Modeling

Initial Modeling

We began the year using PySpark to model the data. PySpark has out of the box support for a number of classification algorithms. We tried Logistic Regression for its simplicity first: it is easily understandable and we thought it would be a good starting point. We also tried Gradient Boosted Trees (GBT). GBT is one of the more powerful classical machine learning algorithms, providing good results. This comes at a tradeoff at being very computationally expensive.

Regex Model

We also created a regex model to be able to determine if sensitive information is within our dataset. An example of what type of information is being detected by the regex can be seen below. This model served as a baseline for the machine learning model. We used pandas to create multiple series of each sensitive label and afterwards we would combine the series together to create a dataframe.

```
ssn_regex = df['Body'].apply(check_for_ssn)
print("ssn_regex done")
phone_regex = df['Body'].apply(check_for_phone_number)
print("phone_regex done")
email_regex = df['Body'].apply(check_for_email)
print("email regex done")
name_regex = df['Body'].apply(check_for_name)
print("name regex done")
code_regex = df['Body'].apply(check_for_code)
print("code regex done")
```

Each of these types of checks was done on the body of the email using lambda functions. These functions contain regex pattern matching to detect if a body of text contains a specific pattern. A detailed view on how the regex checks if an email contains a social security number is shown below.

```
def check_for_ssn(infile):
    if re.search(r"(?!000|.+0{4})(?:\d{9}|\d{3}-\d{2}-\d{4})", infile):
        return True
    else:
        return False
```

As the name implies, the following functions will detect phone numbers and emails respectively.

```
def check_for_phone_number(infile):
    if re.search(r"\d{10}|\d{3}-\d{3}-\d{4}", infile):
        return True
    else:
        return False
```

```
def check_for_email(infile):
    if re.search(r"\w{1,25}@\w{1,25}.com", infile):
        return True
    else:
        return False
```

We also wanted to be able to check for full names, this would be more difficult as there are many different names and a decent chance of false positive matches. We found a name-dataset containing over one hundred thousand first and last names to cross reference to. Using pip install we added this module to run on the body of text for full names.

```
def check_for_name(infile):
    body_text = infile.split()
    for x, y in enumerate(body_text):
        if m.search_last_name(y):
            if m.search_first_name(body_text[x - 1]):
                return True
    return False
```

After running the regex, the series are combined together into a dataframe and then the data frame is exported as a CSV file for reading.

```
df_copy = df_copy.assign(ssn=ssn_regex, phone=phone_regex, email=email_regex, address=address_regex, keywords=keywords_regex)
df_copy.to_csv('regex_output.csv')
```

Modified Approach

We transitioned to Scikit-Learn for our modeling half way through the year. We made this change because there was a larger selection of models for us to choose from, there were more parameters we could optimize (more flexibility), and there were other libraries that had scikit-learn support (gensim and imblearn). We also were familiar with Scikit-Learn and were able to iterate more quickly. During this process we switched to Pandas for data manipulation, moving away from PySpark for this as well.

K-Means

K-Means is a popular clustering algorithm. It works by attempting to fit “k” clusters to the data, assigning each data point to the closest cluster. The clusters are updated to the mean of all of the data points, and each point is reassigned to the nearest cluster. This process is repeated until the clusters converge. We used K-Means (along with Principal Component Analysis) to help visualize and identify any clear class distinctions in data.

Support Vector Machine

Support Vector Machines work by trying to separate points using a hyperplane. Points can be mapped into different spaces to help with this separation. We used a Gaussian kernel. SVM was the model we used in the majority of our experiments. We prefered it because we were very familiar with the algorithm and understood how to parameterize it to get good results.

Logistic Regression

Logistic Regression is typically a good starting point for classification tasks because it does not require much parameter modification to get reasonable results. We used it to get started.

LSTM

Long Short Term Memory or LSTM, is an artificial Recurrent Neural Network or RNN, known for performing well on sequence processing tasks. Unlike standard feedforward neural networks, it has feedback connections. It processes not only single data points but entire sequences of data. It is known to work well on NLP-related classifying tasks, handwriting recognition, speech recognition and anomaly detection.

