Ahmed Awadallah

Text Classification using Clustering and Naive Bayes

Introduction

Employees often use up a large portion of their work time parsing through and responding to important emails. Usually inboxes are also filled with another type of email called spam. The spam emails are generally emails that are sent to a large group of people to solicit something. Spam can be parsed through but doing this manually takes precious time and concentration out of someone's day. Occasionally spam emails can be much more malicious and contain code or links that can cause damage to the computer or company as a whole. Thus there is a vested interest in developing tools that can sift through emails automatically and discard those that are spam while letting the real emails through.

For this case study we are using the spam assassin data. It is a collection of 9348 emails where some are spam and some are not. The goal is to develop a classification model that can distinguish these two classes.

Methods

The spam assassin dataset is a collection of 9348 that are distributed among various folders. To collect that data the files all must be read and recorded within the program using an os walk. Some emails are not simply text but contain a collection of email components. These are a variety of these email types but they all fall under multipart email. As these multipart emails are connected the decision we made was to consider multipart emails as one larger email. This means that for multipart emails we simply combine all the body text of each partner email into one string.

No text preprocessing was performed on any email body. This may look lazy but it appears the tools from sklearn (the countvectorizer) seem to automatically do this. If we are somehow wrong about this an additional reason is that the model performance later is impressive.Because the model performance is good I'd argue against over engineering the problem. If later with new data the model performs significantly worse then this decision should be revisited.

The text for each email contains information about each email. For this project we will be using a bag of wards approach to derive meaning from the text. The approach we chose was a TfidfVectorizer as this will help account for words that are frequent but don't help identify a document. For example the word "the" is already used quite a few times in this case study, it is not likely the word "the" would be helpful in identifying this text as belonging to a case study. The same logic is being applied to the emails.

In addition to using the TfidfVectorizer we also set the hyperparameter min_df to 0.002. The min_df hyperparameter lets the vectorizer ignore words below a certain frequency threshold. We set this hyperparameter as incredibly rare words are not generalizable to new emails and thus shouldn't be included. For example the word "thisiscasestudy3forsmu" is unlikely to appear in any other text and could be used to identify this document as a case study. At the same time since no other document would have it would only be useful for this specific case study and thus not generalizable. Our choices lead to the bag of words to contain 8660 instead of 123781.

We chose 0.002 through an inspection method. We would lower the threshold and inspect the words that were recorded. As soon as we saw the appearance of words that were absurd or nonsensical we would revert to the last value we tried. An example of a word that lead to us deciding to not use 0.001 was "dqoncjxozwfkpg0kpg1ldgegahr0cc1lcxvpdj0iq29udgvudc1myw5ndwfnzsigy29u".

Once we had our word frequency feature matrix we used this as input into a KMeans model that expected two clusters. Two clusters was specified as this problem is a binary problem where we are predicting spam and not spam.

The clusters outputted by the KMeans model were added to the training set as an additional feature now leading us to have 8661 features.

This larger feature set was then used in a MultinomialNB model. Spam will be considered the positive class. We optimized the alpha hyperparameter using 5 fold cross validation and f1 score as an optimizing metric.

F1 score was chosen as we wanted a balanced performance of predicting an email as spam when it's not and predicting an email as spam when it is. It was not specified what the business wanted and there is no clear reason to pick one over the other as the cost is about equal. Misclassifying a couple real emails leads to confusion and frustration at a lack of a response. Misclassifying a couple spam emails leads people to manually check the emails for spam occasionally. This has an added risk of malware or social engineering to occur. Both situations are not ideal but it's hard to say in general for all cases we prefer one over the other. This is why we opted for the balanced approach.

Finally when the optimal alpha was chosen a final model was made. Analysis of the models performance was done through a classification report, confusion matrix, and roc curve.

Results

The following is the results of the KMeans clustering on the data. While this is not a model in and of itself it's interesting to see its performance in classifying the two classes

