COMS 4995 - Applied Machine Learning

Fake Job Post Detection (Team 38)

Ayush Baral (ab5247), Chaewon Park (cp3227), Erin Josephine Donnelly (ejd2170)

Mukesh Bangalore Renuka (mb4862), Smarth Gupta (sg3868)

Abstract

Cyber criminals use fake job postings to obtain personally identifiable information (PPI) from unknowing applicants. As students pursuing internships and employment in a job market undergoing circumstantial shifts, we are motivated to identify and avoid dangerous job postings that pose a threat to ourselves, our peers, and many others in their job search.

Our core task is predicting the fraudulence of job postings using binary classification models based on job postings' text features and meta-features extracted from contextual embedding models, and formulating actionable insights as to which description features are most indicative of fraud.

Dataset Description

We used the *Employment Scam Aegean Dataset* from the University of Aegean's Laboratory of Information & Communication Systems Security. Data was gathered and analyzed by University of Aegean, George Mason University, and Carnegie Mellon University. Dataset contains 17,880 job postings- 17,014 legitimate, 866 fraudulent- published between 2012 and 2014. Variables include 4 strings, 4 HTML fragments, 5 binary, and 5 nominal features and meta-features.

Exploratory Data Analysis & Data Preprocessing

I. EDA

Missing value analysis: We see that 11 out of the 18 features have occasional missing values with two columns having a majority of their values missing. We also see that dropping missing values might further aggravate the imbalance of the dataset and hence we choose to estimate, predict or encode missing values. We also note that the absence of certain features might itself be a strong predictor of a posting being fraudulent and hence we encode this separately as a boolean variable.

II. Data preprocessing

We dropped ['salary_range', 'department', 'job_id'], as the first two have over 80% and 60% of its values missing, and because ['job_id'] is simply an ordinal id string that conveys no meaning. Boolean columns that indicate whether or not the columns [['description', 'company_profile', 'requirements', 'benefits'] exist were added in asd well.

To correctly encode column information, we checked the data types and segregated them into four groups:

- target_encoded_columns = ['industry', 'function', 'location']
- one_hot_encoded_columns = ["employment_type", "required_experience", "required_education"]
- text_columns = ['description', 'company_profile', 'requirements', 'benefits']
- text_len_columns = ['has_description', 'has_company_profile', 'has_requirements', 'has_benefits']

We applied target encoding on ['industry', 'function', 'location'], because one-hot encoding increased feature number to 4000, which may only overwhelm the model with no or very marginal improvement. Additionally, target encoding allows us to weigh or order the values in these features by some natural metric. After applying our proposed encoding method, we ended up with a total of 36 features.

Furthermore, we trained and evaluated the model using four different methods to vectorize the text features and compared them in the result section.

1. Non-text features only
2. Non-text features + CountVectorizer
3. Non-text features + TF-IDF Vectorizer
4. Non-text features + BERT embeddings

For Method 2 and 3, we employed the more traditional method of generating word embeddings by aggregating all text columns into one and then lemmatizing it using WordNetLemmatizer and applying TfidfVectorizer using English stop words, min_df=100, and max_features=128 parameters.

For Method 4, we used Bidirectional Encoder Representations from Transformers (BERT) to extract embeddings of size (17880,128). The small BERT model has 2 attention heads, hidden size of 128 and 2 transformer blocks.

Another caveat is that the dataset has a highly imbalanced ratio of 1:20 between the fraudulent and legitimate examples, so we used stratified splitting and applied four methods to mitigate imbalance: random up-sampling, random down-sampling, Synthetic Minority Oversampling Technique (SMOTE), and balanced class weights.

Evaluation Metrics and Model Results

In our problem, accuracy is not an appropriate metric because our data has a heavily biased label distribution. The model should be conservative when predicting 0 (legitimate) as the cost of false negatives is much greater than the cost of false positives in terms of real-world consequences. Hence, recall is our most imperative metric. We also plotted the Receiver Operating Characteristic (ROC) curve and Precision-Recall (PR) curve.