We tried utilizing this deep learning architecture in an attempt to be able to better examine another alternative to the data prediction of sensitivity classification but we were unable to get meaningful results.

Experimental Setup

Parameter Search

We used Grid Search to find the optimal parameters for each of the models we used. We chose Grid Search because we were fairly comfortable with SVM and knew which parameters to tune.

Evaluation

Data was split into a 75% training set and a 25% test set. We had three different setups for evaluation: 5-Fold Cross Validation over the entire dataset, 5-Fold Cross Validation over the training set, and training on the training set then evaluating on the test set.

Metrics

Our primary metric was Matthews Correlation Coefficient or MCC, chosen because both datasets contained a significant class imbalance. We also looked at accuracy, AUC, and Precision-Recall Score to help validate our results.

Models

Setup

Our models were created using Pipelines. Pipelines are important because they ensure the integrity of training/testing splits while training a model and help prevent data leakage. Without using pipelines metrics can become artificially inflated because information about the test set can leak into the training set through any preprocessing steps. We used the Imblearn library's Pipeline class instead of Scikit-Learn's because the Imblearn correctly handles sampling steps (SMOTE, Removal of Tomek Links).

We wrote two helper classes, sklearn_framework.py and model.py, to help us iterate quickly. sklearn_framework.py contained the scaffolding code needed to do the parameter search and complete each of the three evaluation setups described above. model.py served as a wrapper class to take model configuration input from the notebooks in which we were creating/running the experiments, run the functions from sklearn_framework.py, then return run results and associated metadata.

We created model configurations by specifying the steps to be used in the modeling pipeline, the grid of parameters to search in our grid search, and the columns of the DataFrame to pass in as data for modeling.

Architectures

Listed below are some of the better performing final architectures we tested and the rationale behind them:

- TF-IDF, SMOTE, SVM
  - We used TF-IDF as a starting point. We tried adding SMOTE to oversample the minority class to help with the class imbalance in both datasets

- TF-IDF, SMOTE, Removal of Tomek Links, SVM
  - We tried removing Tomek Links to help clear any easy to remove noisy data points

- TF-IDF, SVD, SMOTE, Removal of Tomek Links, SVM
  - We tried using SVD to shrink the space that needed to be modeled by the SVM

- TF-IDF, SMOTE, Sentiment Analysis, SVM
  - We tried integrating sentiment analysis as a feature (detect disgruntled employees writing angry emails)

- Cleaned Body, TF-IDF, SMOTE, Sentiment Analysis, SVM
  - We tried cleaning the body of the emails before modeling to get rid of some noise

- N-Grams, SVM
  - We tried using N-Grams to capture relevant phrases in the corpus that might indicate sensitive information

- Cleaned Body, N-Grams, Sentiment Analysis, SVM

- We tried cleaning the body of emails before extracting n-grams to avoid collecting noisy phrases

All of these model architectures were tested using both datasets, one using labeling scheme 1 and the other using labeling scheme 2

Research Findings

Initial Attempt

Our best performing model was the Gradient Boosted Trees trained on a combination of Regex features and custom extracted features. We evaluated our model using 5 fold cross-validation. We parameterized our model using a grid search. The optimal model had results of:

Test Area Under ROC 0.79991494991495
TP: 78
TN: 553
FP: 19
FN: 107
Sensitivity: 42.16216216216216
Specificity: 96.67832167832168
predictiveACC: 83.35535006605019
MCC: 0.4993564628245596