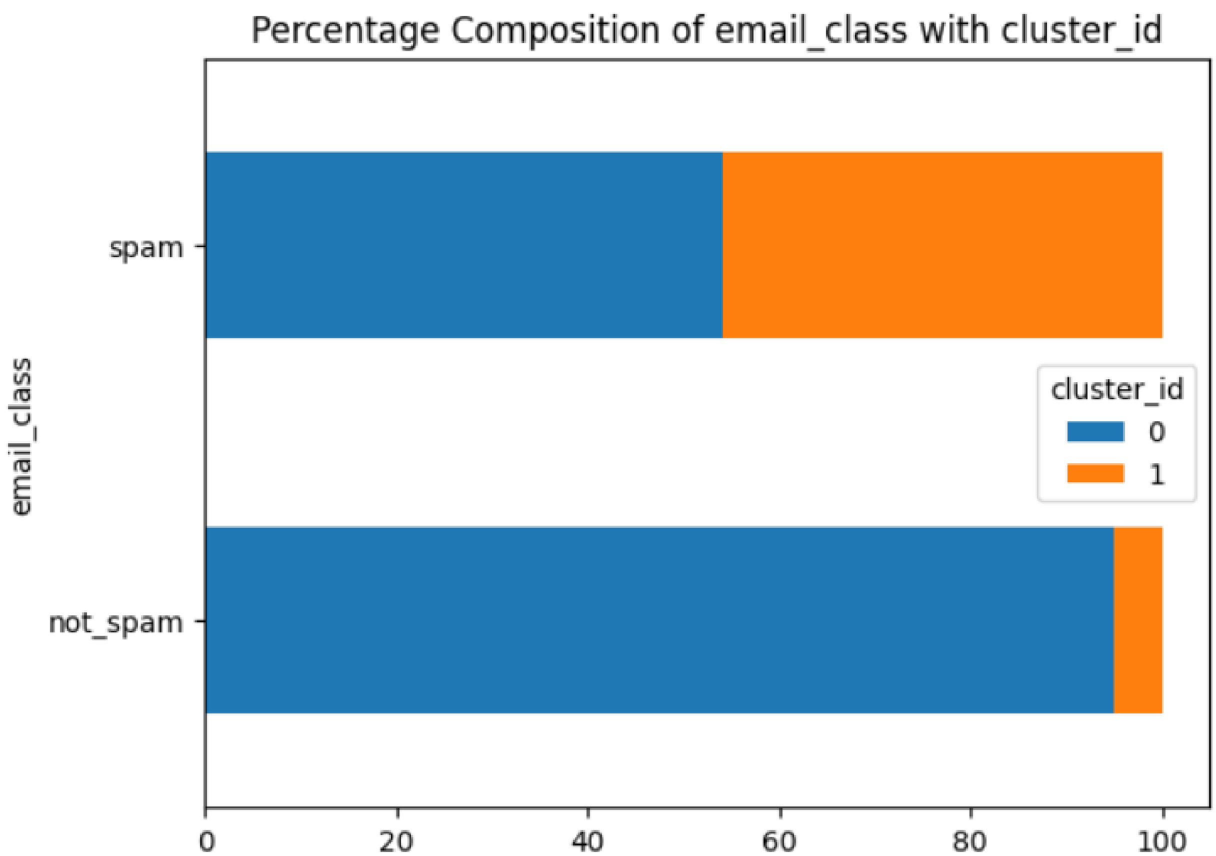

Figure 1: Plot showing how well clusters associated with the spam and not spam classes

Generally the clustering of KMeans did not do a great job separating spam and not spam as the two bars are mixed between the two cluster ids. What can be said is that cluster id 0 is likely associated with not spam and therefore cluster 1 is associated with spam. While we don't need to do this analysis to feed the clustering results into our next model it is still good to see some information about what it means to be spam versus not spam is contained within cluster id.

Remember that our final model is a Multinomial Naive Bayes model. The classification report of our final version of this model is displayed below.

Classification Report

~~~~~~

	precision	recall	f1-score	support
not_spam	0.98	0.96	0.97	6951
spam	0.89	0.94	0.91	2397
accuracy			0.95	9348
macro avg	0.94	0.95	0.94	9348
weighted avg	0.96	0.95	0.96	9348

Figure 2: Classification Report of MultinomialNB Model

As can be seen in the report the overall performance of the model is impressive. The two classes can easily be distinguished with almost 100% accuracy.
~~~~~~

