University of California, Riverside

CS105 Data Analysis Final Project – Resume Screening - Applicant Tracking System (ATS)

Henry Yost, Riya Ashok, Angelina Jordan, Gokul Giridharan, and Refugio Zepeda Jr

Professor: Elena Strzheletska

A report submitted for the final project of CS105 - Data Analysis

June 9, 2025

Abstract This project explores the process and development of an Applicant Tracking System (ATS) for screening and ranking resumes based on their semantic similarity to a job description and ideal resumes. In this project, we use SBERT for embeddings, cosine similarity for filtering, and train a Support Vector Machine (SVM) to predict binary classify candidates based on their resumes. This document shows the results, and how we determined many of the values/processes for our finished project.

Contents

0.1 Project Outline

Presentation Link

For our project, we decided to try to create an Applicant Tracking System (ATS), where we could predict whether an applicant was qualified or not using a supervised model. Additionally, the project would behave as a screening system, filtering and sorting resumes, helping automate the process. Lastly, the resumes would be ranked based on how well they align with a job application in their field.

0.1.1 Phase 1 – Filtering

First, using the resume.csv file, we classified the 24 categories into 5-6 buckets and saved each as a .csv file. Each bucket represents a broader term, such as tech and health. This helps significantly reducing the large count of 2400+ resumes into a significantly more manageable size. The next step is to pre-process the resumes with RegEx. This step will include removing special characters, punctuation, and indentations. After pre-processing the data, we can use hard criteria such as specific education degree requirements to eliminate candidates that do not fit the hard criteria. This helps quickly remove unqualified candidates.

Additionally, for further filtering, we SBERT embeddings of the resumes and job description to compute a cosine similarity value. If the value was over a threshold, then the candidate's resume would move on to Phase 2.

0.1.2 Phase 2 – Ideal Resume Filtering & Human Labeling

Using the SBERT embeddings of the resume's that passed Phase 1, we can compare them to ideal resumes that have been selected with specific criteria. The ideal resumes are converted to an ideal cluster, where cosine similarity is used to compute the similarity of the candidates resumes to the ideal cluster. We are using multiple ideal resumes over a single resume, which gives us a significantly better baseline and prevents overfitting to a single ideal resume.

Additionally, albeit a tedious process, the remaining resumes need to be labeled as qualified or not by a human. This is partly for the supervised learning labels, and that currently no machine or mathematical formula would be able to perfectly determine whether candidates are qualified, especially if their embeddings are close in space.

0.1.3 Phase 3 – Supervised Training

The last phase is to train two types of SVM model's. One SVM's architecture is built from the ground up, and the other is from the scikit library. We trained both the SVM's on the most filtered resume dataset, and the human generated labels. Lastly, we are able to pass in resumes to the model and determine if they are qualified or not.

0.2 Installation & Usage

To install and run this project locally, please follow the written instructions below. Moreover, this installation guide assumes that you are working in a Python environment.

1. Fork the Repository

- Visit the GitHub repository: https://github.com/henry-AY/ResumeScreening
- "Fork" the repository to create your own copy.

2. Clone the Repository

```
git clone https://github.com/<your-username>/ResumeScreening.git
cd ResumeScreening
jupyter notebook
```

3. Open the notebook in a conda environment

This will allow you to run the entire notebook (Note: this might take some time due to many of the computations and graphs).

0.3 Data Pre-processing

As mentioned, the first step is to "bucketize" all of the categories and then pre-process the relevant buckets. Our buckets followed a natural structure as shown below in the bulletined list.

- **Tech**: Information-Technology, Engineering, Digital-Media, HR
- **Health and Wellness**: Healthcare, Fitness
- **Business and Finance**: Business-Development, Sales, Consultant, Finance, Accountant, Banking
- **Creative and Public**: Arts, Designer, Public Relations, Teacher, Advocate
- **Industrial**: Agriculture, Automobile, Construction, Aviation, BPO
- **Miscellaneous**: Chef, Apparel

Additionally, the resume text was cleaned using regular expressions to lowercase all letters, remove unwanted special characters, and standardize spacing. It is important to note that a new dataframe was created from the original, including only the relevant categories and all of the pre-processing.

0.4 Exploratory Data Analysis (EDA)

0.4.1 Dataset Exploration & Variables in the Dataset

The main dataset that we are using is Resume.csv from Kaggle, and this dataset contains over 2484 resumes, each labeled with a specific job category. The dataset was approved by the professor, as we are doing a unique project.

- **ID** – The ID is the unique identifier for each resume
- **Resume_str** – The content/words of the resume (including formatting)

- **Resume_html** – The contents of the resume in HTML format
- **Category** – The labeled job field for the respective resume (HR, Tech, Finance, Aviation...)