Modified Approach

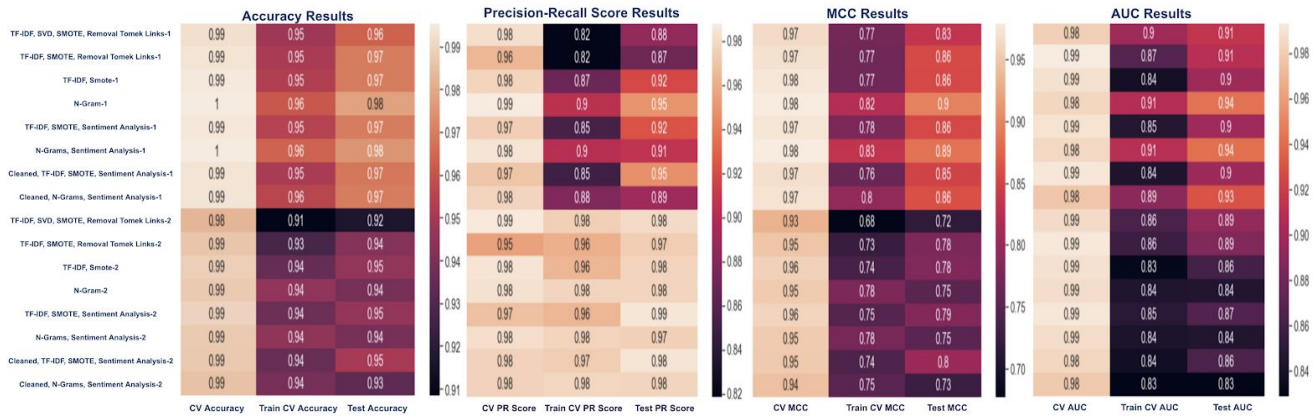
Labeling Scheme 1

The best model for Labeling Scheme 1 consisted of extracting N-Grams, then modeling using a Support Vector Machine. This model achieved an MCC score of .897.

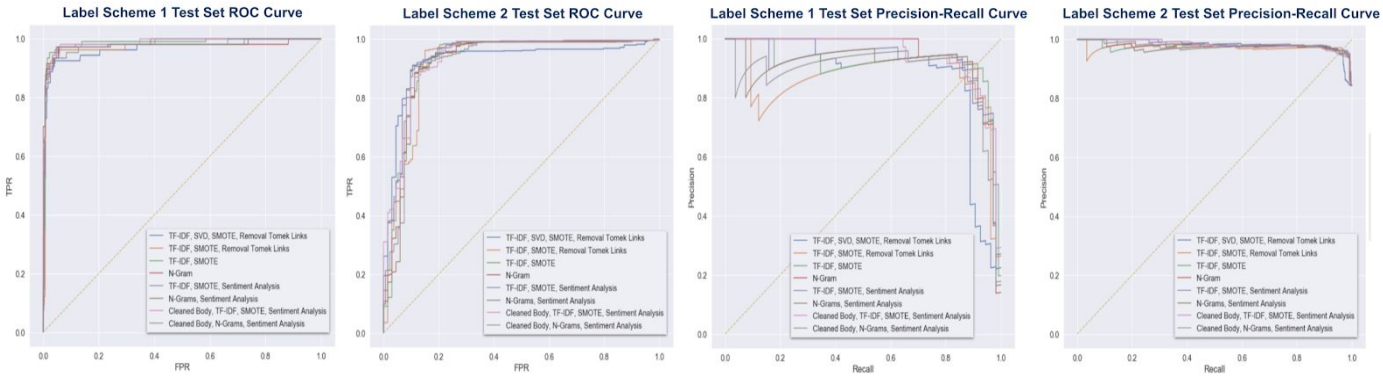
Labeling Scheme 2

The best pipeline for Labeling Scheme 2 consisted of cleaning email bodies, extracting TF-IDF and sentiment analysis features, oversampling with SMOTE, then modeling using a Support Vector Machine. This model achieved an MCC score of .802.