To create the best classifier, we implemented four different binary classifiers and assessed their performance. Our models include:

- Logistic Regression
- SVMs
- Decision Trees and Random Forests
- K-Nearest Neighbors

Overall, we checked the correlation matrix between the different features and dropped highly correlated features (collinearity>=0.9) while just leaving one, so that the model trains correctly with the multicollinearity problem. For KNN, the n_neighbors hyper-parameter was set as 3.

For SVMs, truncated SVD was used to simulate selecting the most principal components of the features, which was too sparse for PCA with some encodings. Letting the number of components to 100 gave the best results through experimentation.

For decision trees and random forests, hyperparameter tuning was also applied to find the best maximum tree depth, number of nodes, splitting criteria, number of trees in the forest, etc. These changes caused no change in the validation recall, however, and seemed to encourage overfitting to the majority class of the training data. It was therefore determined that the default parameters gave the most attention to the minority fraudulent examples.

In general, the default models without any imbalance mitigation showed a test recall of 0.2 ~ 0.6. Balanced class weights method only showed improvement in SVM models. SMOTE and Random Over Sampling showed a similar rate of notable improvement.

Interestingly, the best model with the highest test recall was the KNN + SMOTE model from Method 3 (non-text features + TF-IDF Vectorizer) with a test recall value of 0.9075. We consider this to be our best model despite the false positive occurrences because we prioritize correct identification of the minority class. Figure 1 shows the confusion matrix for the test set for this model.

Additionally, many models achieved over 85% recall of fraudulent examples on the test set with varying supplementary scores including 100% accuracy and recall on the development data with some models. Some of these other models that achieved relatively high recall on the test set had significantly fewer false positive occurrences. One such example is a model with SVM + Random Over Sampling from Method 3, whose confusion matrix on the test set can be found in the Appendix as Figure 2.

The results and metrics of various models with the aforementioned preprocessing and sampling methods can be found in the Appendix following this report.

Insights and Conclusion

I. Features and Relevant Information

Our process and results allow us to answer questions we set out to answer:

How indicative of legitimacy or fraudulence are the textual elements of a job posting?
The textual elements contained sufficient information to predict the legitimacy or fraudulence of job postings with up to 90% recall of fraudulent examples.

Which features of a job posting are most indicative of a fraudulent posting?
The text of the job postings with its various encodings proved to be the most indicative of the posting's legitimacy or fraudulence. These were more indicative than numeric features such as number of missing values in each posting, whether benefits was a missing value, or the target encoded location, for presence of these values between the two classes was not different enough. Finally, some engineered features such as length of various textual elements, either raw or log-scaled, even hurt model performance, and were excluded altogether in favor of letting the models focus on the textual embeddings.

II. Applications and Further Action

To make our insights actionable, we can ask and answer the questions below:

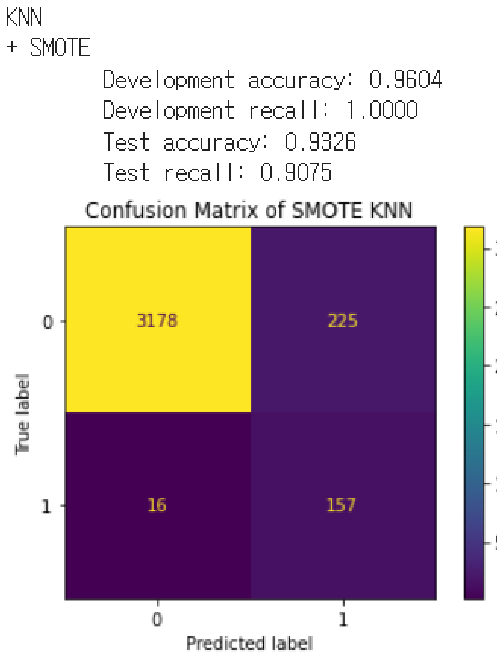
How can we use our insights to inform and protect potential applicants?
Our results show the potential machine learning has to protect users of various job posting platforms against fraudulent or malicious job postings. On an individual level, we can glean features from job postings encountered by us, our peers, or academic advisors in our own job searches and feed them through the models as test examples. On a larger scale, our models can be implemented as a security measure by various job posting platforms to warn users of potential threats or prevent the postings from being available to users altogether.