Dataset Link

0.4.2 Dataset Visualizations

Distribution Histogram

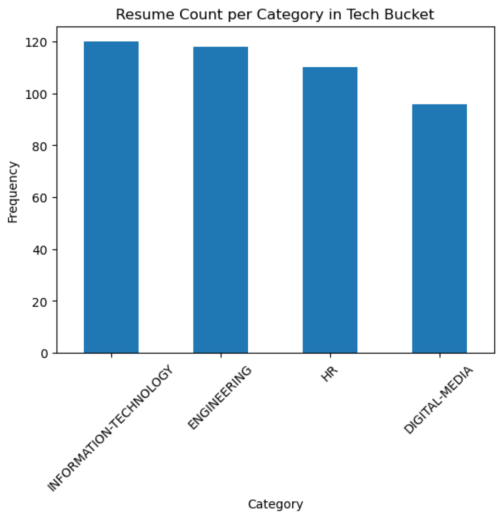

Figure 1: Resume Distribution in Tech Bucket

We can see that our distribution for the resumes is almost equal, and all 4 categories (IT, Engineering, HR, and Digital-Media have over 100 resumes. While this is not a particularly useful EDA for unsupervised or supervised learning, it helps us visualize the distribution of tech resumes in the dataset.

Word Frequency Analysis

Moreover, We were wondering what the most common words in all of the resumes were. We decided the best way to visualize this was with a word cloud graph.

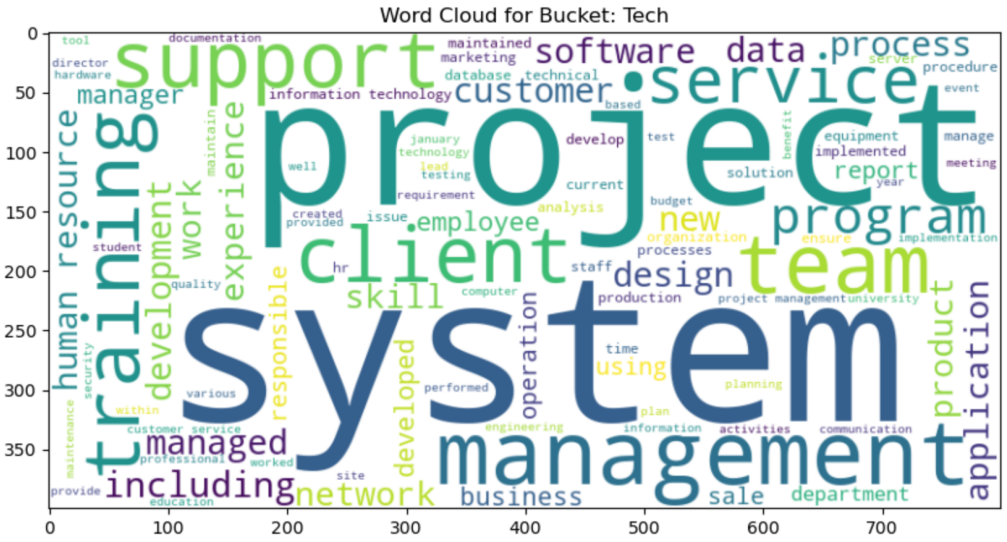

Figure 2: Word Cloud for the Resumes in the Tech Bucket

The graph above is a Word Cloud graph of the tech bucket resumes in the dataset, highlighting domain-specific terms that are common across each tech resume. The larger the word, the more frequent among the resumes. Thus, high frequency terms such as: project, system, software, development, application, data, network are all indicative of the technical knowledge and requirements that are common in the datasets tech resumes. Additionally, we see a lot of high-frequency terms that are indicative of collaborative and/or leadership roles, such as: support, management, managed, and team. This EDA relates to our project, and specifically the supervised aspect, because the supervised model will be trained on SBERT text embeddings, which are then classified.

Education Level Heatmap

Additionally, we wanted to see the relationship between each of the 4 job categories (IT, HR, Engineering, and Digital-Media) and their respective education levels. This was done using a heatmap, because in this instance it felt the most appropriate.

