

Predicting Gender with Email Body Text:

Binary Classification Supervised Learning with the Enron Email Dataset

James Bush

Springboard

Capstone I

TABLE OF CONTENTS

I.	TABLE OF CONTENTS	2
II.	DEFINING THE PROBLEM	3
III.	DATA COLLECTION	4
IV.	DATA WRANGLING	6
V.	EXPLORATORY DATA ANALYSIS	10
VI.	FEATURE ANALYSIS	13
VII.	MODEL IMPLEMENTATION, RESULTS	17
VIII.	CLOSING	22
IX.	REFERENCES	26
X.	APPENDIX	27

Defining the Problem

Can text contained in an email be used to predict the sender's gender? This project will perform a binary classification prediction on the Enron email dataset using an email's body text as the Feature data and gender labels as the Target data. Gender labels are collected by referencing names extracted from email addresses contained in the data. Ideal text data will be referenced as '*genuine*,' or likely written by the original email sender; '*non-genuine*' text is considered not-likely an original typed by the author. Examples of *non-genuine* text include: Copy-paste news articles, automatic reports, or reply/forwarded text. Once text data is collected and cleaned, it will be transformed using *Text Frequency-Inverse Document Frequency* (TFIDF) and input into classification algorithms for modeling. The models applied to the text will be Linear Support Vector Classification and Logistic Regression.

Python (3.8.4) is used to code the project. Pandas (1.0.4), Numpy (1.18.1), and Regex (2.5.80) libraries are used to manipulate data. Natural Language Toolkit (NLTK, 3.5) is used to process text data. Scipy (1.4.1) is used to draw statistical insights. Matplotlib (3.2.1) is used to visualize data, with some Seaborn visual touching-up. Multiple functions from Sci-kit learn (1.4.1) are used to explore features and model data.

Data Collection

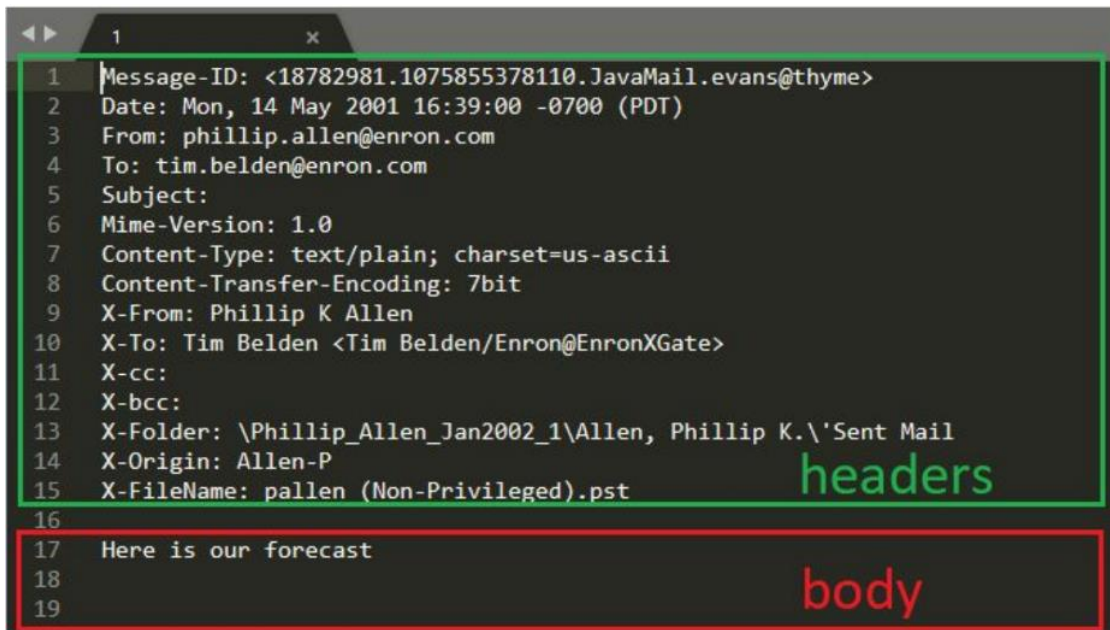
The dataset used in this project is the raw zipped dataset, *enron_mail_20150507.tar.gz*, downloaded from Carnegie Mellon University School of Computer Science (Cohen, 2015) on April 13th, 2020. It is sized ~1.78 GB in the zipped form. Cohen (2015) also links a few pointers to other work done with the dataset worth reviewing.

Once the zipped dataset is downloaded locally, the user function *file_grabber()* uses the *os* library to collect a list of file directories. The list of file directories is then used to collect all emails files across the folders. The directory is unstructured; it resembles what any collection of individual email libraries might look like: Each user having their own schema resulting in different naming, hierarchies, and many *duplicates*.

DataFrame. The resulting DataFrame contains 517401 entries with 19 data columns. 17 of the data columns are extracted from headers found in the email file text:

- Message-ID: Alpha-numeric code unique to each email.
- Date: Date email is sent.
- From: Sender email address, standard format: xxxxx@email.com.
- To: Recipient email address, standard format.
- Cc: List of *carbon copy* standard email addresses.
- Bcc: List of *blank carbon copy* standard email addresses.
- Subject: Email subject string.
- Mime-Version: Mime format version.
- Content-Type: Character set for email file contents.
- Content-Transfer-Encoding: Indicates what encoding, if any, has been done to the message body.
- X-From: Internal sender string.
- X-To: Internal recipient string.
- X-cc: Internal cc string.
- X-bcc: Internal bcc string.
- X-Folder: Local folder where email is located
- X-Origin: Storage folder owner, internal naming schema (first initial, last name mostly)
- X-FileName: Internal file name with extension.

Below is an example of the initial email file contents:



The image shows a screenshot of an email file's contents in a text editor. The editor has a dark background with a tab labeled '1' and a close button 'x'. The email content is as follows:

```
1 Message-ID: <18782981.1075855378110.JavaMail.evans@thyme>
2 Date: Mon, 14 May 2001 16:39:00 -0700 (PDT)
3 From: phillip.allen@enron.com
4 To: tim.belden@enron.com
5 Subject:
6 Mime-Version: 1.0
7 Content-Type: text/plain; charset=us-ascii
8 Content-Transfer-Encoding: 7bit
9 X-From: Phillip K Allen
10 X-To: Tim Belden <Tim Belden/Enron@EnronXGate>
11 X-cc:
12 X-bcc:
13 X-Folder: \Phillip_Allen_Jan2002_1\Allen, Phillip K.\'Sent Mail
14 X-Origin: Allen-P
15 X-FileName: pallen (Non-Privileged).pst
16
17 Here is our forecast
18
19
```

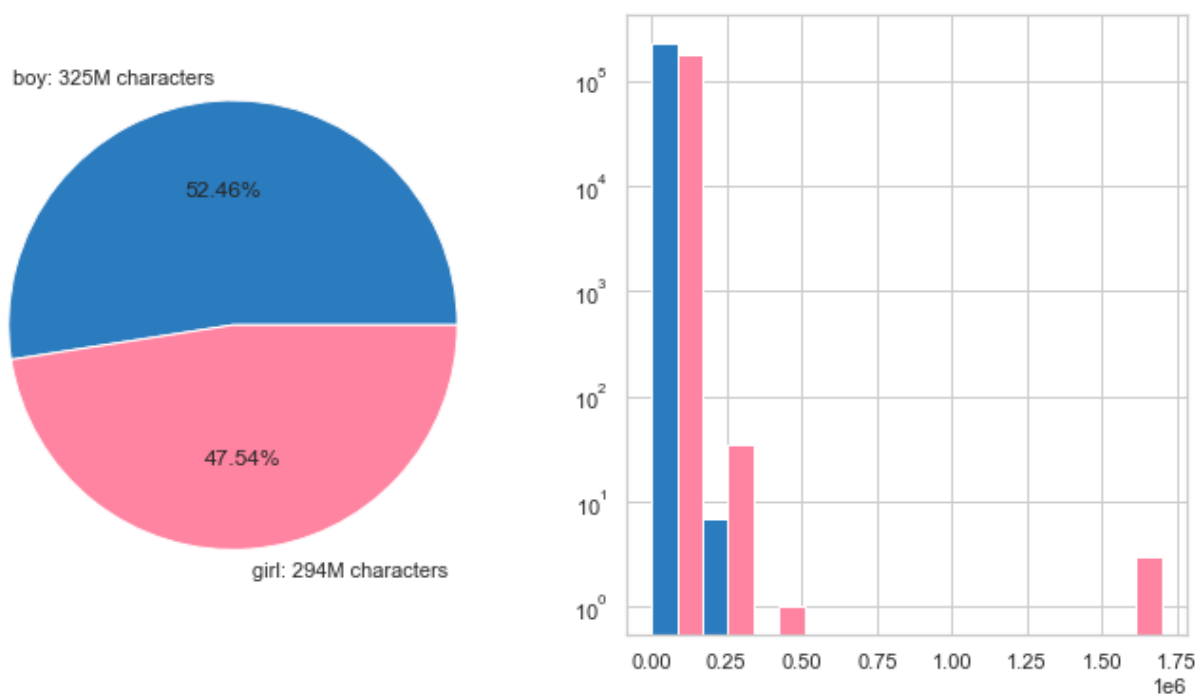
The first 15 lines are enclosed in a green rectangular box, and the word "headers" is written in green text to the right of this box. The last three lines (17-19) are enclosed in a red rectangular box, and the word "body" is written in red text to the right of this box.