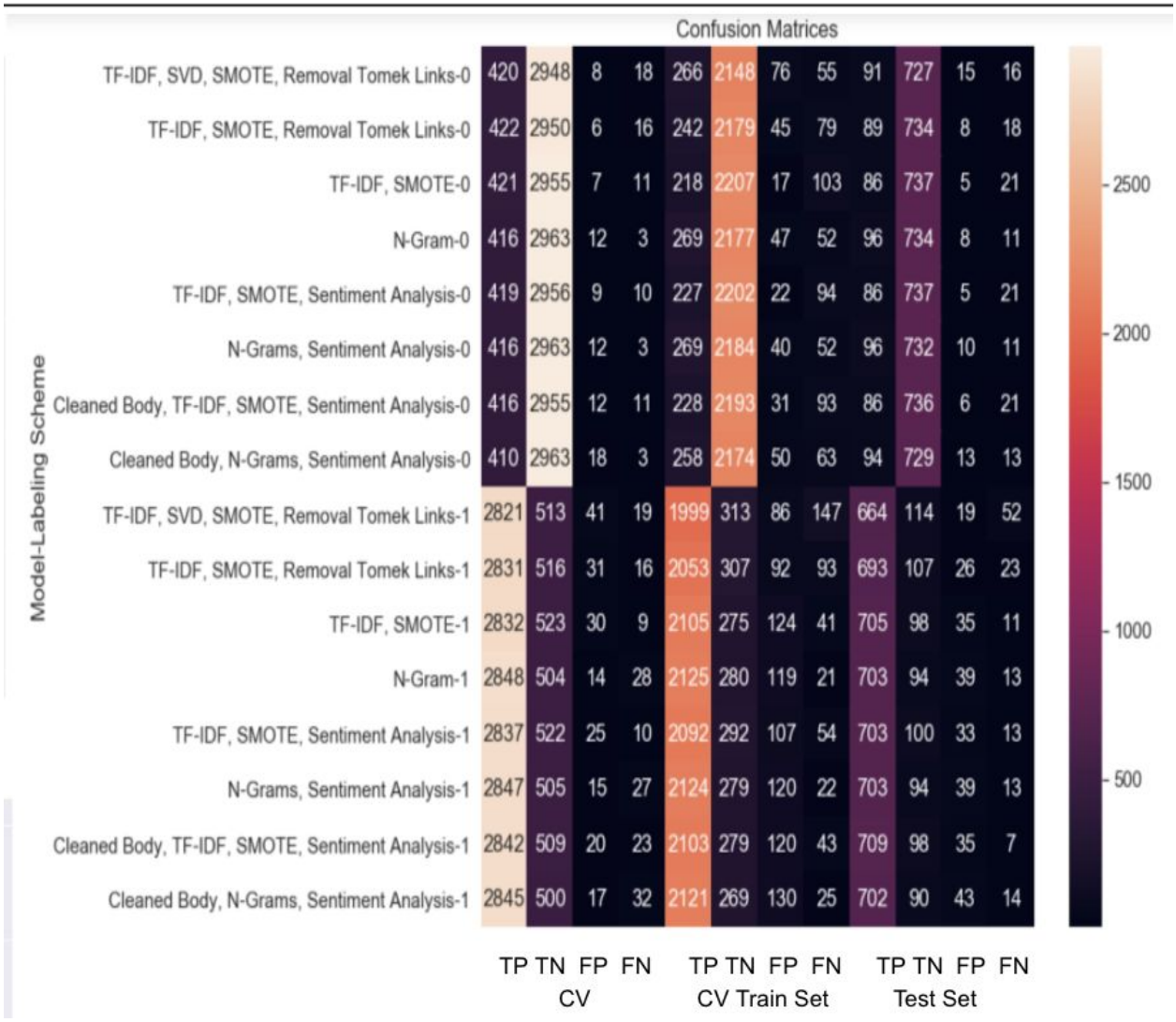
General Results Summary:



ROC and Precision Recall Curves:



