CSE 5717 – ATS Feedback System

Team 1: Loc Tran, Andrew Graham, Kunal Bagga

Problem Statement

The Applicant Tracking System (ATS) is a frequently used system that hiring managers used to filter applicants. The system reads the text and asserts how well the applicant fits the job description by giving a 0 if the applicant will not get an interview and a 1 if the applicant will get an interview. Our goal is to give feedback on how well a resume fits with the job description, like how well the skills align.

To achieve our goal, we use binary classification for the interview classification and multi-classification for the subjective feedback. To measure the performance of our models, we use accuracy.

Data Set and Preparation

Dataset Selection

Resume Dataset

This dataset was aggregated from 3 other datasets. These other datasets were compiled of resumes from livecareer.com, this github repository, and other resumes scraped from the internet. This dataset provides columns such as person_id, title, experience, and skills. Person_id is an int while the other columns are strings.

Job Description Dataset

This dataset was created through the Python Faker library and was refined using ChatGPT, meaning that this dataset is all synthetic data. This dataset provides columns such as job_id, experience, qualification, job title, skills, and responsibilities. Job_id is an int while the other columns are strings.

Data Preparation

In order to narrow down the scope of the project, we decided to only train our model on CS and tech-related jobs and resumes. We started by downsizing both datasets down to the first ten thousand rows. Then, to ensure that our resumes were unique, we merged all the rows by person_id. Since our resume data set was also separated into multiple different files, we needed to merge them all into one data frame that represented the entirety of the resume data. Once we got two data frames, one for the job descriptions and one for the resumes, we needed to filter them until only the CS and tech-related jobs remained.

To do so, we used text embeddings, specifically, the SentenceTransformers Python Module. In order to filter, we needed to generate a reference vector to compare with. Using ChatGPT, we generated two lists, one with CS job title names and CS keywords, which were encoded using the SentenceTransformers model. Then, we encoded both the resume and job datasets, checking the similarities between the reference vectors and the dataset embeddings using their cosine similarity. Once done, we manually checked the highest similarities, deciding on an initial threshold of 37.5% similarity for both the jobs and resumes. Finally, we created a mask to get the indices of those jobs and resumes, ending up with our resume data frame having a shape of (7856, 5) and our job data frame having a shape of (3217, 6).

Model Selection and Evaluation

Generating Labels for Binary Interview Classification

To generate the labels for our binary classifier, we used Gemini's API. We chose Gemini 2.5 Flash Lite to be the model for generating our response, as this one had the most requests per minute (RPM) while also having a high requests per day (RPD) limit. The downside of this is that the model is not as accurate as other models when parsing through text and creating a response, but it is much more efficient than other models. Our prompt was:

> You are an expert recruiter for tech-related and cs-related jobs. Given a candidate's resume and a job description, rate how likely the candidate is to get an interview for the job with either 1 or 0:
>
> - 1 means they will get the interview.
> - 0 means they will not get the interview.
>
> Return your answer strictly as either 1 or 0.
> Resume: {resume_text}
> Job Description: {job_text}

Since each prompt and request takes some time to run and would take much longer on our filtered datasets, we decided to filter the data we would use to generate the labels, only using the most representative resume and job pairs. Again, we used our embeddings to compute the cosine similarity, sorting it from smallest to largest similarity values and choosing the top 25 resumes. We also chose 50 jobs to generate 1250 total resume-job pairs, which takes an average of 9 minutes to generate labels for. If we had the time and usage for the API, we could possibly do more pairs to make the labels more accurate for the model training.

Binary Interview Classification Models

First, in order to get the X and y training data, we use the resume_id and the job_id to get the indices from our embedding matrix, and let the vectors at those indices be the X and y. Then, we used train_test_split to get X_train, X_test, y_train, y_test. Since the shape of our X was (400, 2, 384), we used PCA to reduce the dimensionality so it was only 2 dimensional, getting X_train_reduced_full and X_test_reduced_full. Then, we trained the Random Forest Classifier (RFC) on our data.