How can our predictions be used to implement a screening process on popular job posting sites to protect applicants on a larger scale?
Job posting platforms such as LinkedIn, Glassdoor, and Indeed as well as other services that search these sites to suggest job postings to potential applicants can implement similar models to display warning or hide postings that are deemed likely to be fraudulent. Online career services have developed significantly since 2012 and 2014 when our data was obtained, and likely with it the frequency and trickery of malicious job postings. Not only does this demonstrate the need to implement security measures to protect potential applicants, but it also encourages customizations of these models based on the specific data available through each of these services.

How can these results encourage further developments in fraudulence detection?
Judging by the success of our models on job posting data and the potential benefit if applied to job services, we anticipate that models trained on data from different fields would show the same potential for good. Such applications include classifying legitimate and fraudulent instances of housing postings, social media accounts, news articles, online dating profiles, and e-tail sites. Textual elements carry a great amount of information which can be captured through different embeddings and understood by machine learning models, giving insights to users who may be otherwise unaware when relying solely on features available from human perception.

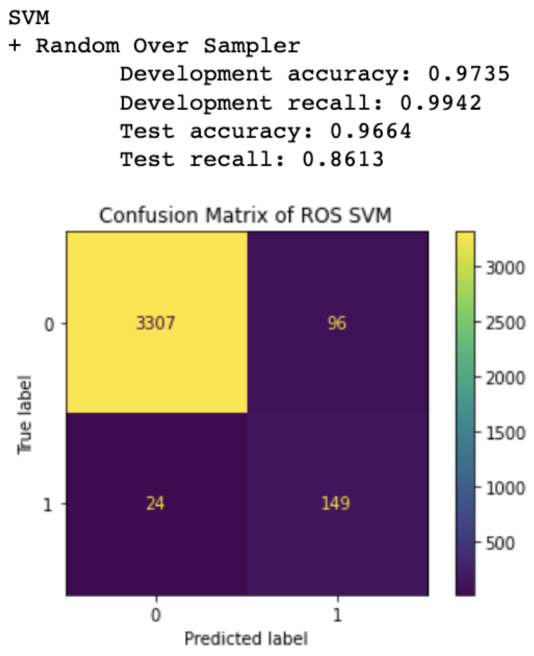
Appendix

I. Figures



```
KNN
+ SMOTE
        Development accuracy: 0.9604
        Development recall: 1.0000
        Test accuracy: 0.9326
        Test recall: 0.9075
```

Figure 1: Confusion matrix for the model with the highest recall on the testing set. Text features were encoded using TF-IDF Vectorizer.



```
SVM
+ Random Over Sampler
        Development accuracy: 0.9735
        Development recall: 0.9942
        Test accuracy: 0.9664
        Test recall: 0.8613
```

Figure 2: Confusion matrix for a model with relatively high recall on the testing set and relatively low false positive rate. Text features were encoded using TF-IDF Vectorizer.

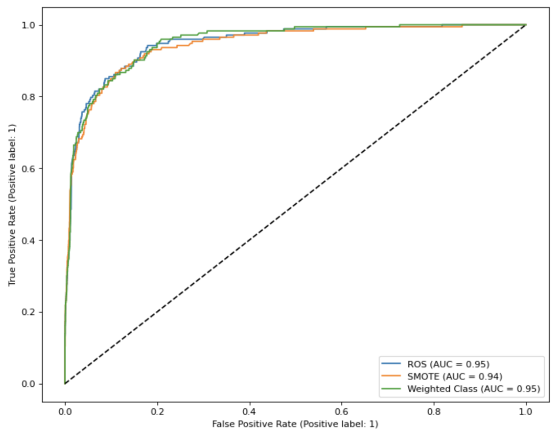
II. Model Performance Charts and Performance Curves

The best performance (as measured by recall) per model is denoted in **red**, the best across all models is highlighted in **yellow**