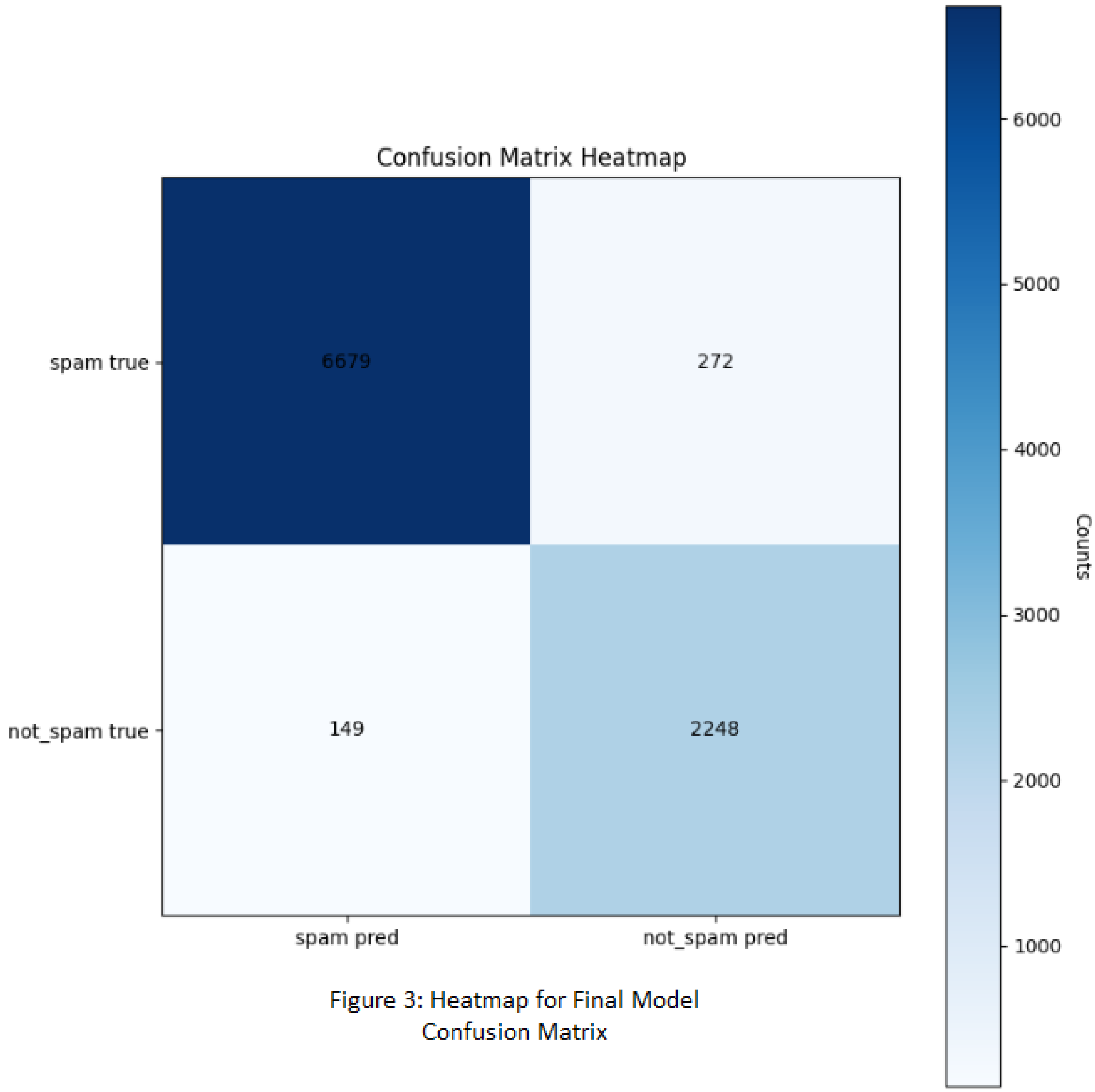

Figure 3: Heatmap for Final Model
Confusion Matrix

This confusion matrix provides further evidence of the strength of our model but from a different perspective. The truth labels and the predicted labels align well with each other. Only very few cases fail to be predicted properly for both classes. Since we decided a balanced optimizing approach these results are in line with that decision.