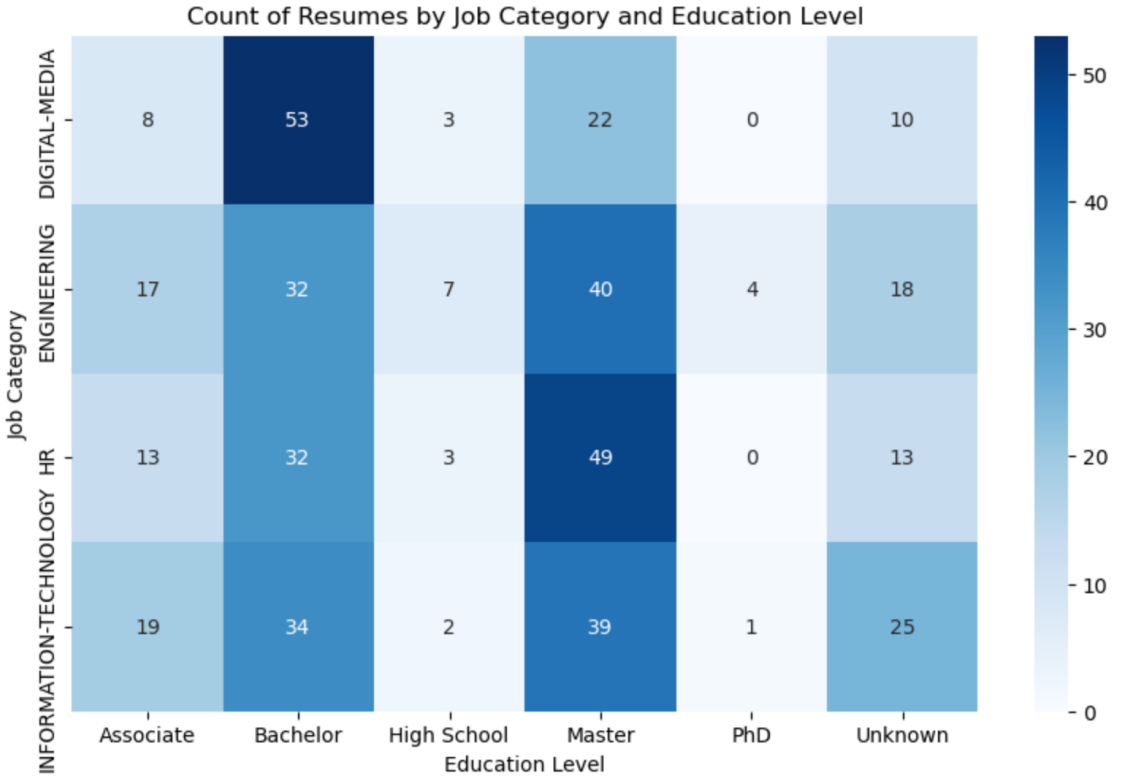

Figure 3: Education Level in relation to Job Category

This is a heatmap that shows the count of different education levels against the 4 job categories in the tech bucket. This is important because we plan on passing the education level as a feature, and looking at the distribution among the dataset can give us important insight. For example, Engineering and IT have good distribution across all degree types, with Masters being among the highest in both groups. PhD's are rare, especially in HR and Digital Media, which have 0 PhDs. This makes sense because generally, engineering students do not pursue a PhD due to specialization not being typically valued unless the role specifically demands so.

Interestingly, the Unknown category has a concerning number of resumes. However, I believe that this is due to more experienced individuals in the workforce, who have been in the industry for 15-20+ years, and their experience is more relevant than their education. However, this would be interesting to look into more.

t-SNE Embeddings Explanation

Lastly, we were interested in the SBERT embeddings and their relationships to each other in 2D-space. This was interesting to us, because it is the foundation of our supervised model, and it is really interesting to see the natural clusters forming purely on the semantics of the resumes and their categories.

What is SBERT? SBERT stands for Sentence-BERT, which is an expansion of the BERT (Bidirectional Encoder Representations from Transformers) model designed to generate meaningful sentence-level embeddings. The original BERT model is very effective for understanding text contextually, it was not optimizing for comparing sentences directly. This is where SBERT

comes in, SBERT is able to quickly compute semantic similarity between sentences, making it a great option for this project.

In our case, we use SBERT to encode both resumes and job descriptions into numerical vectors that capture their meaning. While humans find it difficult to understand these embedding vectors, machines and supervised learning techniques are able to understand them extremely well. These embeddings were then compared using cosine similarity to measure how semantically close a resume is to a given job post or an ideal cluster.

Embeddings are numerical vector representations of text. When you pass in a sentence through the SBERT Model, it outputs a high-dimensional vector (e.g., 384, 786 dimensions) that represents the semantic meaning of the encoded text. As mentioned previously, these embeddings vectors allow us to compare any two pieces of text with mathematical functions.

What is a t-SNE Plot? t-SNE stands for t-Distributed Stochastic Neighbor Embedding, and it is a machine learning algorithm used to reduce high-dimensional data (such as the SBERT embedding vectors) down to a two or three dimensions, thus we are able to graph and visualize them.