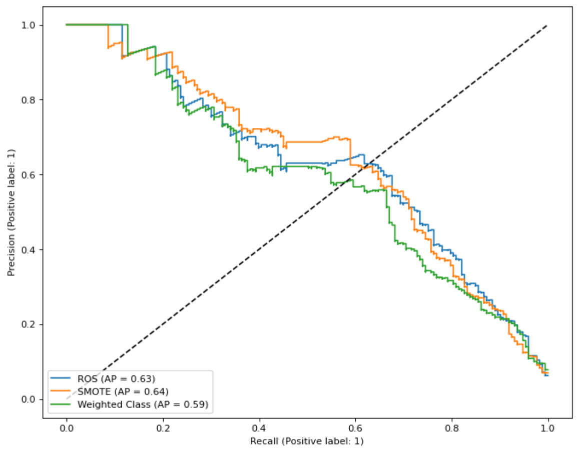
Method 1 Model Performance Chart—non-text features only

	Default	**Random Over Sample**	**SMOTE**	**Balanced Class Weights**
Logistic Regression	**Dev Acc: 0.9676** **Dev Recall: 0.4228** **Test Acc: 0.9636** **Test Recall: 0.3006**	**Dev Acc: 0.8868** **Dev Recall: 0.8817** **Test Acc: 0.8798** **Test Recall: 0.8150**	**Dev Acc: 0.8918** **Dev Recall: 0.8874** **Test Acc: 0.8826** **Test Recall: 0.8035**	**Dev Acc: 0.9676** **Dev Recall: 0.4228** **Test Acc: 0.9636** **Test Recall: 0.3006**
KNN	**Dev Acc: 0.9824** **Dev Recall: 0.7648** **Test Acc: 0.9715** **Test Recall: 0.5838**	**Dev Acc: 0.9858** **Dev Recall: 0.9740** **Test Acc: 0.9606** **Test Recall: 0.7052**	**Dev Acc: 0.9804** **Dev Recall: 0.9625** **Test Acc: 0.9547** **Test Recall: 0.7168**	**Dev Acc: 0.9824** **Dev Recall: 0.7648** **Test Acc: 0.9715** **Test Recall: 0.5838**
Decision Tree	**Dev Acc: 0.9966** **Dev Recall: 0.9683** **Test Acc: 0.9662** **Test Recall: 0.6474**	**Dev Acc: 0.9901** **Dev Recall: 0.9986** **Test Acc: 0.9614** **Test Recall: 0.6763**	**Dev Acc: 0.9958** **Dev Recall: 0.9784** **Test Acc: 0.9622** **Test Recall: 0.6994**	**Dev Acc: 0.9901** **Dev Recall: 0.9986** **Test Acc: 0.9622** **Test Recall: 0.6647**
Random Forest	**Dev Acc: 0.9966** **Dev Recall: 0.9697** **Test Acc: 0.9790** **Test Recall: 0.6590**	**Dev Acc: 0.9901** **Dev Recall: 0.9986** **Test Acc: 0.9681** **Test Recall: 0.6532**	**Dev Acc: 0.9958** **Dev Recall: 0.9827** **Test Acc: 0.9681** **Test Recall: 0.6994**	**Dev Acc: 0.9926** **Dev Recall: 0.9942** **Test Acc: 0.9729** **Test Recall: 0.6532**
SVM	**Dev Acc: 0.9675** **Dev Recall: 0.3781** **Test Acc: 0.9611** **Test Recall: 0.2428**	**Dev Acc: 0.9090** **Dev Recall: 0.9264** **Test Acc: 0.9004** **Test Recall: 0.8497**	**Dev Acc: 0.9189** **Dev Recall: 0.9048** **Test Acc: 0.9100** **Test Recall: 0.8092**	**Dev Acc: 0.9009** **Dev Recall: 0.9134** **Test Acc: 0.8937** **Test Recall: 0.8266**