Data Wrangling

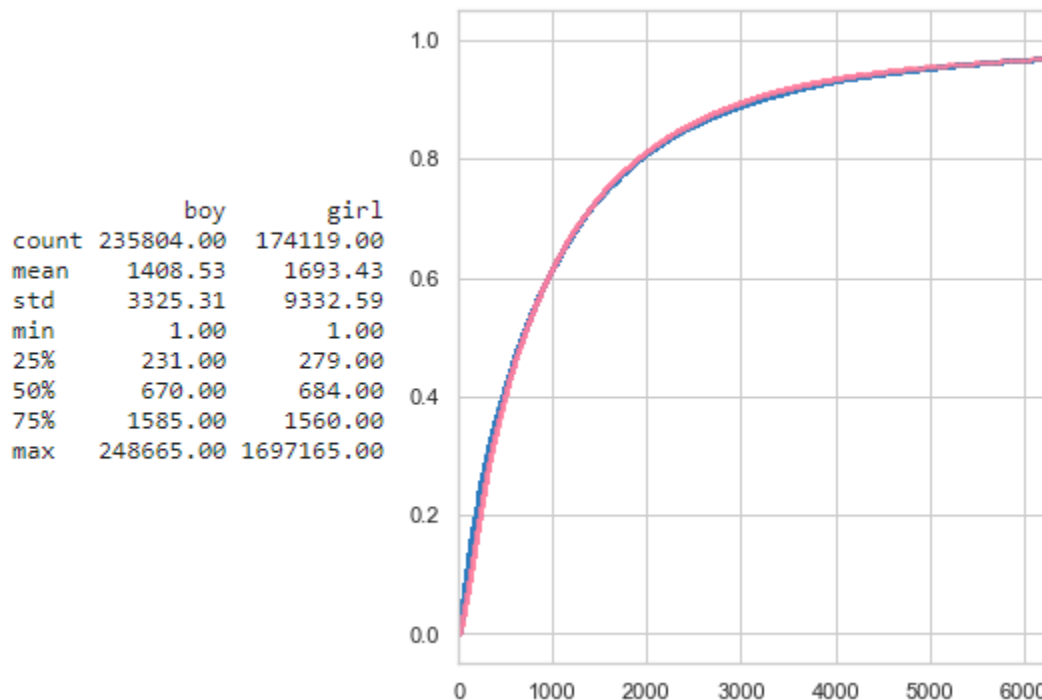
Solving the Problem. Two subsets of data are needed to solve our problem: Email Body to compose our *feature variables*; and, Sender Gender to compose our *label* or *target variables*. To quantify the entire dataset, Email Body Character Counts are used.

Getting Gender. Email senders are labelled by gender. Gender is assigned to names parsed from the standard *From:* and internal *X-From:* variables (Supporting code contained in the notebook *gender_collect_genders.ipynb*). After names are parsed from the sender emails using *regex*, they are matched to name-gender keys created with two sources: Automated internet search; and, the *Names Corpus* (Kantrowitz & Ross, 1994) from NLTK.org. Label assignment conflicts between the two sources were manually reviewed and selected through a combination of user judgment and web search. Gender labels are then converted to Boolean values (Boy: 0; girl: 1).

Initial Counts. The Enron Email Dataset is composed of ~517,000 email files, with an estimated 626 M (million) characters. ~235,000 are labeled **boy** senders and ~174,000 are labeled **girl** senders – for a ratio of 1.35 boy-to-girl. Boy senders return ~332 M characters and girl senders return ~294 M, for a 1.13 ratio. For the rest of the wrangling portion of the project, the character count relationship between genders will be emphasized because it better represents the frequency present in TF-IDF vectorization.



Standard deviations are significantly larger than the mean for both genders, suggesting high variance across the data. This can be seen in the histogram, demonstrating the heavy left skew.



The Empirical Distribution Function (ECDF, right) has the x-axis set to its 95% limit. An ECDF is a function that shows the relationship between the percentage of data contained (Y-axis) at each value tick (X-axis). In this case, the x-axis represents *Characters per Email*. To better understand this graph, note the intersection point of 1000, 0.6. This marker can be interpreted as saying, “There are 1000 characters or less contained in the email for 60% of the emails.” Capping the x-axis at 95% of the emails shows that about 95% of the dataset will have 6000 characters or less, per email. Both the 95% limit at ~6000 characters and 75% quartile at ~1600 characters suggest further investigation when considered with the boy and girl max character values of ~250,000 and 1.6 M (respectively). Outliers here might be spam email, empty character sets, or any of a class of causes that result in significantly high characters-per-email; and probably data that is *non-genuine*.

Data Cleaning. Data cleaning is handled in 3 stages: *By filter*; *by split*; and, *by string*. A review of similarity using *Cosine Similarity* is applied after – then data is sent to preprocessing. After preprocessing, a final data cleaning regarding the above noted *character count outliers* is applied. The *filtering stage* sets conditions, then filters out entire emails based on those conditions. The *splitting stage* identifies patterns within the email body that signal unwanted text, and splits the email body on those patterns. (*Note: Quoted-printable text codes are removed in the splitting stage, but are*

string substitutions. They are implemented here to correct email body text so the subsequent functions can be applied). The *string stage* isolates string patterns and addresses them with substitution, leaving the previous and following text in place.

The specific cleaning functions applied are listed below by category, along with the notebook the code is contained in:

- Clean by Filter (*clean_clean_by_filter.ipynb*):
 - Drop duplicates from *features* (Email body);
 - Drop NaN values from *targets* (Gender);
 - Remove sender email addresses without the “@enron” domain name;
 - Remove copy/pasted articles by targeting the ‘Copyright’ string;
 - Remove emails automatically generated by output reports;
- Clean by Split (*clean_clean_by_splits.ipynb*):
 - Split on five consecutive *m-dash* characters (‘-’);
 - Split on various patterns observed preceding forwarded text;
 - Split on patterns observed preceding reply text;
 - Split on timestamps that precede forwarded, reply text;
- Clean by String (*clean_clean_by_strings.ipynb*):
 - Substitute URLs with a single space character;
 - Substitute email addresses with a single space character;
 - Replace a list of words followed by a colon (‘:’) observed in duplicate text strings;

Cosine Similarity. Cosine similarity, “Computes similarity as the normalized dot product of X and Y,” writes Scikit-learn (2019). It, “Measures the cosine of the angle between two vectors projected in a multidimensional space” (Prabhakaran, 2020). The size of our dataset did not allow for the entire matrix to be calculated. Pandas *chunking* is implemented to divide the dataset into 7 subsets where a cosine similarity matrix can be returned. Scores at/above 99% are recovered from the matrix and email bodies are referenced to return ‘similar’ email body text for visual inspection.

Preprocessing. Prior to modeling, all email body text is passed to a function for preprocessing – or, processing the text data in a way that compliments the modeling function. Preprocessing steps in this project include:

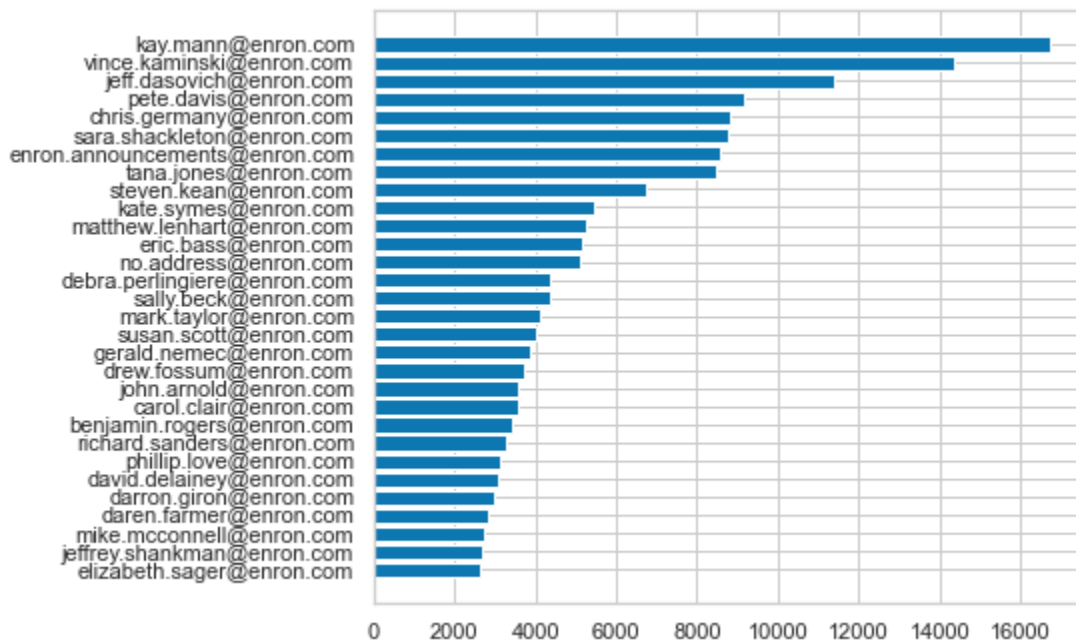
- Remove non-word characters;
- Remove underscore character;
- Remove single characters;
- Remove numbers;
- Reduce space character multiples;
- Remove stop words;
- Remove names included in the gender-name key;
- Lemmatize words;

The accessory code for preprocessing can be found in the *preprocess_run_preprocess.ipynb* notebook.