As mentioned previously, the SBERT embeddings exist in extremely high dimensions, which is impossible to visually directly. t-SNE helps us simplify these embeddings onto a 2D-plane. t-SNE is used significantly throughout this project to visually plot resume clusters based on their semantics. We have also included other metrics (such as job description and ideal clusters) in the embeddings plot to visually see the differences between the resumes.

This makes the selection process not only quantitative (cosine similarity scores) but also interpretable , as we can visualize how resumes relate to each other and the job.

t-SNE Embeddings Visualization

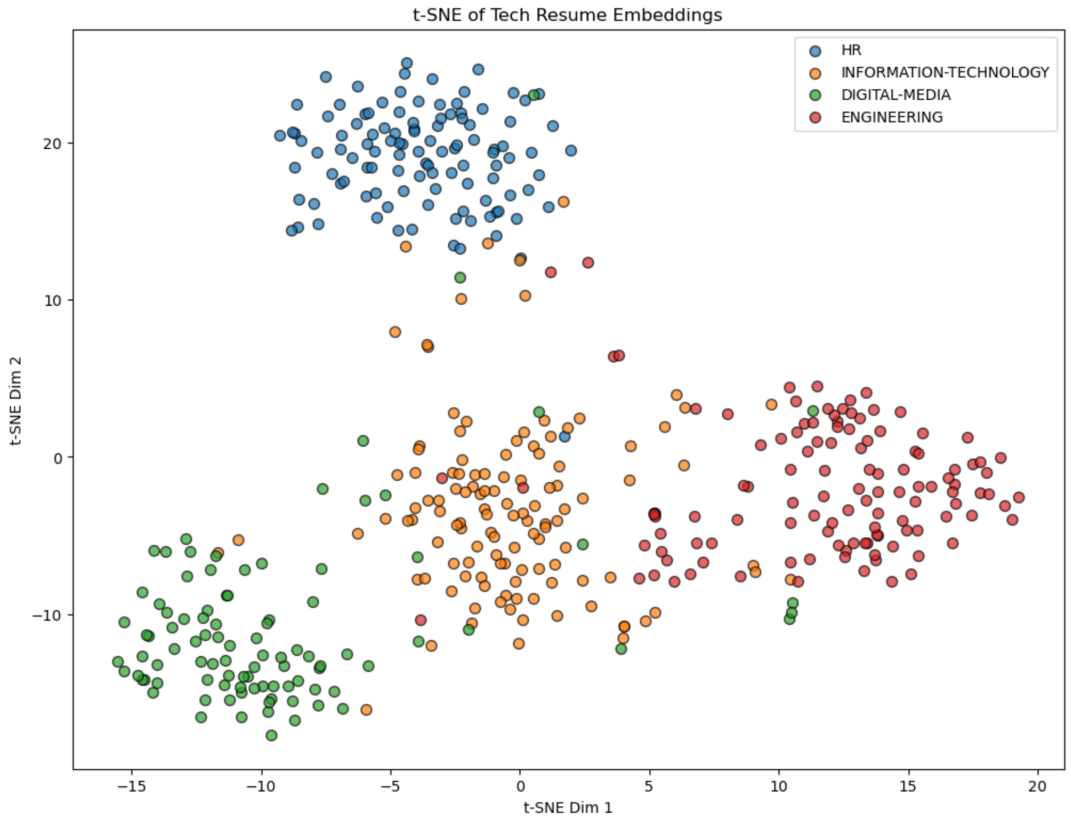

Figure 4: t-SNE of Tech Resume Embeddings

t-SNE Visualization of Resume Embeddings: This graph is a visual representation of the flattened SBERT embeddings of the tech resumes. The position of the resumes is based on their embeddings; thus, the closer to each other, the two are more semantically similar. Thus, points clustered around each other mean they have greater semantic similarity.

As mentioned in the explanation above, the clusters of the SBERT embeddings mean that they are semantically more similar. Thus, it is normal and expected to see the natural clusters that are forming. Each of the categories in the tech bucket (HR, IT, Digital Media, and Engineering) has formed natural clusters because semantically their resumes are similar, and arguably quite distinct from one another. However, interestingly, there are a handful of outliers, and some overlap primarily between IT and Engineering. This is logical because IT and Engineering have overlapping disciplines. Thus, it suggests overlapping language in resumes from both categories, such as common phrases: "project management", "training", "team collaboration", etc, thus it is more difficult to distinguish between the two with SBERT embeddings.

The overlap between IT and Engineering, and the similar overlap in the other categories, poses a risk for the supervised model. The supervised model might misclassify resumes between IT and Engineering. While these categories and resumes may blur in the real world, as mentioned previously, the non-clean decision boundary might make it difficult for the supervised model. However, further cleaning and comparison with the job description and ideal resumes will hopefully remove this ambiguity.