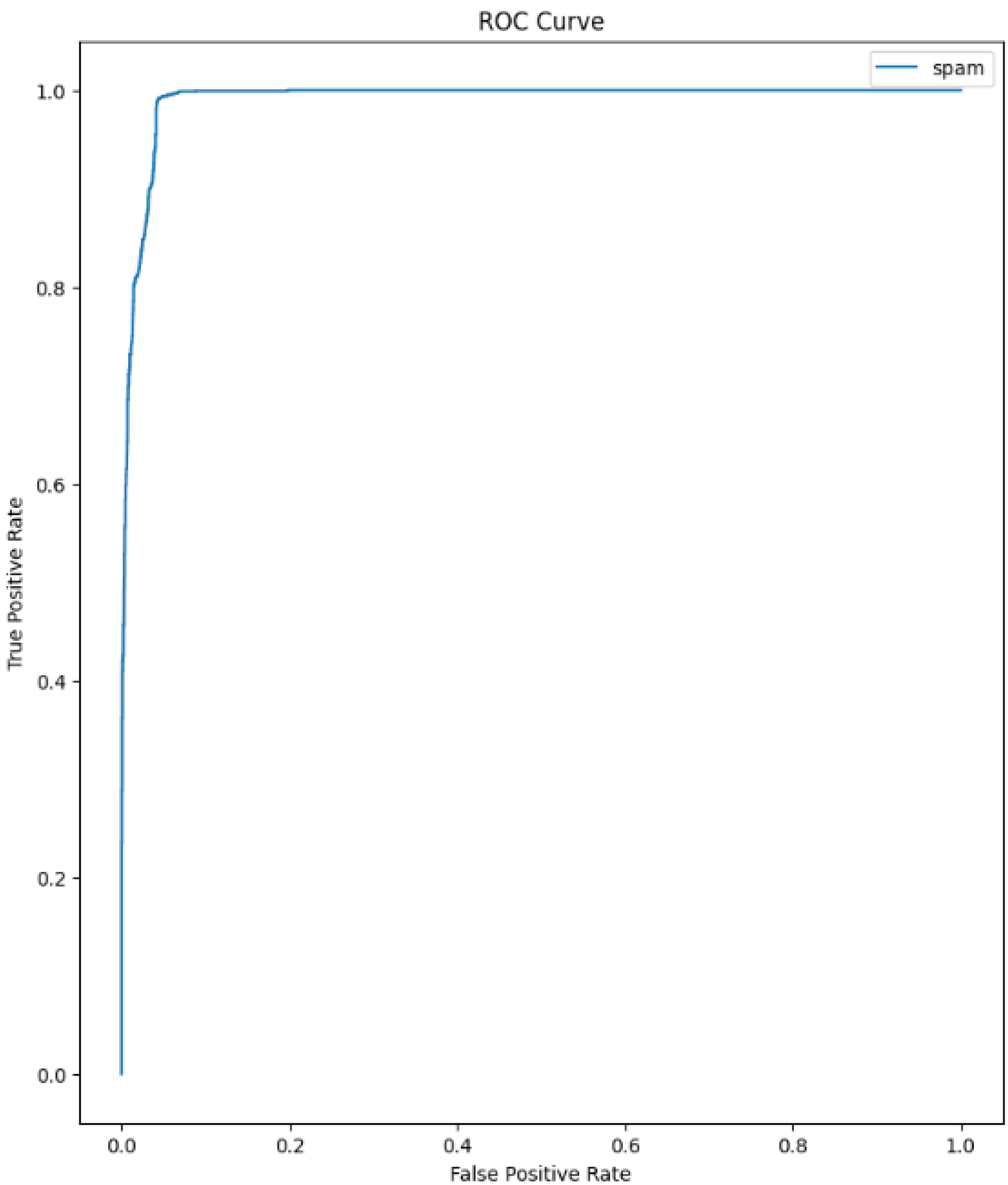

Figure 4: ROC of Final Model

```
AUC Report
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
spam AUC: 0.9906
```

Figure 5: AUC for Final Model ROC Curve

The ROC Curve provides further confidence in the performance of our model. Visually this is the curve of a good classifier but the numeric metric is also supplied for those who prefer that. These results are here to again show that this model does well but it is also here to provide some new context for an old question. Earlier we decided to use spam as the positive class and use f1 score as our optimizing metric. This was based on the idea that we were okay with both False Positives and False Negatives occurring at a close to equal rate. The above plot shows that there exists a threshold where we could in fact maximize the TPR while incurring a False Positive Rate Cost. Before it may not have been worth considering but now that we can show that it can be possible to choose one with minimal cost to the other may make this an attractive option to choose. Food for thought.

Conclusion

Overall through the use of bags of words, clusters, and naive bayes we were able to create a strong classifier for the spam assassin dataset. Further avenues of exploration remain in regards to what category we care about more and the costs of those decisions.

There is one loose end though. At the start of the analysis we decided to vectorize all the text data first and then perform analysis. This vectorized data formed our feature set. The question is was this the right choice? Technically by looking at all the text input information about the hold out set is present within the data even if the naive bayes model did not train on them.