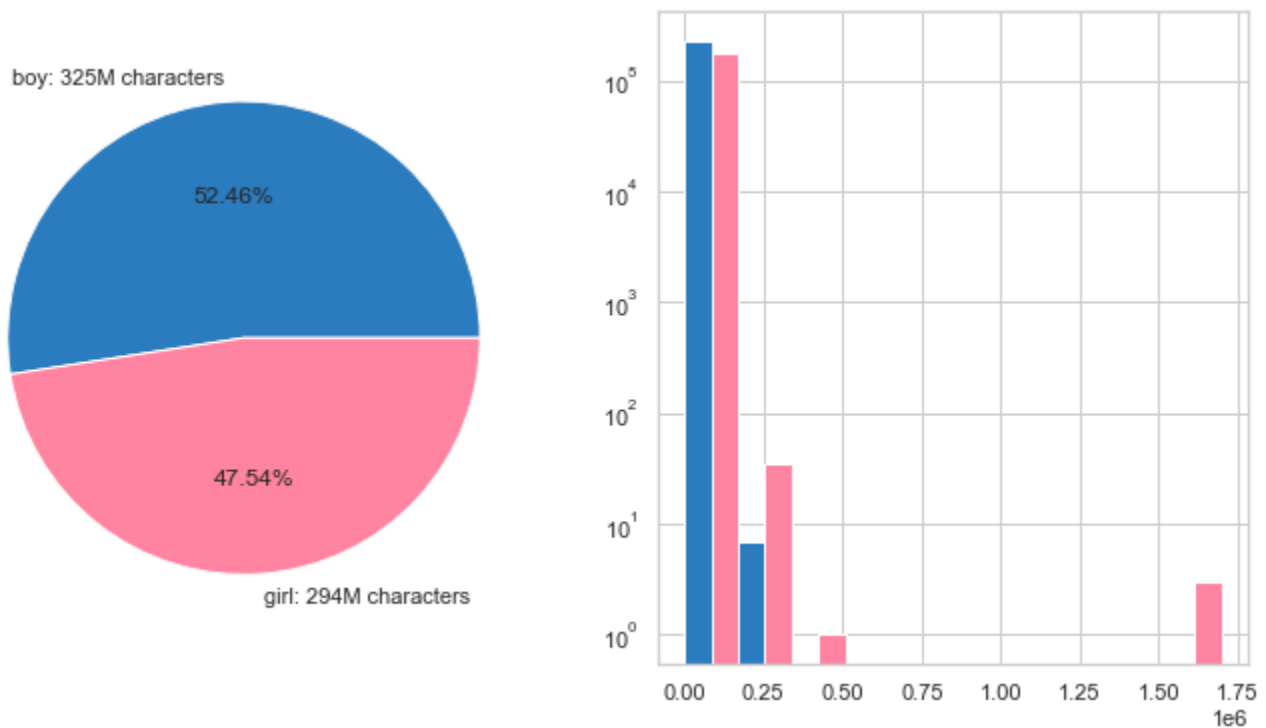
Clean Outliers. The final step in wrangling the data is removing outliers relative to character count. Email samples from outliers returned various non-genuine emails. A filter is applied returning all email bodies with character counts less than 3 sigma; or, 3 standard deviations from the mean (On the standard distribution, 3 sigma represents a probability threshold for ~99.7% dataset value inclusion).

Exploratory Data Analysis

Noted earlier, there is a significant left skew in the dataset across both labels. This skew is also present relative to *emails sent, by sender*:

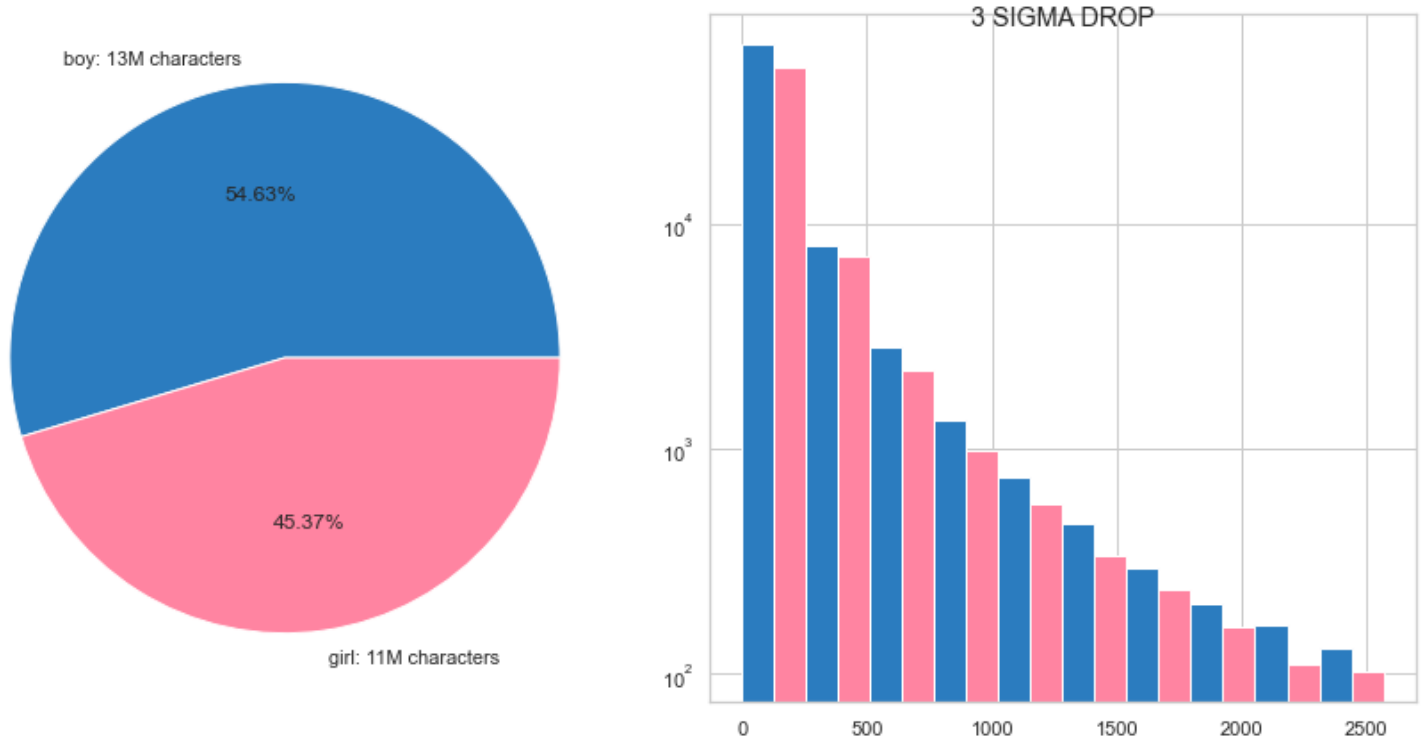


Initial Gender Distribution



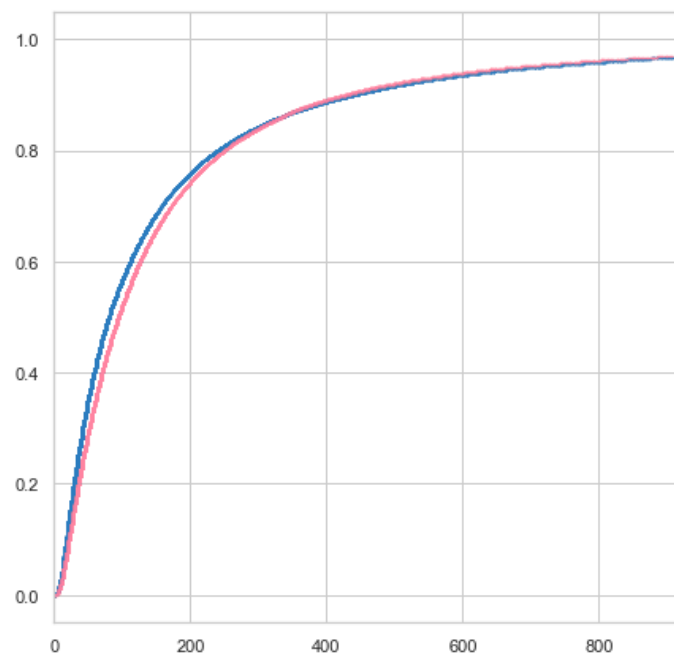
Labels applied to the initial data reflect a **1.13 boy-girl ratio** in the character counts across the corpus.

Gender Distribution After Cleaning



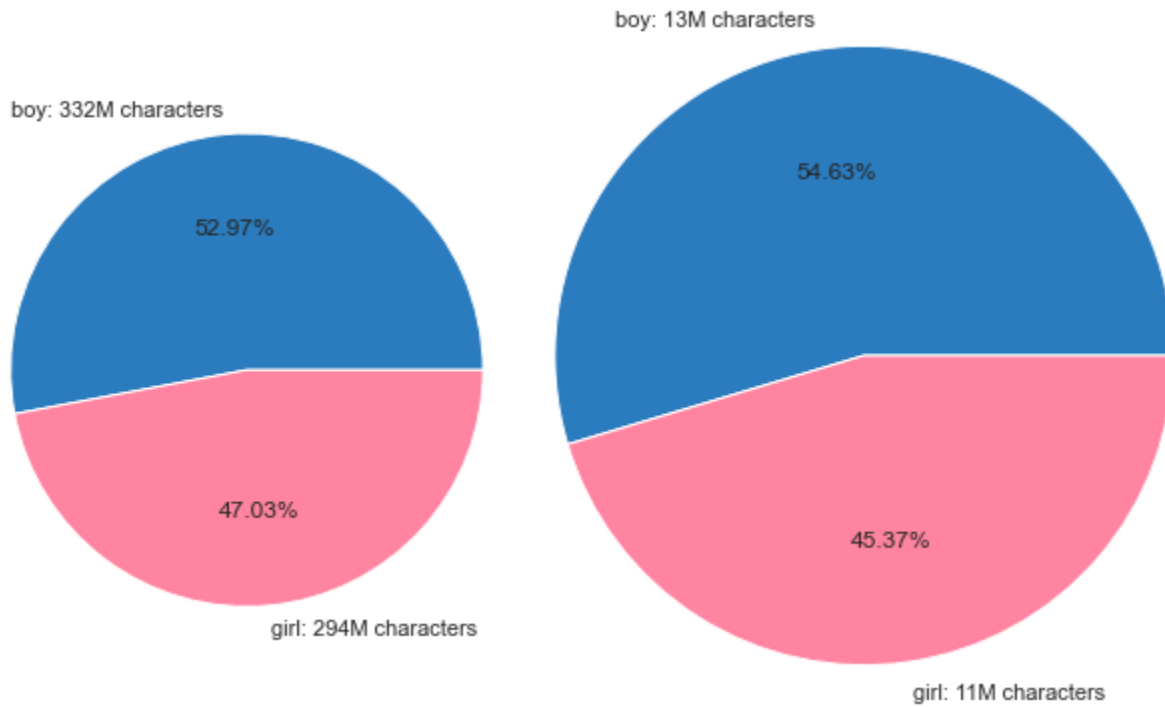
Post-Cleaning. After data has been wrangled, a **1.18 boy-girl ratio** can be observed in character counts. This ratio increases to **1.24** when considering *emails sent by gender*. A smoother histogram can be observed in the post-cleaning data as well. Though the standard deviation is still higher than the mean in both genders, their relative mean and standard deviation are more similar to one another, signaling a better balance across the feature space relative to gender.