0.5 Tesla Software and Energy Engineer

0.5.1 Ideal Candidate Resume Selection for Tesla Software and Energy Engineer Role

We decided to pick Ideal candidates to compare against, because similar to real-world hiring, companies/recruiters often compare candidates to a handful of 'perfect fit' resumes. We can mimic this process by creating an ideal cluster and calculating the similarity to those 'perfect fit' resumes.

These resumes were selected as the best matches for the Tesla Software and Energy Engineer position. Selection was based on skills, experience, and the overall role alignment. We filtered through the full resume dataset using both strict filtering (education, degree, etc.) and semantic similarity (SBERT embeddings). Resumes were then ranked for their similarity to the job position, and then further evaluated by a group member for their relevance to the position, skills, etc... These two top resumes were selected as the ideal candidates for said position at Tesla.

0.5.2 Job Position Source & Description

LinkedIn Tesla Job Application

Responsibilities:

- Develop, enhance, and debug new and existing real-time software in C/C++ in embedded RTOS environments.
- Collaborate with hardware and systems teams to build testing infrastructure.
- Contribute to the design and bring-up of state-of-the-art HIL/SIL validation infrastructure.
- Implement software tests for HIL/SIL systems.
- Assist in developing tools for testing and system integration.
- Own the implementation of software and firmware features end-to-end.

Qualifications:

- Pursuing a degree in Computer Science, Electrical Engineering, Physics, or a related field.
- Strong understanding of Python and ability to debug simple circuits (desired).
- Solid foundation in electrical and electronic fundamentals to understand schematics (desired).
- Mindset oriented toward test-driven development (desired).
- Proactive, engaged, and solution-focused while tackling challenging problems.
- Takes ownership of assignments and is accountable for overall team success.
- Capable of delivering high-quality C/C++ code for embedded systems.

0.5.3 Ideal Candidates

	ID	Resume_str	Category	Cleaned_str	education_level
1695	82246962	TEST ENGINEERING Profile I a...	ENGINEERING	test engineering profile i am seeking the chal...	Associate
1802	12472574	QA ENGINEERING TEAM LEAD Care...	ENGINEERING	qa engineering team lead career overview eight...	Unknown

Figure 5: Ideal Tesla Job Candidates

As mentioned previously, these candidates were selected based on strict requirements, skills and experience, and overall alignment of the role. Interestingly, candidate one has an "Associate"'s degree, and candidate two has an "unknown" degree. However, this is negligible given that their experience in the relevant sectors heavily outweighs their level of education. Additionally, this will have a near negligible impact on the embeddings similarity scores.
For cleanliness and best practice, both ideal candidates were removed from the dataset, and now we start the process of extracting relevant information from the job description. The primary focus was to extract the relevant degrees that would qualify a candidate for the Software and Energy Engineer position at Tesla.

0.5.4 Extracting Qualification Requirements from Job Descriptions

To filter candidates aligned with the Tesla Intern position, we parsed the job description to extract specific degree requirements using regular expressions. We targeted the phrase structure beginning with `"Pursuing a Degree in..."`.
The following regular expression was used to extract the required degree fields:

```
Pursuing a Degree in (.+?)(?:\.|$)
```

This RegEx structure is designed to break down degree requirements following a structure "Pursuing a Degree in...". The $(.+?)$ is a non-greedy way to capture the degree fields. The $(?:\. | \$)$ behaves as the non-capturing group, meaning it does not group any strings and simply ends the RegEx capture after a period or end of the string.
The extracted degree fields included:

- Computer Science
- Electrical Engineering
- Physics

In addition to the extracted fields, relevant fields were included, and promptly replaced with fields of study not explicitly in the job description, however, commonly considered equivalent in hiring contexts:

- Software Engineering
- Computer Engineering
- Mechanical Engineering
- Data Science
- Robotics
- Information Technology

- Applied Physics

This was used as a hard criteria for candidates who did not meet the academic background requirements for the position. This was done with a simple strict word search through the resumes.

0.5.5 Resume Similarity and Embedding Visualization

Now, a majority of the non-relevant candidates in terms of hard criteria and degree have been removed from the applicant pool, however, there is still plenty of further processing to do. However, first we wanted to look at the embeddings of the current resumes, and where they lied in a 2D-space against each other and the Tesla job description. The resulting 2D plot is a visual representation how resumes relate to each other and to the job description.

- Each resume is colored-coded by its original category (e.g., IT, Engineering, HR, etc.).
- The Tesla job description is represented as a black `X` marker.