Code

Appendix A

```
import os
import sys
import email
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn import metrics as mt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from scipy.sparse import hstack
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report


# Build a spam classifier using naive Bayes and clustering.
# You will have to create your own dataset from the input messages.
# Be sure to document how you created your dataset.

INVALID_INPUT = 1


class UnexpectedEmailFileStructureError(Exception):
    def __init__(self):
        msg = "This script expects a file path to a directory that contains "
        msg += "the spam assassin dataset. The spam assassin dataset is a "
        msg += "group of five folders with the names, easy_ham, easy_ham_2, "
        msg += "hard_ham, spam, spam_2. Did you specify a folder path that "
        msg += "contains these folders?"
        super().__init__(msg)

```

```python
class MissingInputError(Exception):
    def __init__(self):
        msg = "This script expects exactly one input, a file directory that "
        msg += "contains the spam assassin dataset. You input nothing. "
        msg += "Did you forget to add an argument with the script?"
        super().__init__(msg)


class ExcessiveInputError(Exception):
    def __init__(self):
        msg = "This script expects atleast one input, a file directory that "
        msg += "contains the spam assassin dataset. You called this script "
        msg += "with more than one argument. Only use one and try again. "
        super().__init__(msg)


def recursive_payload_retrieval(email_payload):
    email_body = ""
    for sub_email in email_payload:
        sub_email_payload = sub_email.get_payload()
        if type(sub_email_payload) is list:
            email_body += recursive_payload_retrieval(sub_email_payload)
        elif type(sub_email_payload) is str:
            email_body += sub_email_payload
    return email_body


def check_correct_file_structure(inputs):
    email_direc = inputs[1]
    expected_direcs = ["easy_ham", "easy_ham_2", "hard_ham", "spam", "spam_2"]
    first = True
    for root, dirs, files in os.walk(email_direc):
        if first:
            first = False
            for expected_direc in expected_direcs:
                if expected_direc not in dirs:
                    print(expected_direc)
                    print(dirs)
                    raise UnexpectedEmailFileStructureError
```

```python
def check_valid_inputs(inputs):
    if len(inputs) < 2:
        raise MissingInputError
    elif len(inputs) > 2:
        raise ExcessiveInputError


def load_emails(inputs):
    email_direc = inputs[1]
    data = {"email_body": [], "email_type": [], "email_class": []}
    for root, dirs, files in os.walk(email_direc):
        for f in files:
            file_path = os.path.join(root, f)
            with open(file_path, "r", encoding="latin-1") as email_file:
                if "cmds" in file_path:
                    continue
                body = ""
                msg = email.message_from_file(email_file)
                t = msg.get_content_type()
                payload = msg.get_payload()
                if type(payload) is list:
                    body = recursive_payload_retrieval(payload)
                elif type(payload) == str:
                    body = payload
                else:
                    print("Unexpected Condition")
                    raise Exception()

                data["email_type"].append(t)

                data["email_body"].append(body)

                if "spam" in root:
                    data["email_class"].append("spam")
                else:
                    data["email_class"].append("not_spam")
    return data
```

```python
def categ_categ_compare(df, categ1, categ2, file_name="no_name.png"):
    horizontal = 1
    vertical = 0
    df_count = df.groupby([categ1, categ2])[categ1].count().unstack().fillna(0)
    df_relative_count = df_count.div(
        df_count.sum(axis=horizontal),
        axis=vertical)*100
    title = "Percentage Composition of " + categ1 + " with " + categ2
    df_relative_count.plot.barh(stacked=True, title=title)
    plt.savefig(file_name, bbox_inches='tight')
    plt.clf()
```

```python
def multinom_nb_cv(x, y, print_progress=False, n_jobs=1):
    best_alpha = 0
    best_f1 = 0
    first = True
    iter_count = 20
    report_freq = int(iter_count/20)
    i = 0
    t1 = time.time()
    for param_alpha in np.logspace(-3, 3, iter_count):

        model = MultinomialNB(alpha=param_alpha)

        f1_cv = cross_val_score(model, x, y, cv=5,
                                scoring='f1')

        f1 = sum(f1_cv)/len(f1_cv)

        if first:
            first = False
            best_alpha = param_alpha