	boy	girl
count	76597.00	61665.00
mean	180.93	186.65
std	289.02	279.92
min	1.00	1.00
25%	35.00	43.00
50%	80.00	94.00
75%	192.00	205.00
max	2565.00	2557.00



The most interesting observation is the *amount of data post-cleaning*, relative to the initial email body character values.

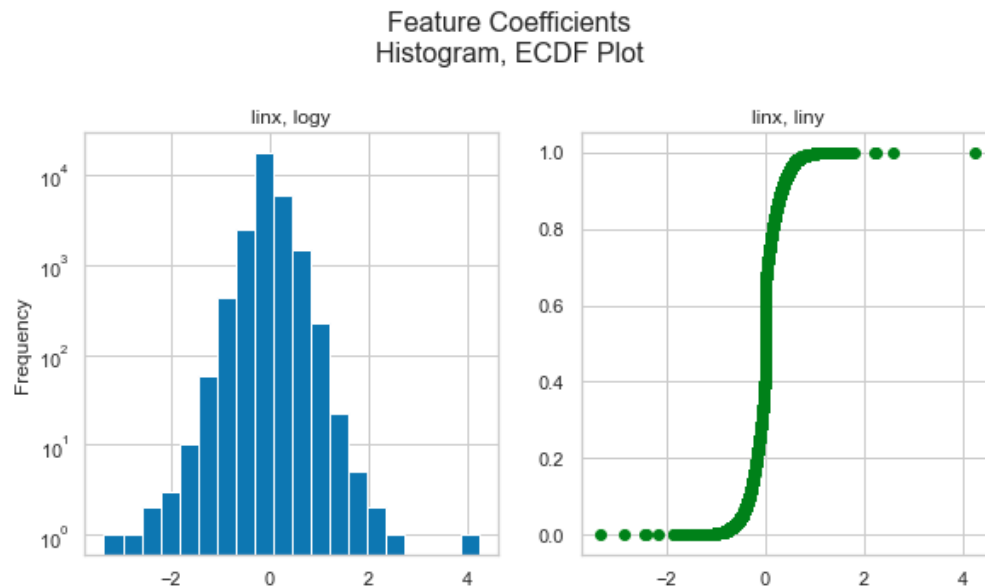
Above, the 95% x-axis limit in the ECDF is around 900 (When it was previously 6000).



Boy characters in the email body text were reduced from 332 M to 13 M, while *girl characters* were reduced from 294 M to 11 M; this equates to about 4% of the original dataset as *unique* sender text.

Feature Analysis

Threshold Parameters. Threshold parameters for *feature frequency* and *word count* can be applied using Sklearn's `TfidfVectorizer()`. To evaluate, an $n=30000$ randomized sample is passed to Sklearn's `LogisticRegression()`. Feature frequency is returned from Sklearn's `CountVectorizer()`, and character count is calculated with Pandas by applying the string function `.count()`.



Above are the histogram and ECDF plots for the *Feature Coefficients*. The distribution looks balanced across the center 0, with relatively balanced distributions in both the positive and negative tails. The top 10 features returned by coefficient magnitude only return 2-3 easily recognizable words. 9 of the 20 returns are 2 characters long; a string length that might reflect user initials or noise left over from data cleaning that might be *non-genuine* sender text.



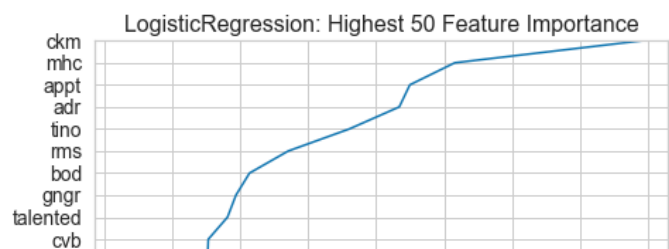
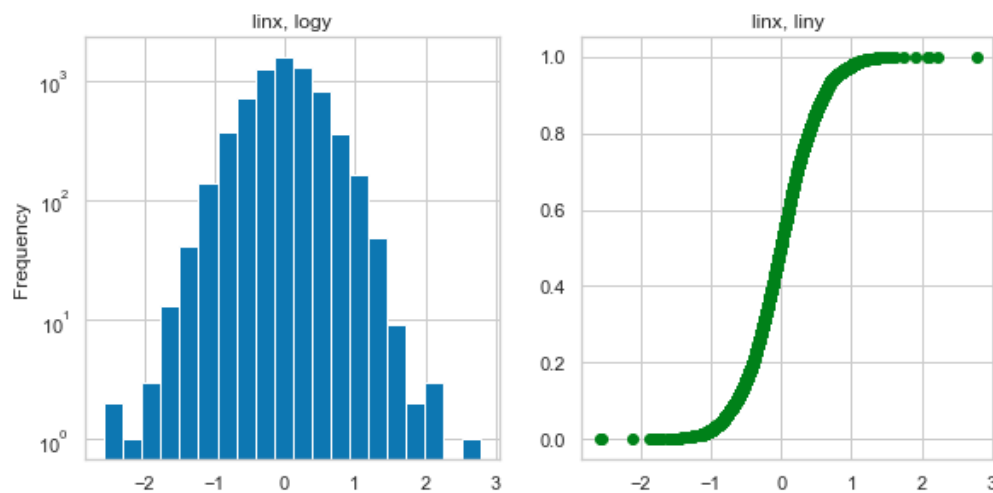
feature_name	feature_coef	feature_frequency	character_count	feature_name	feature_coef	feature_frequency	character_count
pl	-3.365496	259	2	df	4.257436	179	2
bt	-2.879711	100	2	ckm	2.606287	47	3
jmf	-2.445996	67	3	dq	2.229836	20	2
dg	-2.403187	145	2	mhc	2.219864	30	3
mat	-2.180302	68	3	appt	1.797386	24	4
cgy	-1.882465	23	3	doorstep	1.741485	39	8
executables	-1.857019	15	11	abb	1.737722	40	3
kk	-1.833449	25	2	alos	1.667401	22	4
kh	-1.806247	29	2	adr	1.629356	34	3
db	-1.762353	32	2	dp	1.566950	59	2

The first adjustment to the *word count* and *character count* thresholds is:

- Word Count: Lower limit of 7 document frequency (df; Or occurring in at least 7 documents), Upper limit of 60% (Or cannot occur in more than 60% of documents); and,
- Character Count: Lower limit of 3 (Or no less than three characters per word), Upper limit of 13 characters;

Restricting the feature frequency and word characters produce a more standardized distribution (Seen especially when comparing the two ECDF plots). While the magnitude of the coefficient positive and negative ranges is decreased, the distribution still remains balanced with the 0 point at the 50% estimator from the ECDF plot.

Feature Coefficients
Histogram, ECDF Plot



feature_name	feature_coef	feature_frequency	character_count
mat	-2.560905	71	3
jmf	-2.542075	77	3
cgas	-2.123214	59	4
executables	-1.874143	16	11
gtv	-1.825940	23	3
shout	-1.799436	35	5
cng	-1.750343	73	3
hgm	-1.724560	11	3
latter	-1.719614	35	6
aec	-1.713489	14	3

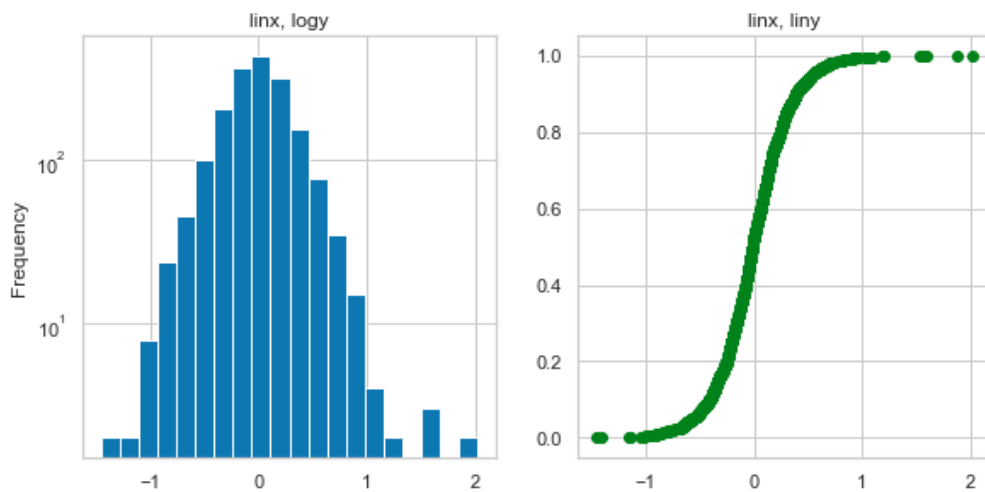
feature_name	feature_coef	feature_frequency	character_count
ckm	2.782829	54	3
mhc	2.231569	35	3
appt	2.099367	26	4
adr	2.068282	43	3
tino	1.921104	31	4
rms	1.741359	19	3
bod	1.627470	18	3
gngr	1.587551	25	4
talented	1.562603	13	8
cvb	1.506041	8	3

However, a significant number of the features are still unidentifiable; meaning they might be initials, left over from data cleaning, or some other *not-unique* text.