Method 1 Model AUC-ROC Curves



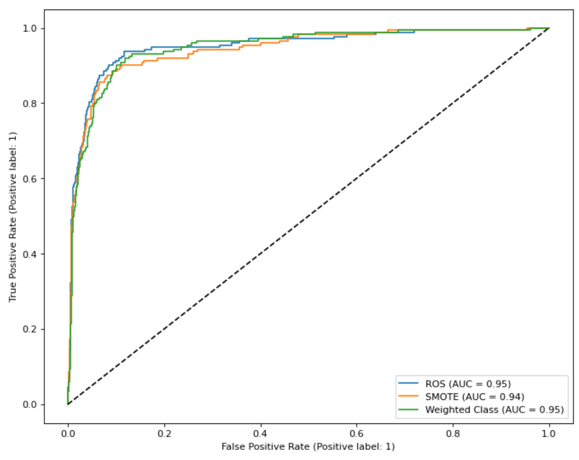
Method 1 Model Precision-Recall Curves



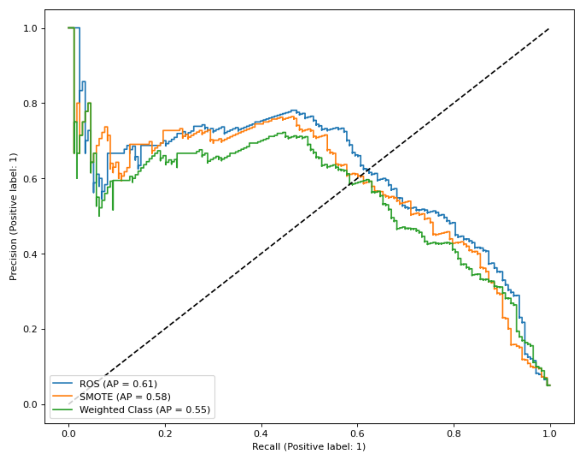
Method 2 Model Performance Chart—non-text features + CountVectorizer

	Default	Random Over Sample	SMOTE	Balanced Class Weights
Logistic Regression	**Dev Acc: 0.9759** **Dev Recall: 0.5974** **Test Acc: 0.9726** **Test Recall: 0.5318**	**Dev Acc: 0.9151** **Dev Recall: 0.9293** **Test Acc: 0.9100** **Test Recall: 0.8728**	**Dev Acc: 0.9341** **Dev Recall: 0.9048** **Test Acc: 0.9304** **Test Recall: 0.8439**	**Dev Acc: 0.9759** **Dev Recall: 0.5974** **Test Acc: 0.9726** **Test Recall: 0.5318**
KNN	**Dev Acc: 0.9833** **Dev Recall: 0.7850** **Test Acc: 0.9732** **Test Recall: 0.6590**	**Dev Acc: 0.9812** **Dev Recall: 1.0000** **Test Acc: 0.9583** **Test Recall: 0.7746**	**Dev Acc: 0.9339** **Dev Recall: 1.000** **Test Acc: 0.8853** **Test Recall: 0.8786**	**Dev Acc: 0.9833** **Dev Recall: 0.7850** **Test Acc: 0.9732** **Test Recall: 0.6590**
Decision Tree	**Dev Acc: 1.0000** **Dev Recall: 1.0000** **Test Acc: 0.9667** **Test Recall: 0.6821**	**Dev Acc: 1.0000** **Dev Recall: 1.0000** **Test Acc: 0.9642** **Test Recall: 0.5838**	**Dev Acc: 1.0000** **Dev Recall: 1.0000** **Test Acc: 0.9622** **Test Recall: 0.6416**	**Dev Acc: 1.0000** **Dev Recall: 1.0000** **Test Acc: 0.9676** **Test Recall: 0.5954**
Random Forest	**Dev Acc: 1.0000** **Dev Recall: 1.0000** **Test Acc: 0.9810** **Test Recall: 0.6127**	**Dev Acc: 1.0000** **Dev Recall: 1.0000** **Test Acc: 0.9790** **Test Recall: 0.6243**	**Dev Acc: 1.0000** **Dev Recall: 1.0000** **Test Acc: 0.9796** **Test Recall: 0.6474**	**Dev Acc: 1.0000** **Dev Recall: 1.0000** **Test Acc: 0.9762** **Test Recall: 0.5376**
SVM	**Dev Acc: 0.9671** **Dev Recall: 0.3203** **Test Acc: 0.9662** **Test Recall: 0.3006**	**Dev Acc: 0.9491** **Dev Recall: 0.9957** **Test Acc: 0.9418** **Test Recall: 0.8150**	**Dev Acc: 0.950** **Dev Recall: 0.9827** **Test Acc: 0.9427** **Test Recall: 0.7919**	**Dev Acc: 0.9218** **Dev Recall: 0.9870** **Test Acc: 0.9181** **Test Recall: 0.8382**