            best_f1 = f1
        else:
            if (f1 > best_f1):
                best_alpha = param_alpha
                best_f1 = f1

        if print_progress and ((i+1) % report_freq == 0):
            print(f"Iteration {i+1}. "
                  f"Percent done = {(i+1)/iter_count*100:.4f}%")

        i += 1
    t2 = time.time()
    elapsed_time = t2-t1

    report_str = f"Best f1={best_f1:.4f}, "
    report_str += f"alpha={best_alpha}, time={elapsed_time:.4f}s\n"

    return best_alpha, report_str
```

```python
def log_section(title="No Title", content="No Content"):
    log_file.write(title + "\n")
    log_file.write("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")
    log_file.write(content + "\n")
    log_file.write("\n")


def plot_heatmap(data,
                 xlab,
                 ylab,
                 size=9,
                 title="untitled heatmap",
                 cbar_label="untitled",
                 save_name="untitled.png"):

    fig, ax = plt.subplots(figsize=(size, size))
    im = ax.imshow(data, cmap=plt.cm.Blues)

    for i in range(len(ylab)):
        for j in range(len(xlab)):
            ax.text(j, i, data[i, j],
                    ha="center", va="center", color="black")

    # Show all ticks and label them with the respective list entries
    ax.set_xticks(np.arange(len(xlab)), labels=xlab)
    ax.set_yticks(np.arange(len(ylab)), labels=ylab)

    cbar = ax.figure.colorbar(im, ax=ax)
    cbar.ax.set_ylabel(cbar_label, rotation=-90, va="bottom")

    ax.set_title(title)
    fig.tight_layout()
    plt.savefig(save_name, bbox_inches='tight')
    plt.clf()
```

```python
if __name__ == '__main__':
    inputs = sys.argv
    check_valid_inputs(inputs)
    check_correct_file_structure(inputs)
    # Log Start
    log_file = open("log.txt", "w")
    log_file.write("Case Study 3 Report\n\n")

    # Use input to find and load emails
    email_data_raw = load_emails(inputs)
    email_df = pd.DataFrame(email_data_raw)

    # Id to class maps
    email_class_to_id = {"spam": 1, "not_spam": 0}
    id_to_email_class = {1: "spam", 0: "not_spam"}

    # TFIDF Vecorizer
    email_bodies = email_df["email_body"]
    vectorizer = TfidfVectorizer(min_df=0.002)
    features = vectorizer.fit_transform(email_bodies)

    # Cluster with Kmeans
    cluster_model = KMeans(n_clusters=2)
    cluster_model.fit(features)
    email_df["cluster_id"] = cluster_model.labels_

    # Plot to see which clusters refers to what email class
    categ_categ_compare(email_df,
                        "email_class",
                        "cluster_id",
                        "categorical_cluster_comparison.png")

    # add cluster feature to vectorizer features
    cluster_ids_raw = cluster_model.labels_
    cluster_ids_reshaped = cluster_ids_raw.reshape(-1, 1)
    features_and_cluster_ids = hstack((features, cluster_ids_reshaped)).A
```

```python
# Optimizing Alpha
email_id = email_df["email_class"].map(email_class_to_id)
best_alpha, report = multinom_nb_cv(features_and_cluster_ids,
                                    email_id,
                                    n_jobs=5,
                                    print_progress=True)
log_section(title="Optimization Report",
            content=report)

# Best Model
final_clf = MultinomialNB(alpha=best_alpha)
final_clf.fit(features_and_cluster_ids, email_df["email_class"])

# Classification Report
y_pred_final = final_clf.predict(features_and_cluster_ids)
y_true = email_df["email_class"]
classif_rep = classification_report(
    y_true,
    y_pred_final
)
log_section(title="Classification Report",
            content=classif_rep)
```

```python
# Confusion Matrix Heatmap
conf_mat = confusion_matrix(y_true, y_pred_final)

base_labels = list(id_to_email_class.values())
label_true = []
label_pred = []
for i in range(len(base_labels)):
    true_label = base_labels[i] + " true"
    pred_label = base_labels[i] + " pred"
    label_pred.append(pred_label)
    label_true.append(true_label)
plot_heatmap(conf_mat,
             label_pred,
             label_true,
             size=8,
             title="Confusion Matrix Heatmap",
             cbar_label="Counts",
             save_name="conf_heatmap.png")

# ROC Curve
y_pred_prob = final_clf.predict_proba(features_and_cluster_ids)
y_label = id_to_email_class[1]
preds = y_pred_prob[:, 1]
fpr, tpr, thresholds = mt.roc_curve(email_id, preds, pos_label=i)
plt.plot(fpr, tpr, label=y_label)
title_str = 'ROC Curve'
plt.title(title_str)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
auc_report = f"{y_label} AUC: {mt.auc(fpr, tpr):.4f}\n"
log_section(title="AUC Report",
            content=auc_report)
plt.legend()
plt.savefig("roc_plot.png", bbox_inches='tight')
plt.clf()
log_file.write("\n")
log_file.close()
```