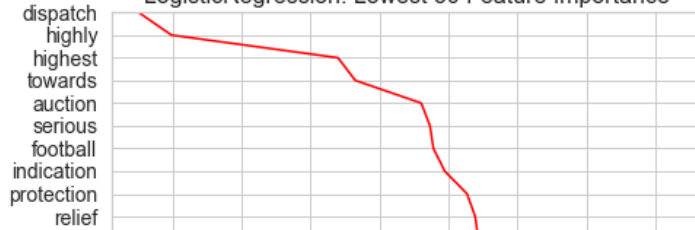
The final adjustment to the *word count* and *character count* thresholds is:

- Word Count: Lower limit at 50 df, Upper limit at 50% df; and,
- Character Count: Lower limit at 5 characters per feature word, Upper limit at 11;

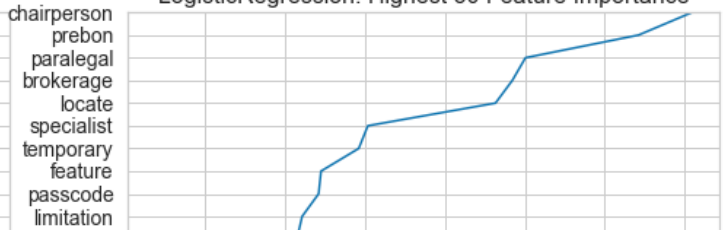
Feature Coefficients
Histogram, ECDF Plot



LogisticRegression: Lowest 50 Feature Importance



LogisticRegression: Highest 50 Feature Importance

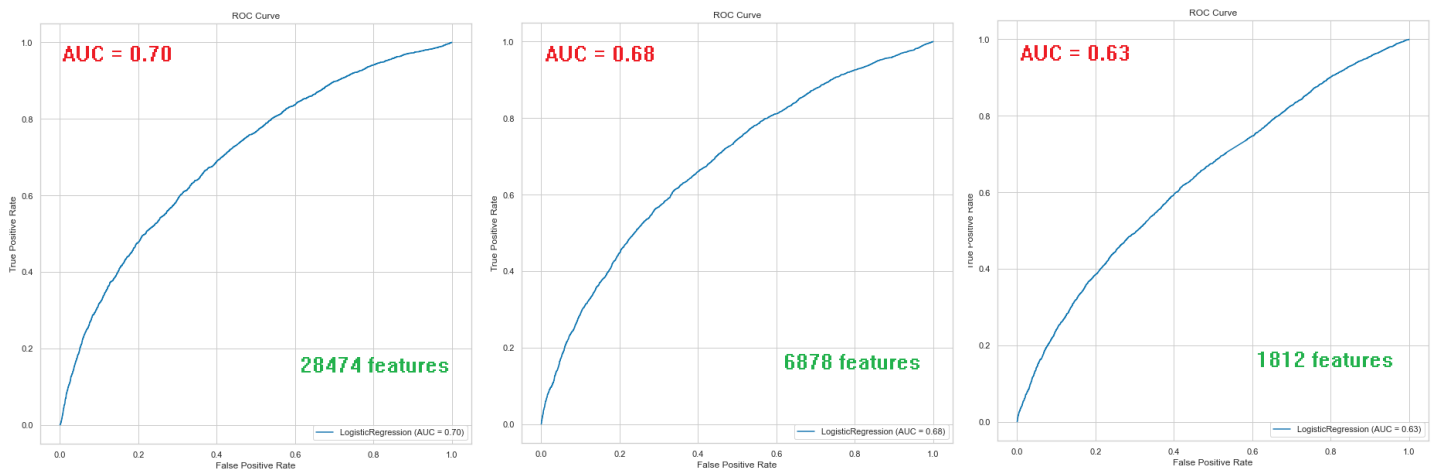


feature_name	feature_coef	feature_frequency	character_count	feature_name	feature_coef	feature_frequency	character_count
dispatch	-1.448862	68	8	chairperson	2.018544	82	11
highly	-1.401186	94	6	prebon	1.884477	115	6
highest	-1.160044	59	7	paralegal	1.601634	65	9
towards	-1.134885	83	7	brokerage	1.568712	87	9
auction	-1.039440	140	7	locate	1.526571	62	6
serious	-1.026771	68	7	specialist	1.207088	201	10
football	-1.021749	77	8	temporary	1.184306	79	9
indication	-1.005445	55	10	feature	1.089935	80	7
protection	-0.973110	63	10	passcode	1.083886	80	8
relief	-0.961283	71	6	limitation	1.042528	78	10

The final adjustment shows a further reduction in the coefficient score magnitudes, and some slight skew beginning to form. However, the features returned *are all intelligible*.

Conceptualizing Thresholds. One way to conceptualize tuning the Feature Frequency and Feature Character Count thresholds are to see the *Feature Frequency* as influencing variability across the data distribution, and to see the *Character Count* as influencing word quality. The rest of the feature analysis code, and returns, can be viewed in the notebook *features_feature_analysis.ipynb*.

While the feature word-recognition improved with more strict thresholds, the ROC Curve AUC value declined (Seen below, in order of review); however, the rate of the decline is much greater between the 2nd and 3rd frames than it is between the 1st and 2nd frames.



From these observations, the parameters input to the vectorizer for the model are:

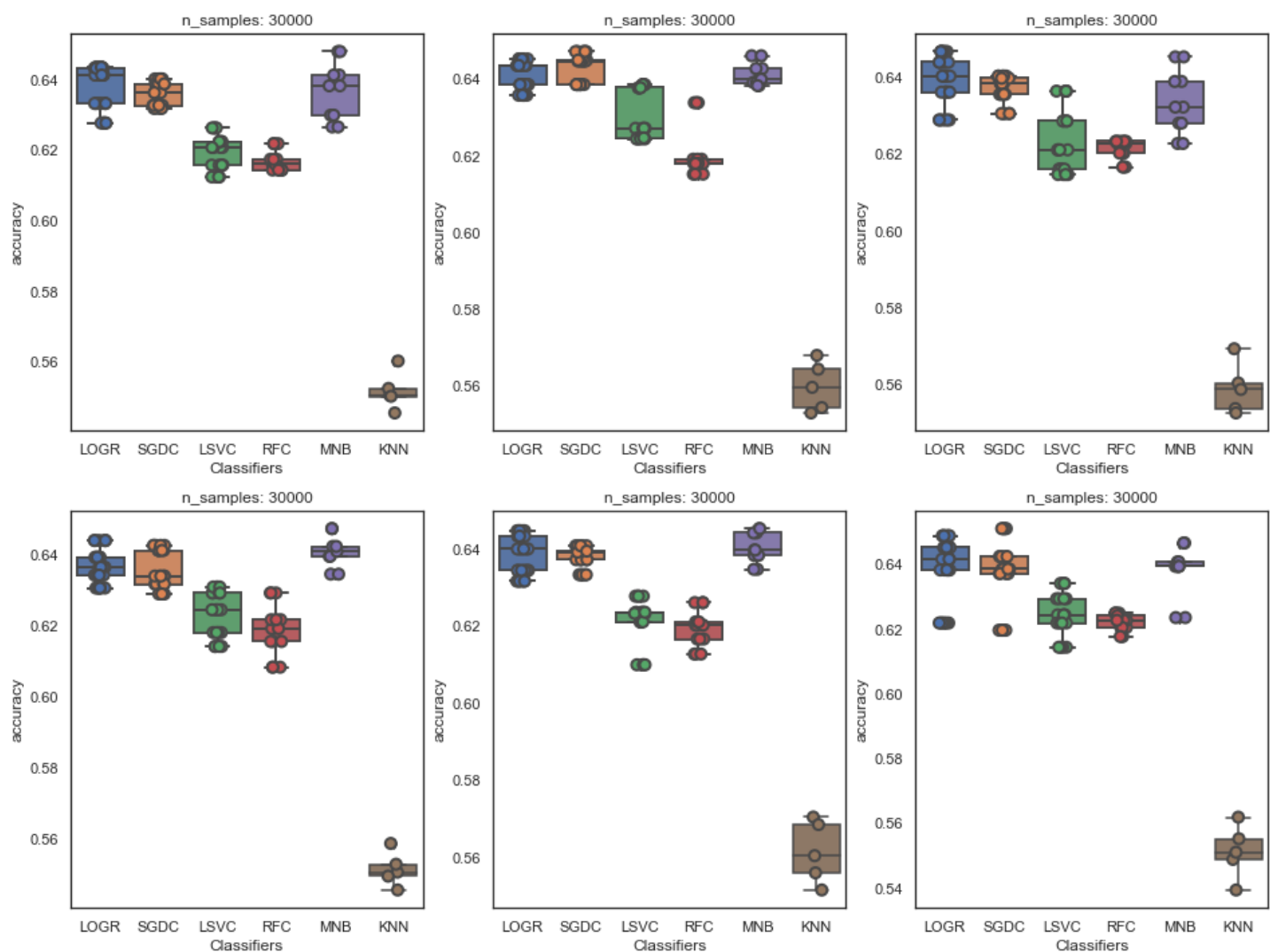
- Feature Frequency: Minimum df: 7, Maximum df: 70%; and,
- Feature Characters: Minimum characters: 4, Maximum characters: 13;

The data distribution has a heavy left skew - parameter tuning will not have a linear relationship between the upper and lower threshold adjustments (1 unit of change to minimum df *does not* translate to 1 unit of change to maximum df).