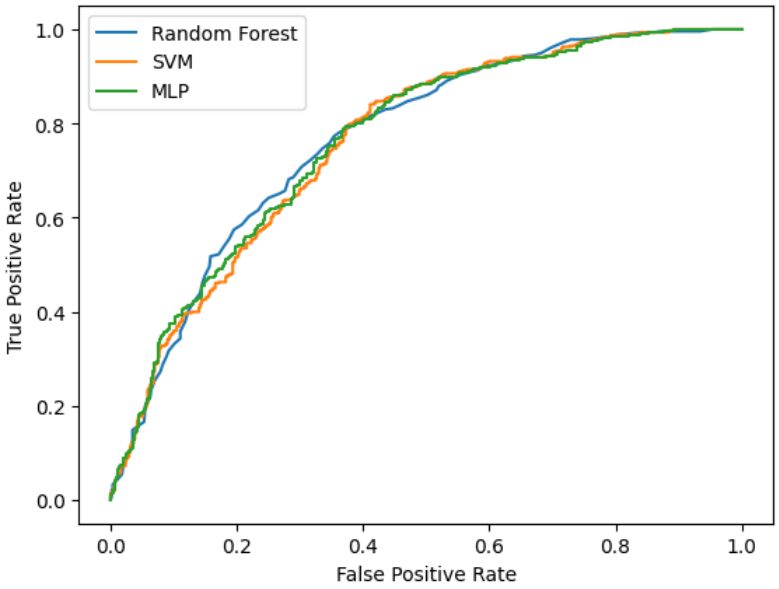

Figure 1: ROC Curves for all Models

Finally, we fit X_train_reduced_full and y_train with the RFC, and ran predictions against the X_test_reduced_full data, getting an accuracy score of 63.2%. We chose the RFC because of it's strong capabilities as a binary classifier. Next, we used the SVM Classifier to train our data, following a similar method as with the RFC. When running X_test_reduced_full against the classifier, we got an accuracy score of 68.8%.

Finally, we trained our dataset with an MLP Classifier, which due to it being a neural network, is a good choice for our model training. When running our test dataset against the MLP classifier, we managed to get an accuracy of 70.4%, which is the highest one of all the classifiers. This makes sense since MLP is better suited for complex and non-linear data like ours. SVM also works well because svm excels at small datasets.

Generating Labels for Multiclass Subjective Classification

To generate the labels for the Subjective Feedback System, we used a similar method with Gemini as we did for the binary classifier labels. We chose the same Gemini model, but changed the way we selected the data that would be used to generate the labels. For the subjective feedback, we decided to classify it into three categories: not related, moderately related, and closely related. We decide this by looking the candidate's skills and comparing it the skills of the jobs. Initially, we chose the first 100 rows for the resume skills and the job skills, pairing them together and running each pair against the model with the following prompt:

> You are an expert recruiter for tech-related and cs-related jobs. Given a candidate's skills and a job description's skills, compare each list of skills, and return the following:
>
> - 0 if the skills are not related.
> - 1 if the skills are moderately related.
> - 2 if the skills are closely related.
>
> Return your answer strictly as either 0, 1, or 2.
> Resume: {resume_text}
> Job Description: {job_text}

Multiclass Feedback Classification

For the multiclassifcation feedback, we were only able to classify the skills. We did this by getting the embedding of the skills and finding the minimum similarity of the embeddings for each category. Based on the similarity for each category, we could sort the relationship between the skills into their category of not related, moderately related or closely related. We achieved an accuracy of 31.66%, which is very low. This is because we used the minimum similarity between the two. We should've used the median or another model like SVM or logistic regression.

Evaluation

For each of our models, we check the ROC curve to see how the classifiers performs at different thresholds. Based on the curve, if there is a larger area under the curve, then our models did well. While we do have the curves in a good general shape, having a decently high true positive rate, it is not the best that it could be. This is also reflected in the other way we evaluated our models, which was using the generated test dataset on our trained models and seeing what the accuracy was. The average accuracy of the models was 67.4%, which is not very good for a binary classifier.

Error Analysis and Improving Model

The models do not perform that well on classifying whether the resume candidate will get an interview or not. Currently, the highest accuracy is 70%, which is not the most

accurate result for a binary classifier. Our method for the subjective feedback is also not well implemented since our accuracy is low.

While generating our labels, we decided to go from generating 400 labels for the binary classifier to generating 1250 labels since we decided to purchase a higher tier for the Gemini API, which made the generation process much faster. Because of this, our accuracies for the RFC and the SVM went down from 65% to 63.2% and 70% to 68.8% respectively. However, our accuracy for the MLP went up from 67% to 70.4%, which was a sign that adding more labels improved our models and helped negate bias and overfitting when training. Then, for our Feedback Classification, we initially had 100 generated labels which we used for the embeddings and our algorithm to get feedback. However, we realized that this was not enough for a good accuracy so we changed it and generated 300 labels instead. However, we got a lower accuracy score when we generated more labels, showing that our data was overfitting.

For improving our models in the future, some ideas would be to generate better and more accurate labels. This could be done by using a better model for the LLM that has higher accuracy while still maintaining efficiency. We could also use better APIs than Gemini since Gemini is not the best LLM that exists. Again, if we had more time to run and more usage for the Colab and the API, we would be able to generate more labels for our model training, possibly improving the accuracy. We also realized that we could sample our data randomly when filtering and selecting which resumes and jobs would be used to generate the labels. This would allow for higher representation of our dataset since it would randomly select rows instead of using the rows in front, which could have skewed or biased data towards one specific job compared to other jobs. If we had more time, we could also fix our algorithm for the subjective feedback system, exploring other methods of embeddings and using tokens instead of full documents for our datasets. For the subjective feedback, since generating more labels did help our accuracy, it is possible that there is something wrong with our generation method or criteria, so if we had time, we would look at that again and redo it.

Conclusion

In conclusion, to train and ensure that there were objective ways of evaluating and training our model, we used LLMs to generate labels to train our models. We found out that the best model of the binary classifiers was the MLP, which was expected. For the feedback, we discovered that our method was not the best and that we should have explored other methods of training the models and getting accurate feedback.