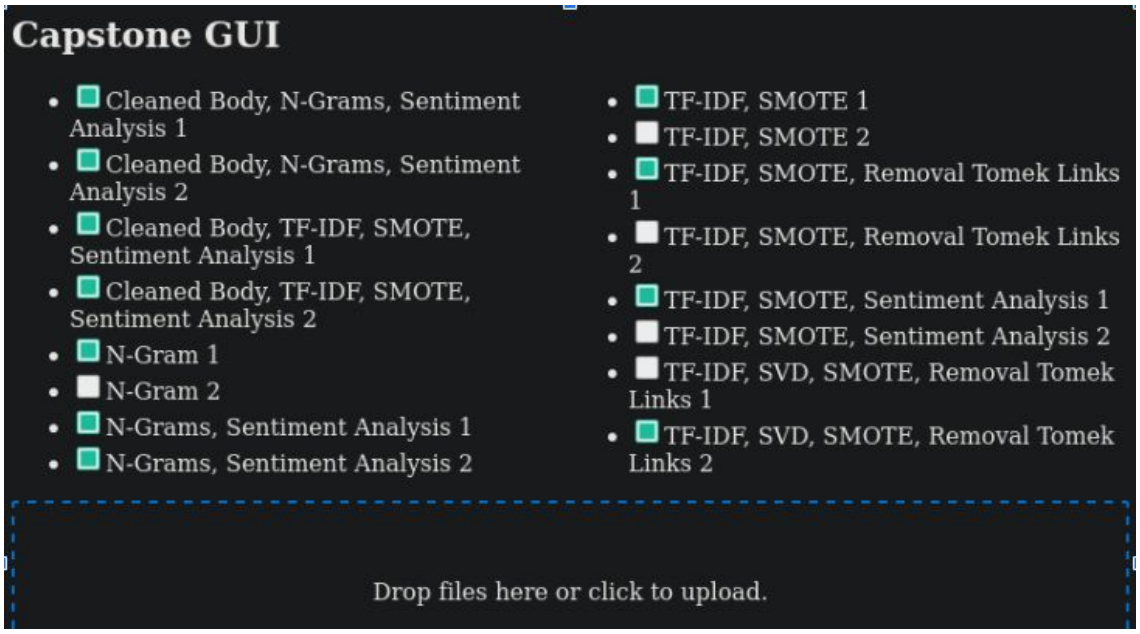
Confusion Matrices for Scikit-Learn Models



Entire results summary can be found at
https://github.com/VCU-CS-Capstone/2019-CS316-CapitalOne-Email-Content-Anomaly-Detection/blob/turnover/SRC/turnover/modeling/final_report.csv#L1

Graphical User Interface

Aside from having the tool be integrated into a company's existing security toolkit, we created a graphical user interface to provide a tool to demonstrate the capabilities of the tool. Using Flask, a micro web framework written in Python, we created a simple user interface that allows users to drop in images and PDF's. The images and PDFs will be converted into text and then will be sent to our model to detect if there is any sensitive material in them.



The GUI is a simple creation that was used to provide a demonstration of the sensitivity detection model at work and can always be improved in the future to provide additional features. A few of the biggest issues being that using ImageMagick and then Tesseract may be very lossy compared to a python library that does OCR on PDFs however none were able to be successfully utilized. Additionally, the models should be decided using HTTP form data, however it is currently hard-coded. The GUI can allow for a tool to act as a stand alone tool that does not rely on a company's built in security toolkit.

Project Tools

This section outlines the tools that we used during the course of the project to help save time. Each tool provides a description of what the tool is and how it was used during the course of the project.

Faker

A Python package that helps generate fake data. Using this package we were able to create sensitive data that our tool could find.

Jupyter Notebook

The Jupyter Notebook is an open-source web application that allowed us to create and share documents that contain live code, equations, visualizations, and narrative text. It was used throughout the development process to help visualize what parts of the code were doing so that members of the team could follow.

Pandas and Scikit-Learn

We started off using the Pandas python library for data manipulation. Once the data was formatted correctly we used the Scikit-Learn library to model the data.

Imbalanced Learn

We used imbalanced learn to oversample data using SMOTE and to undersample data by removing Tomek Links

PySpark

PySpark is the Python library for Apache Spark, a distributed computing platform. PySpark has built in support for working with massive data, as well as common data science functionality and algorithms. We used PySpark because the volume of data going to be fed into this tool is very large.

ImageMagick

ImageMagick is a unix command-line tool that is able to convert one image format to another. We used this tool to convert pdfs into image formats to be passed into Tesseract.

Tesseract OCR

Tesseract OCR is an optical character recognition engine with open source code. It uses artificial intelligence for text search and its recognition of images. It was released under the Apache License, Version 2.0, and development has been sponsored by Google since 2006. We used Tesseract to take in images to extract text. We also used PDFs that have been converted into images to extract text from them.

Flask

Flask is a Python web framework that we utilized in order to make the standalone demonstration gui. It utilizes all of the modeling pipelines as well as imagemagick and tesseract to produce a demo for the entirety of the file being passed in and determining sensitivity.

Keras

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow. It was designed to enable fast experimentation with deep neural networks and is focused on being user-friendly.