During the initial observation, the 75% quartile for Feature Frequency returned 7; meaning that 75% of the features are present in only 7 documents or less. The first adjustment used 7 as a lower threshold, contributing to a 0.02 reduction in AUC value (Combined with the other adjustments). When the next adjustment increased the document frequency threshold to 50, a reduction of 0.07 AUC was observed. A minimum character threshold of 4 is set because character counts of 2 and 3 were observed to populate features that are difficult to associate with ‘genuine’ features needed for the model to have predictive power. The upper range of 13 is set to restrict features to actual words – cutting out any long, unintelligible strings of characters.

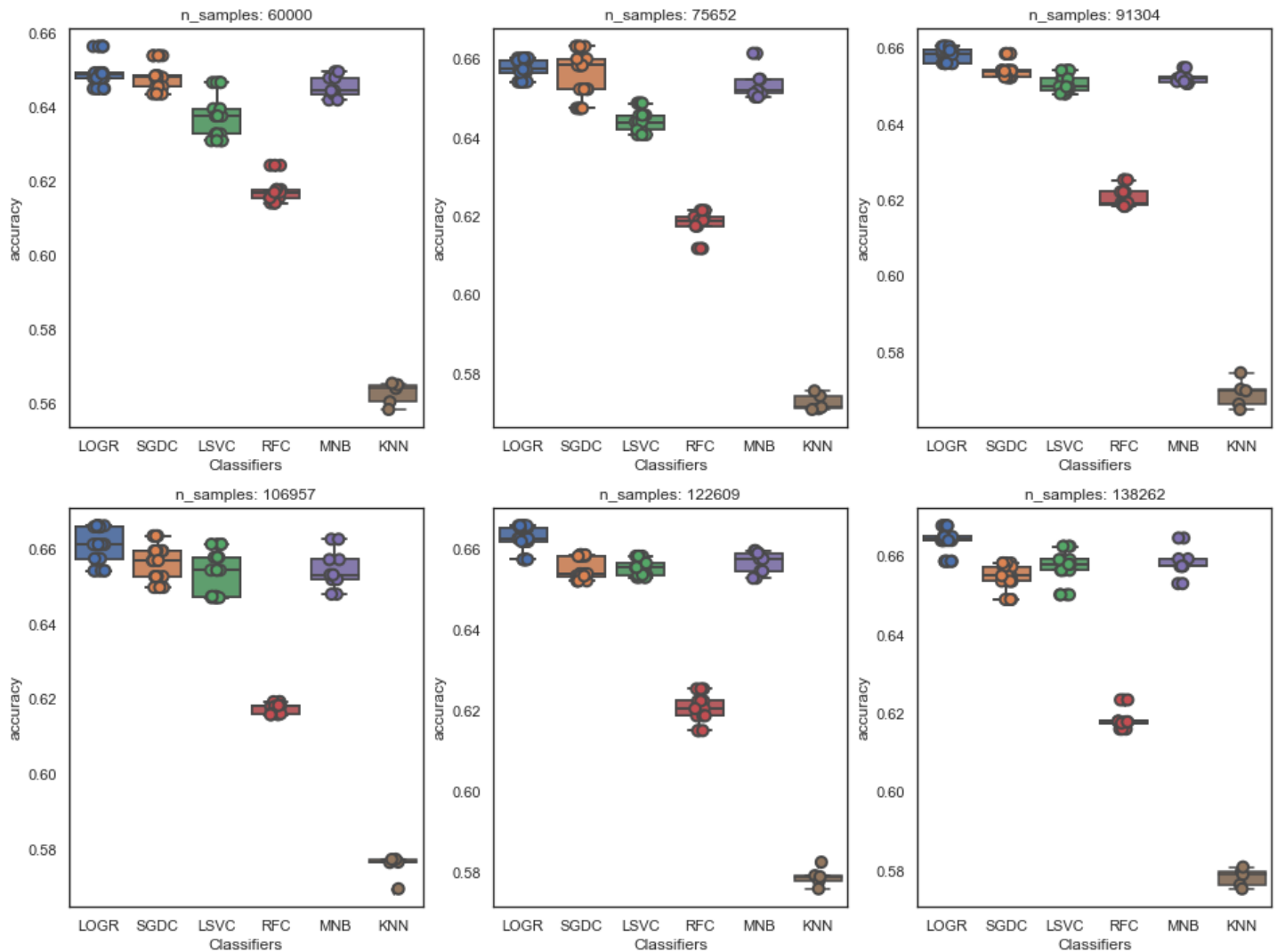
Model Implementation, Results

Evaluating Across Models. Data is initially evaluated across 6 algorithms using Sklearn: Random Forest Classifier (RFC), Multinomial Naïve Bayes (MNB), Logistic Regression (LOGR), Stochastic Gradient Descent Classifier (SGDC), Linear Support Vector Classification (LSVC), and, K Neighbors Classifier (KNN). A 5-fold cross-validation with a 60/40 train/test split, a randomized sample of $n = 30000$, and default parameters, is performed for all models. This experiment is repeated 6 times and the outputs are displayed below. Models are scored using *Accuracy* (Correct boy predictions + correct girl predictions / total number of predictions).



Varying Sample Counts. The same experiment is run with the same parameters, except the sample size is adjusted at each run across the following list of values: 60,000; 75,652; 91,304; 106,957; 122,609; and 138,262. Values are the result of splitting the linear space between 60,000 and 138,262 into 6 points including the end values. 138,262 is

the total number of emails at this stage of the project, and 60,000 is chosen to start the interval above the previous $n = 30,000$ mark.



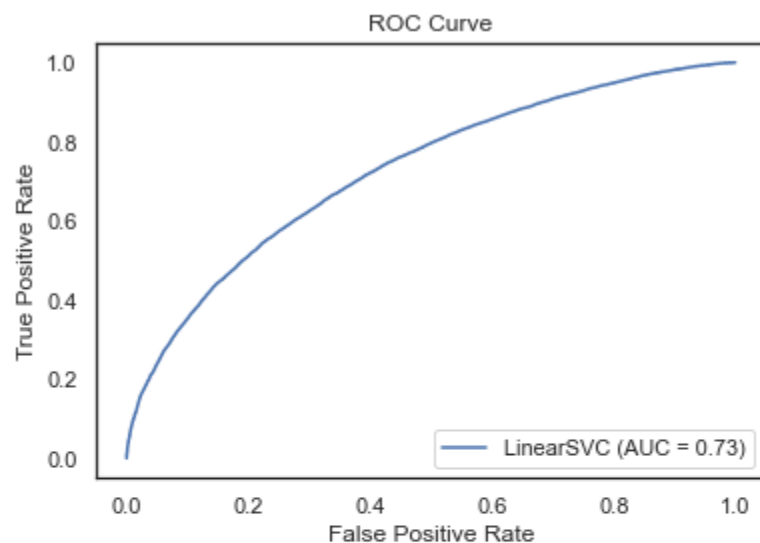
As seen above, Random Forest Classifier (RFC) and K-Neighbors Classifier (KNN) performed noticeably lower than the other four algorithms, and did not increase across varying sample intervals. When varying the sample intervals, all of the algorithms reached their max score at the $n = 75,652$ point (~55% of dataset) except the Linear Support Vector Classification (LSVC), which appears to peak at the $n = 91,304$ point (66%). Both the Logistic Regression (LOGR) and LSVC algorithms' score increases again in the higher sample intervals of $n = 122,609$ (89%) and $n = 138,262$ (100%). LOGR and LSVC are chosen for parameter evaluation.

Linear Support Vector Classification. For both parameter evaluation and best fit, data is transformed using `Sklearn TfidfVectorizer()` with a minimum document frequency of 7, maximum document frequency of 70%, and a token

character range from 4 to 13. A stop words list composed of the NLTK ‘English’ stop words combined with a list of names was removed from the dataset during preprocessing – so the ‘None’ argument is passed here to the *stop_words* parameter.

Grid search 10-Fold cross-validation with a 70/30 train/test size is used to evaluate parameters. The parameter grid contains values for *C*, the regularization parameter of the Linear Support Vector Classification algorithm, where the values are 10 points across a base 10 log space from -5 to 4 (1.e-05, 1.e-04, ..., 1.e+03, 1.e+04). The dataset is weighted with *class_weight* = ‘balanced’ (Where labels are used to automatically adjust the weights inversely proportional to label frequencies using $\text{label number of samples} / 2 * \text{total number samples}$), and max iterations are set to *max_iter* = ‘1000’. The grid search returns a value of ‘0.1’ for *C*.