Method 2 Model AUC-ROC Curves



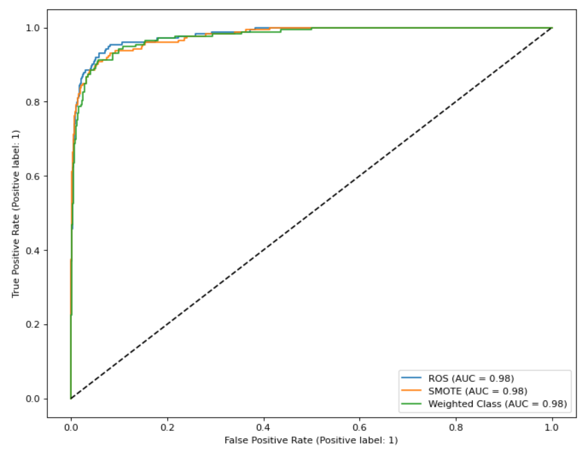
Method 2 Model Precision-Recall Curves



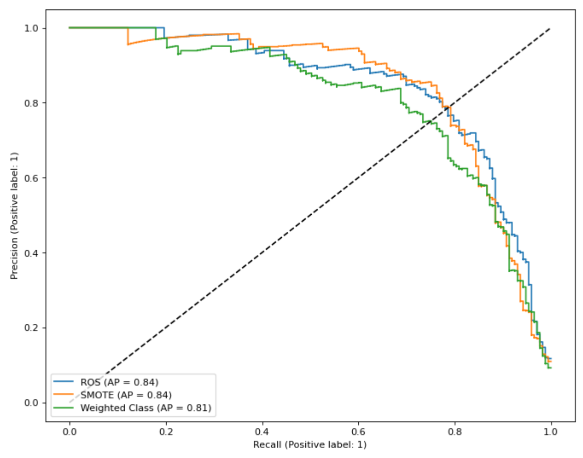
Method 3 Model Performance Chart—non-text features + TF-IDF Vectorizer

	Default	Random Over Sample	SMOTE	Balanced Class Weights
Logistic Regression	Dev Acc: 0.9741 Dev Recall: 0.5368 Test Acc: 0.9678 Test Recall: 0.4162	Dev Acc: 0.9194 Dev Recall: 0.9250 Test Acc: 0.9234 Test Recall: 0.8786	Dev Acc: 0.9283 Dev Recall: 0.9105 Test Acc: 0.9304 Test Recall: 0.8613	Dev Acc: 0.9741 Dev Recall: 0.5368 Test Acc: 0.9678 Test Recall: 0.4162
KNN	Dev Acc: 0.9873 Dev Recall: 0.8095 Test Acc: 0.9776 Test Recall: 0.6590	Dev Acc: 0.9849 Dev Recall: 1.0000 Test Acc: 0.9715 Test Recall: 0.8092	Dev Acc: 0.9604 Dev Recall: 1.0000 Test Acc: 0.9326 Test Recall: 0.9075	Dev Acc: 0.9873 Dev Recall: 0.8095 Test Acc: 0.9776 Test Recall: 0.6590
Decision Tree	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.9701 Test Recall: 0.6590	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.9603 Test Recall: 0.6069	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.9539 Test Recall: 0.6590	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.9581 Test Recall: 0.6012
Random Forest	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.9801 Test Recall: 0.5954	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.979 Test Recall: 0.6127	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.9815 Test Recall: 0.6763	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.9762 Test Recall: 0.5260
SVM	Dev Acc: 0.9773 Dev Recall: 0.5527 Test Acc: 0.9715 Test Recall: 0.4393	Dev Acc: 0.9734 Dev Recall: 0.9971 Test Acc: 0.9687 Test Recall: 0.8728	Dev Acc: 0.9796 Dev Recall: 0.9841 Test Acc: 0.9729 Test Recall: 0.8439	Dev Acc: 0.9556 Dev Recall: 0.9827 Test Acc: 0.9539 Test Recall: 0.8844