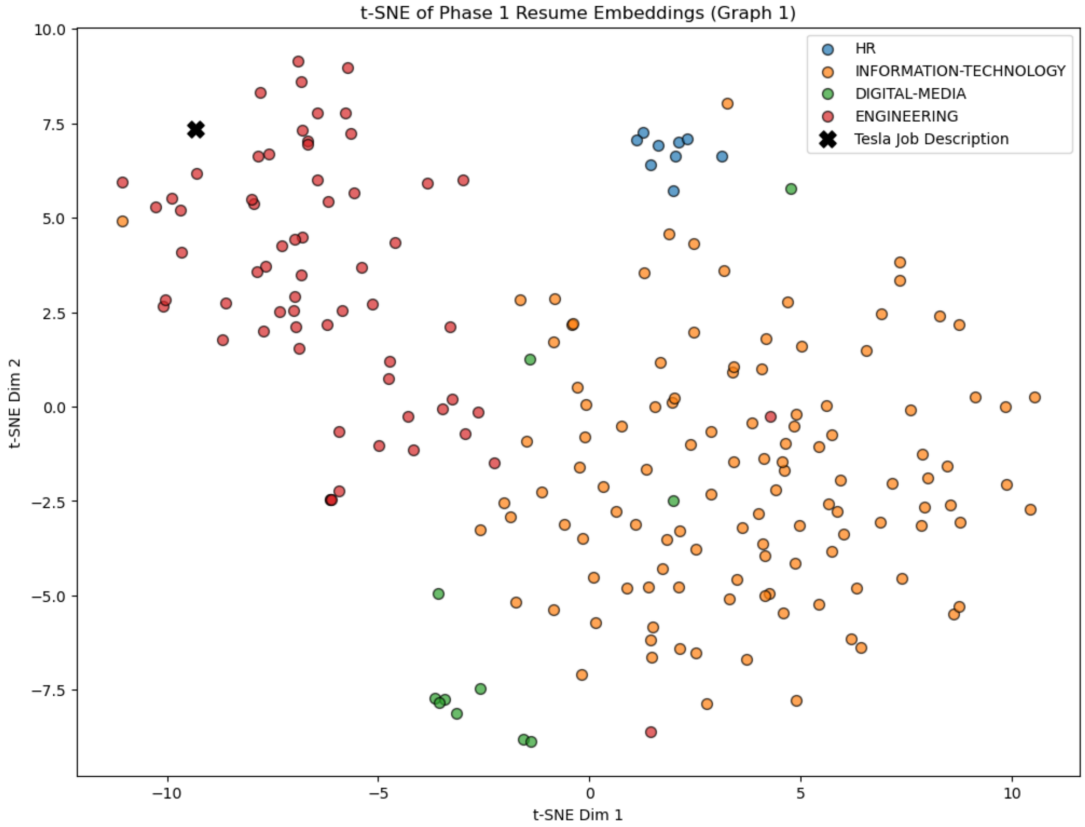

Figure 6: t-SNE of Tech Resume Embeddings after hard criteria

t-SNE Visualization of Resume Embeddings and Job Description: This graph is a visual representation of the flattened SBERT embeddings of the closest resumes in terms of similarity. In addition to the SBERT embeddings of the Job description (marked by an 'X'). Additionally, similar to the previous graph, the position of the resumes is based on their embeddings; thus, the closer to the job description, means the two are semantically more similar.

This t-SNE plot shows the embeddings of the resumes that remained after the hard criteria cleaning of Phase 1. These are all of the candidates who meet the baseline requirements for the position. Additionally, you may notice that the embeddings are still quite spread out; this is because we have not applied semantic filtering yet. However, in general, we can see that a significant portion of the resumes not in the relevant field have been removed from the selection process, thus leaving primarily IT and Engineering, which overlap quite well with the job position.

0.5.6 Cosine Similarity Threshold

In order to determine how well each resume aligns with the Tesla Job Description semantically, we used a pre-trained SBERT model `all-MiniLM-L6-v2` to compute the semantic embeddings. Consequently, each of the resumes and the job description were encoded into high-dimensional vectors. To evaluate how closely the resumes align with the job description, we compute the cosine similarity using the formula below, where the embeddings of resume i, denoted as $\tilde{r}_i$, and the job description embedding $\tilde{j}$:

$$\text{Cosine Similarity}(\tilde{r}_i, \tilde{j}) = \frac{\tilde{r}_i \cdot \tilde{j}}{\|\tilde{r}_i\| \, \|\tilde{j}\|} \tag{1}$$

These calculations were performed only on resumes that passes the initial hard filtering stage of Phase 1. Thus, the resulting cosine similarity scores formed a distribution which we visualized using a histogram to determine an appropriate similarity threshold to shortlist candidates.