Best Parameters. The Linear Support Vector Classification is fit to the entire dataset with the above parameter selections and *C* = ‘0.1’. The resulting ROC Curve, confusion matrix, and classification report are:

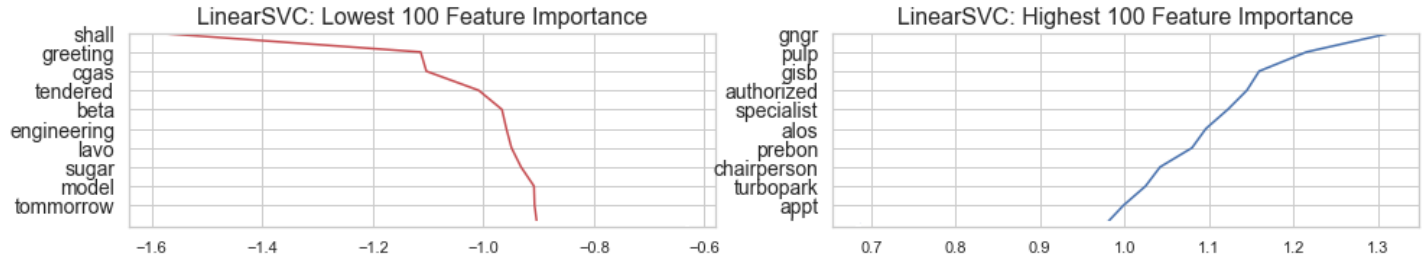


```
[[15637  7315]
 [ 6599 11928]]
      precision    recall  f1-score   support

      0.0         0.70      0.68      0.69      22952
      1.0         0.62      0.64      0.63      18527

 accuracy                   0.66      41479
 macro avg                  0.66      0.66      0.66      41479
 weighted avg               0.67      0.66      0.67      41479
```

The top 10 features by correlation coefficient (Where the lowest coefficient values represent ‘boy’ features and the highest coefficient values represent ‘girl’ features) are:

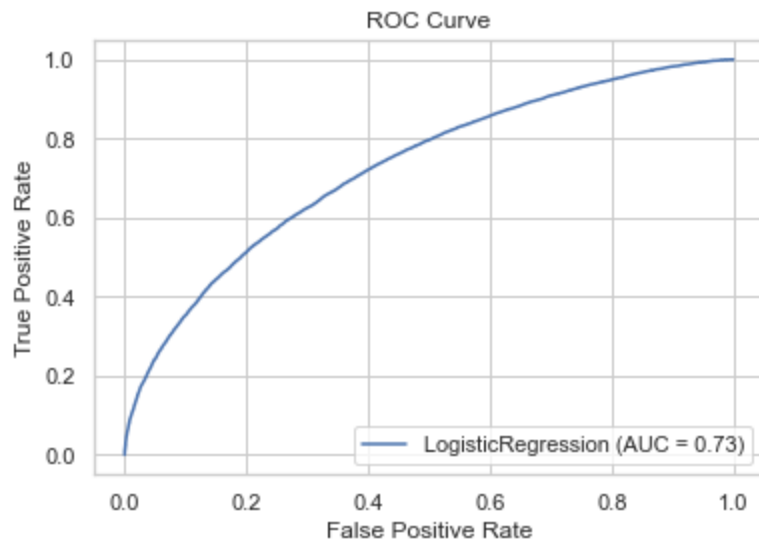


Logistic Regression. Again, for parameter estimation and best fit data is transformed using Sklearn

TfidfVectorizer() with a minimum document frequency of 7, maximum document frequency of 70%, and a token character range from 4 to 13. Stop words were removed during preprocessing and contained the NLTK stop words 'english' list combined with a list of names.

Like above, Grid search 10-Fold cross-validation with a 70/30 train/test size is used to evaluate parameters. The parameter grid contains values for C and for the solver, denoting which algorithm is used in the optimization problem. The values for C are 10 points across a base 10 log space from -5 to 4 ($1.e-05$, $1.e-04$, ..., $1.e+03$, $1.e+04$). The values for the solver are 'liblinear' and 'saga'. The dataset is weighted with $class_weight = 'balanced'$ (Where labels are used to automatically adjust the weights inversely proportional to label frequencies using $label\ number\ of\ samples / 2 * total\ number\ samples$), and max iterations are set to $max_iter = '1000'$. The grid search returns a value of '1.0' for the Logistic Regression C parameter, and 'liblinear' for the *solver*.

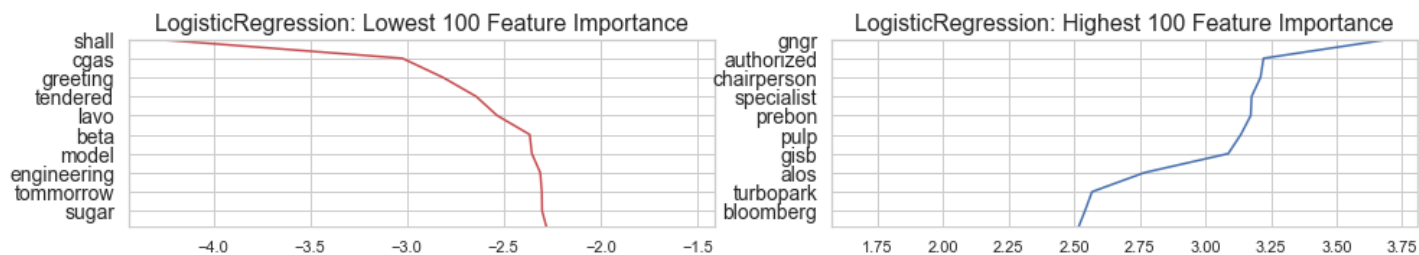
Best Parameters. The Logistic Regression model is fit to the entire dataset with the above parameter selections, $C = '1.0'$, and *solver* = 'liblinear'. The resulting ROC Curve, confusion matrix, and classification report are:



```
[[15594 7358]
 [ 6573 11954]]
```

	precision	recall	f1-score	support
0.0	0.70	0.68	0.69	22952
1.0	0.62	0.65	0.63	18527
accuracy			0.66	41479
macro avg	0.66	0.66	0.66	41479
weighted avg	0.67	0.66	0.66	41479

The top 10 features by correlation coefficient are:



The full 100 charts can be viewed in the notebook *model_model_implementation.ipynb* or at the end of this document.

Closing

Results. Both models returned similar features with respect to the labels. Three observations are made related to the returned features: Gender-related nouns; diction; and, spelling.

Gender Nouns. The first observation across both labels is the presence of gender-related features. For example, in the top 100 boy Features is *wife* at #19. In the top 100 girl Features is *husband* at #13, and *maternity* at #25.

Diction. Diction refers to the choice of words used in speech. In the top 100 boy Features are 3 expletives: *fuck*, *fucking*, and *shit*. Slang words like *dude*, *brgds* (Assumed to be some form of, ‘Best regards’), and *fundies* (A, “Pejorative slang abbreviation used to refer to religious fundamentalists”; Wikipedia 2020) are also seen in the boy Features. Across the boy features are multiple words that can be loosely related to finance, the market, or commodities: *buying*, *buyback*, *bidweek* (Related to commodity futures), *tendered*, *quote*, *model*, *intra*, *ercot* (Assumed to be an acronym for, ‘The Electric Reliability Council of Texas’), *tnrcc* (‘Texas Natural Resource Conservation Commission’), and *yield*.

The girls’ top 100 Features lacks expletives or slang words. Commodities-related acronyms are present like *gisb* (‘Gas Industry Standards Board’), and *vepco* (‘Virginia Electric and Power Company’), but they are primarily organization acronyms – unlike the boy list where different parts of speech were observed. In the girl list there are many nouns that refer to a person or position: *specialist*, *chairperson*, *attendee*, *husband*, *paralegal*, *coordinator*, *participant*, *assistant*, *employee*, *clerk*, *broker*, *suspect*, *interviewer* (Compared to *wife*, *somebody*, *dude*, and *counterpart* in the boys list).

Spelling. For spelling we will consider words that are both consciously misspelled (e.g. ‘tonite’) and unconsciously misspelled (e.g. ‘recieved’). With the boys’ list, spelling Features include: *tonite*, *till*, *cant*, *definatly*, *calender*, and *recieved* – where the first three are likely consciously misspelled, while the last three are unconsciously misspelled (i.e. The writer isn’t aware that they are misspelling the word).

The girls’ list doesn’t have as many in the top 100 with only *thnx*, *okey*, and *chnages* – the former two being conscious and the latter likely unconscious.

Assumptions. To speculate on the insight gained from the Features, it is helpful to first consider what controls word selection. Consider word selection with respect to 2 aspects: *Context*, and, *Author’s identity*.

Context is used to reference what the words are describing. A contextual sentence written about an upcoming proposal meeting might look like, “John, did you receive the updated proposal from client XYZ this morning? There are a few changes we should go over before meeting with them tomorrow.”

Second, consider the *author’s identity* – characteristics about the author that might shape their word choice or other attributes of the text. This could be their regional dialect, their education level, temperament, social environment, etc. One example is the stereotypical teen texter, “Hey r u here yet idk if u r but ill brh so txt me when u r.”

Drawing the comparison this way allows *word use* to be considered with respect to *causal factors*. For the first example, the cause would be the context. In the second example, the cause are aspects influencing the individual. In the first example, if 100 people all doing the same work project from the same vantage point were asked to write a sentence about the same activity within that work project, how much variation across the words in their sentences would be observed? Probably not too much – after all, once each writer identifies the objects of the sentence and picks from a narrow selection of verbs related to the activity, the rest of the sentence is controlled by those selections. With respect to the second example – if 100 people were asked to write *any* sentence, how much variation across the words in their sentences would be observed? Probably a lot. It can also be said that sentences *without a* strong context (That aspect from the first example that has a high influence on word selection) have a higher chance of reflecting the *author’s identity*.

Considering the features this way, a correlation based on the *context* of job roles can be seen within the gender features. For example, the boy Features relate to *buying*, *buyback*, and *bidweek*, while the girl Features reference mostly organizations. Does this indicate a majority of boys are on the trading floor, and a majority of girls handle regulatory or administrative tasks?