Method 3 Model AUC-ROC Curves



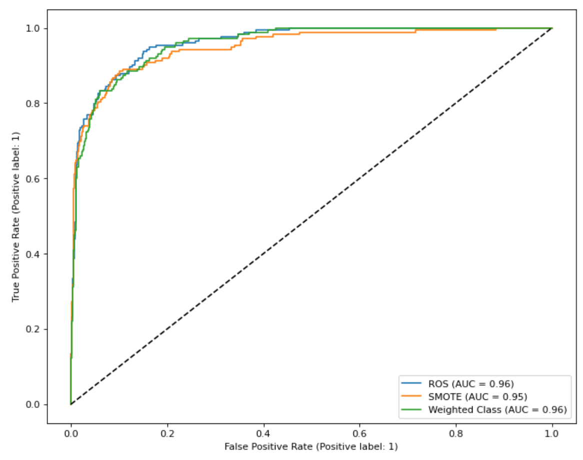
Method 3 Model Precision-Recall Curves



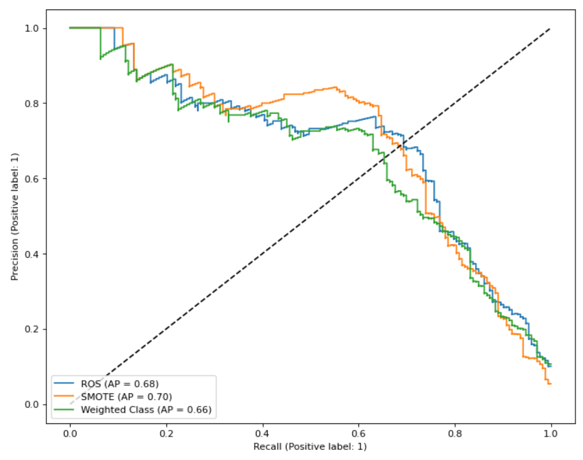
Method 4 Model Performance Chart—non-text features + BERT embeddings

	Default	Random Over Sample	SMOTE	Balanced Class Weights
Logistic Regression	Dev Acc: 0.9725 Dev Recall: 0.5180 Test Acc: 0.9670 Test Recall: 0.3757	Dev Acc: 0.9075 Dev Recall: 0.9062 Test Acc: 0.9044 Test Recall: 0.7977	Dev Acc: 0.9177 Dev Recall:0.8889 Test Acc: 0.9150 Test Recall: 0.7861	Dev Acc: 0.9725 Dev Recall: 0.5180 Test Acc: 0.9670 Test Recall: 0.3757
KNN	Dev Acc:0.9842 Dev Recall: 0.7460 Test Acc: 0.9734 Test Recall: 0.6185	Dev Acc: 0.9870 Dev Recall: 1.0000 Test Acc: 0.9628 Test Recall: 0.7630	Dev Acc: 0.9602 Dev Recall: 1.0000 Test Acc: 0.9320 Test Recall: 0.8786	Dev Acc: 0.9842 Dev Recall: 0.7460 Test Acc: 0.9734 Test Recall: 0.6185
Decision Tree	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.9673 Test Recall: 0.6821	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.9639 Test Recall: 0.6243	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.9564 Test Recall: 0.6647	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.9609 Test Recall: 0.6127
Random Forest	Dev Acc: 0.9999 Dev Recall: 0.9986 Test Acc: 0.9790 Test Recall: 0.5665	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.9796 Test Recall: 0.5838	Dev Acc: 1.0000 Dev Recall: 1.0000 Test Acc: 0.9807 Test Recall: 0.6763	Dev Acc: 0.9999 Dev Recall: 0.9986 Test Acc: 0.9762 Test Recall: 0.5087
SVM	Dev Acc: 0.9635 Dev Recall: 0.2496 Test Acc: 0.9603 Test Recall: 0.1908	Dev Acc: 0.9279 Dev Recall: 0.9437 Test Acc: 0.9200 Test Recall: 0.8439	Dev Acc: 0.9437 Dev Recall: 0.9134 Test Acc: 0.9334 Test Recall: 0.8035	Dev Acc: 0.9156 Dev Recall: 0.9322 Test Acc: 0.9108 Test Recall: 0.8382

Method 4 Model AUC-ROC Curves



Method 4 Model Precision-Recall Curves



III. Feature Correlation Plot