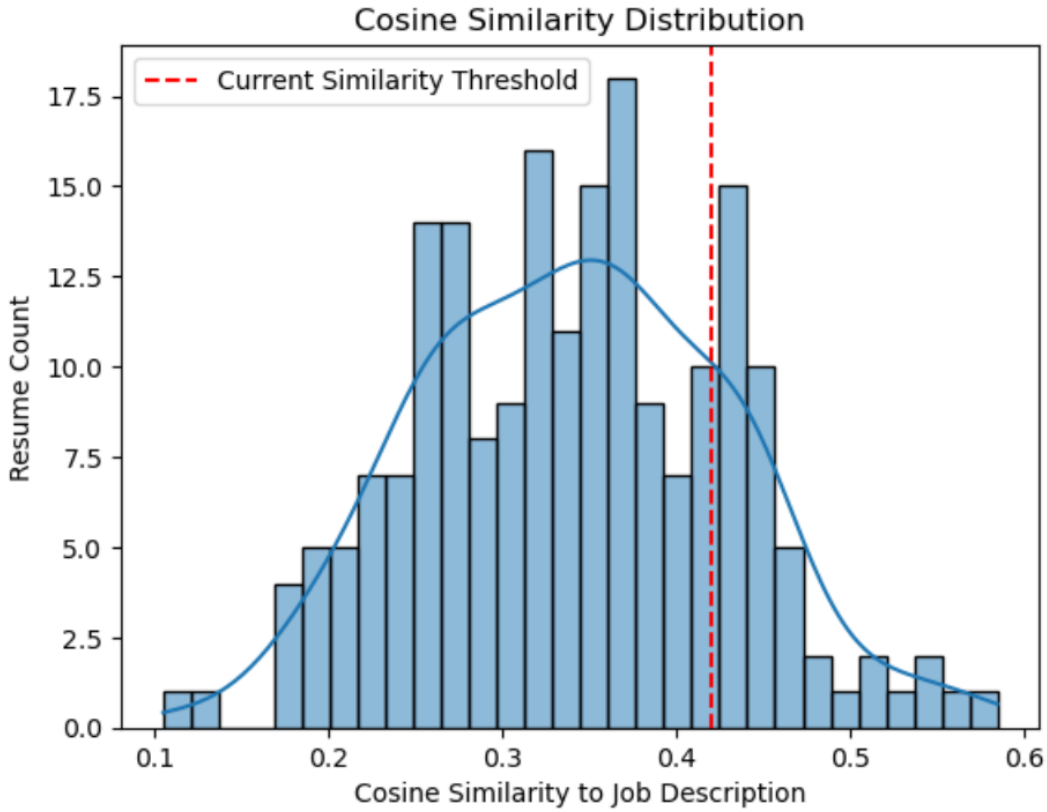

Figure 7: Distribution of Cosine Similarity Scores

This distribution shows us that there is a peak at around the 0.36 cosine similarity score and

a natural fall at around the 0.38 similarity point. Therefore, we picked a threshold value of 0.42 to shortlist the candidates. This ensured that resumes with a moderate to significantly high semantic match would remain as potential candidates. Additionally, this value, although low, balances inclusion with selectivity.

Thus, we now had 42 candidates, as shown in the function call `df.info()` below.

```
<class 'pandas.core.frame.DataFrame'>
Index: 42 entries, 222 to 1804
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   42 non-null     int64
 1   Resume_str           42 non-null     object
 2   Category             42 non-null     object
 3   Cleaned_str          42 non-null     object
 4   education_level      42 non-null     object
 5   matching_degree      42 non-null     bool
 6   Computed_similarity  42 non-null     float32
dtypes: bool(1), float32(1), int64(1), object(4)
memory usage: 2.2+ KB
```

0.5.7 End of Phase 1 Embedding Visualization

Similarly, we visualized the t-SNE embeddings of the resumes an the job description to see how well the current candidate selection aligns semantically with the job descrpition (and each other). This plot, allows us to also gain insight into the effectiveness of our filtering and shortlisting process.