Reflecting on *author identity* could draw a parallel to the presence of expletives in the boy Features and their absence in the girl Features. Typically, the use of expletives in a professional atmosphere is frowned upon; there isn’t really a professional context that provokes expletives. Instead, authors’ who demonstrate traits like assertiveness, independence, or dominance can be more likely to ignore the expectation not to use expletives - and use expletives in a workplace setting anyway. This same assumption might extend to the increased number of misspelled words (Where these ‘masculine,’ traits are ignoring grammar expectations), slang words, and other insights that haven’t been observed.

Final Thoughts. (What follows is an abstraction related to drawing conclusions from features). What best describes a *girl* – someone who handles administrative tasks, or someone who rarely cusses, misspells words, or utilizes

slang? What better describes a *boy* – someone who works on the trading floor or someone who cusses in a workplace? Another distinction between *behavior* and *job title* can be drawn. *Behavior* might be abstracted to both *nature* (i.e. Genetics, natural selection, something core to the nature of *female* and *male*) and nurture. A *job title* might be abstracted to an employer (Because the employer assigns the label), an industry (Because certain job titles are only found in specific industries), or even a time period (Some job titles are relevant to the technologies of that era). What happens if a work environment is shifted to a different industry, culture, or time period? While a long-term causal influence like ‘*nature*’ could take thousands of years to evolve (And subsequently, *appear* ‘constant’ when sampled over short time periods), industry and business thrive on change in order to succeed. The job titles held primarily by women in one industry, culture, or time period might be held by men in the next industry, culture, or time period. Since job titles change frequently with respect to gender assignment, they *might* have less predictive power – our goal of this exercise.

The goal of this stream of inference is not to argue for *specific behaviors for women or men*, but rather, to question the influence of inferential assumptions on the significance of Feature returns – specifically these higher order assumptions that are significant but harder to objectify. Secondly, if higher order assumptions are factored, it must be acknowledged that bias will have a strong influence on their interpretation. Finally, how could we quantify higher order assumptions to improve model accuracy (And subsequently – should we)?

Future Improvements. This project can be improved by considering the processes that introduce bias and adjusting them to reduce the possibility; and, applying a more robust *modeling process* and *model selection process*.

This project suffers from two processes that could introduce significant bias: *The collection of gender using names parsed from email addresses* (Or the bigger problem of proper nouns limiting the predictive power of the model); and, *the magnitude of cleaning performed on the data set*. Initial modeling of this dataset prior to removing the names returned Features composed almost entirely of names. Despite removing all names parsed from email addresses and all names collected from the NLTK names list, some names still made it through to the Feature returns from the final model: *jforney*, a ‘first-initial, last-name’ pattern of name. Also, many proper nouns like *Tallahassee*, *Clarksdale*, and *Sacramento* were returned on the Feature list – proper nouns that would reduce the predictive power of the model if it were fed data that was not contextually relevant to the training dataset. Hence, a better approach to dealing with this class of Features could be developed. A process that leverages initial Feature returns as stop word inputs, different parameter settings for the word size and document frequency, incorporate other dictionaries into stop words, or classify grammar

rules to identify ‘proper nouns’ for removal – though the latter suggestions feel computationally expensive and not efficient. If a model has to model classification (Catch proper nouns), *in order to model classification* (Model the data inputs), there’s probably a better way to solve that problem (Do everything to avoid the loop within the loop).

After cleaning and preprocessing, the dataset was reduced by ~95% (If quantified by total character counts). Also, there are some characteristics that manifest in grammar that might help with the classification – punctuation, single characters, capitalization, etc. More work can be done exploring the impact of the cleaning and preprocessing performed, including evaluations on the efficacy of their implementation. For example: Consider a cleaning process where the user spends hours applying a filter to the data, running data through cosine similarity to return similar documents, and making an observation; designing another filter based on that observation, applying the new filter to the data, etc. At some point there is a plateau in the efficacy of users’ efforts to clean the data. However, the iterative observation-adjustment process used to clean and preprocess this data felt more like a vacuum where each subsequent adjustment removed the previous cleaning concern to highlight another. As a result, there isn’t a higher-order governing principle that provides insight to what was accomplished during the cleaning process past the same observation-adjustment activity.

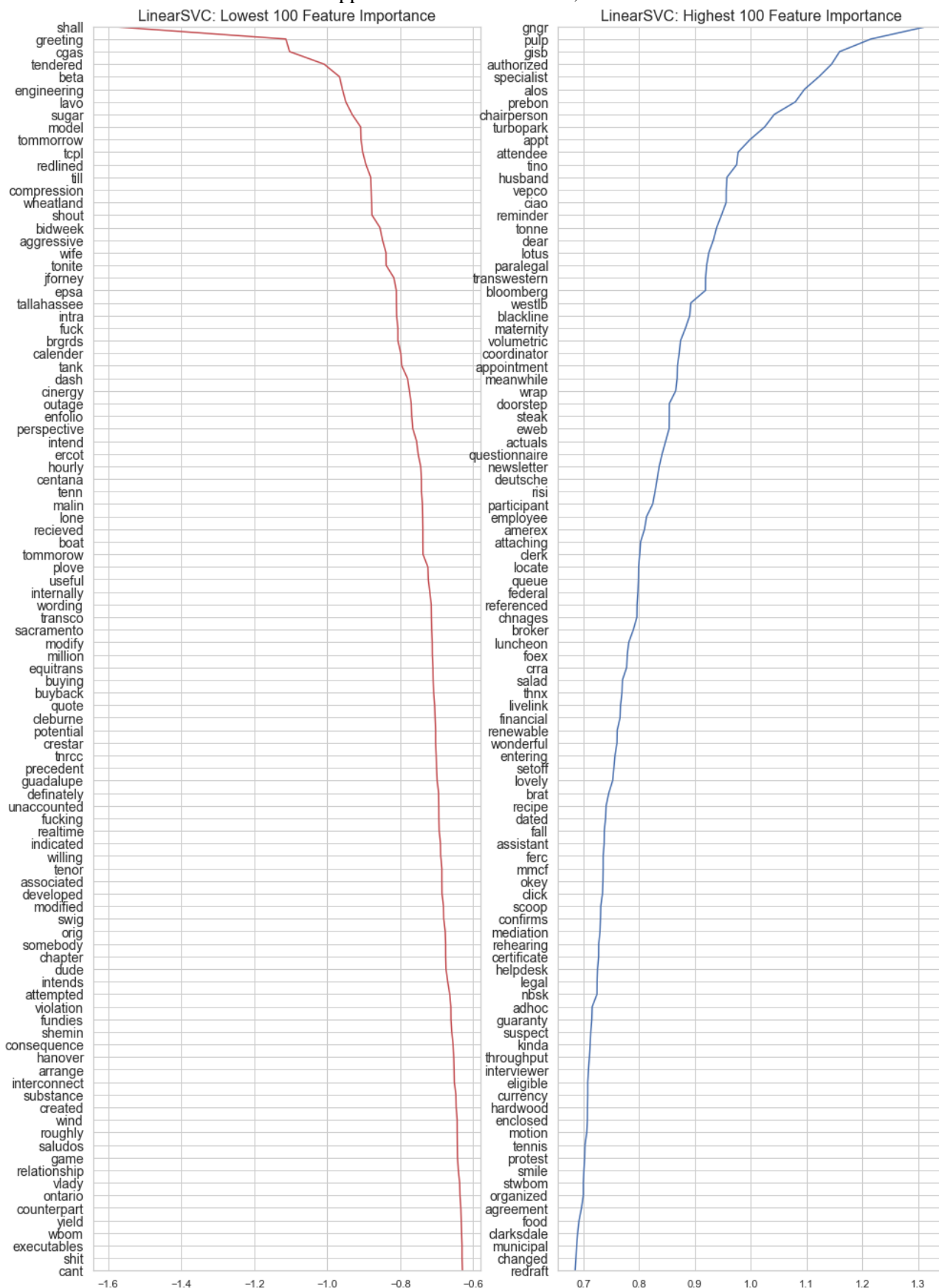
Some novice mistakes listed here can be addressed to improve this project: When the best parameters are returned to the model for the final fit, a randomized n-fold cross-validation *was not applied to the fit* (But should have to increase the quality of best fit); The initial classifier evaluation considered 6 models – however, this stage has redundancies. The Stochastic Gradient Descent Classifier (SGDC) is *not a linear model in-and-of-itself*; instead, it is an estimator that, “Implements regularized linear models with stochastic gradient descent (SGD) learning...,” Scikit-learn (2019a), that defaults to fitting a Linear Support Vector Machine (LSVM). The Linear Support Vector Classification (LSVC) model is the same algorithm as the LSVM but with the linear kernel *set as default* so the LSVC can perform much faster and scale with large datasets better. Therefore, evaluating both LSVC and LSVM simply just creates more computation time since the LSVC will perform better with the full dataset by proxy of design.

References

- Cohen, W. W. (2015). Enron Email Dataset. Retrieved from <https://www.cs.cmu.edu/~./enron/>.
- Kantrowitz, M., Ross, B. (1994). *Names Corpus, Version 1.3*. Retrieved from <http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/nlp/corpora/names/>
- Prabhakaran, S. (2020). *Cosine Similarity – Understanding the math and how it works (with python codes)*. Retrieved from <https://www.machinelearningplus.com/nlp/cosine-similarity/>.
- Scikit-learn. (2019a). *Sklearn.linear_model.SGDClassifier*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
- Scikit-learn. (2019b). *Sklearn.metrics.pairwise.cosine_similarity*. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html.
- Wikipedia. (2020). *Fundie*. Retrieved from <https://en.wikipedia.org/wiki/Fundie>

Appendix

Linear Support Vector Classification, 100 Features



Logistic Regression, Top 100 Features