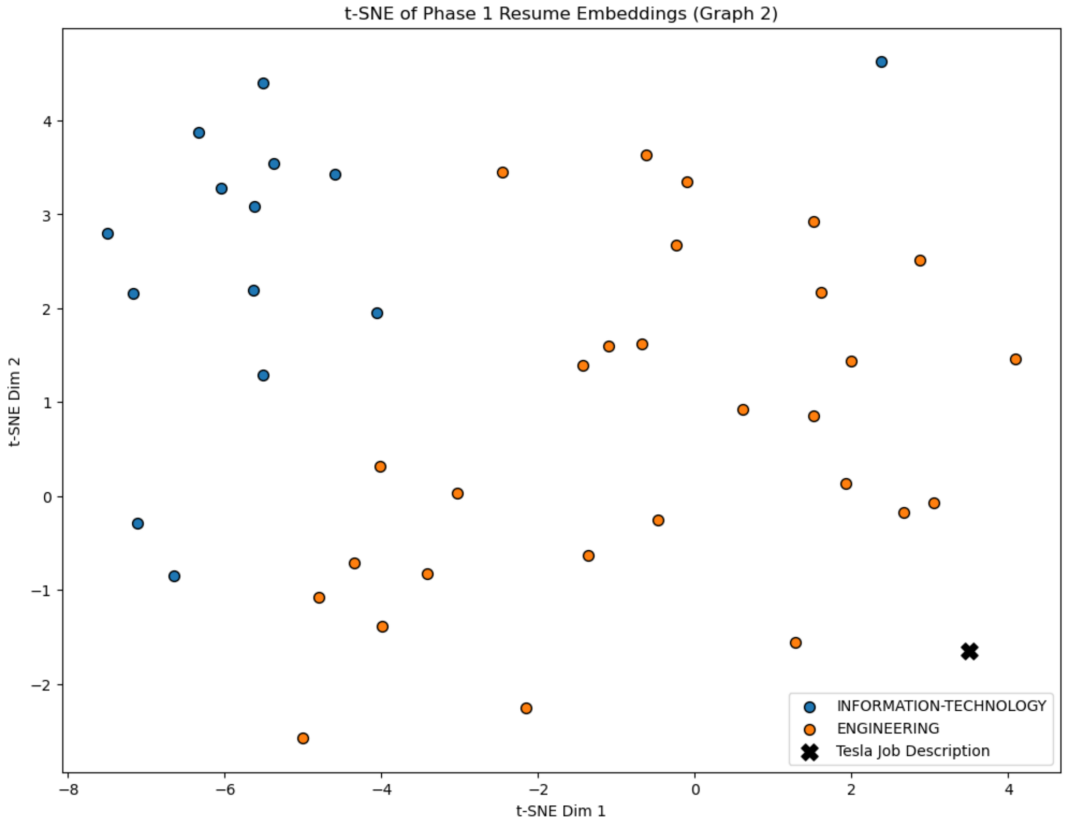

Figure 8: t-SNE of Tech Resume Embeddings

t-SNE Visualization of Resume Embeddings and Job Description: This graph is a visual representation of the flattened SBERT embeddings of the closest resumes in terms of similarity. In addition to the SBERT embeddings of the Job description (marked by an 'X'). Additionally, similar to the previous graph, the position of the resumes is based on their embeddings; thus, the closer to the job description, means the two are semantically more similar.
This t-SNE plot shows the embeddings of the resumes that remained after the process of phase 1. We can see that the majority of the applicants who were initially accepted past the baseline requirements have now been removed, and the applicant pool has been simplified to those who have moderate to high similarity to the job description. Interestingly, a majority of the overlap in the original bucket between the 4 categories has mostly been removed, and now we see only two categories: IT and engineering.

0.5.8 Phase 2 & Comparing Resumes to Ideal Cluster

In Phase 2, we shifted our focus to refining the current selection of candidates by comparing each resume to an "ideal" selection of resumes. The process was described earlier in the paper, and the ideal subset consists of handpicked resumes based on strong aliment with the job description. The goal was to compute the semantic similarity score between each cleaned resume and the an ideal cluster.
Thus, to achieve this, we first computed the average SBERT embedding vector for all resumes in the ideal subset:

```
def average_embeddings(subset):
    avg_embeddings = subset['Cleaned_str'].apply(lambda x: m_SBERT.encode(x))
    matrix_embeddings = np.vstack(avg_embeddings)
```

```
    avg = np.mean(matrix_embeddings, axis = 0)
    return avg
```

Listing 1: Computing Average Embeddings of Ideal Resumes

Then, for each resume in the cleaned dataset, we calculated its cosine similarity score with the average embeddings of the ideal resumes. This gives us a measure of how semantically similar each candidate is to the ideal cluster. Each of the computed scores are stored in the `Phase2_computed_similarity` column.

```
<class 'pandas.core.frame.DataFrame'>
Index: 42 entries, 222 to 1804
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   ID                           42 non-null     int64
 1   Resume_str                   42 non-null     object
 2   Category                     42 non-null     object
 3   Cleaned_str                  42 non-null     object
 4   education_level              42 non-null     object
 5   matching_degree              42 non-null     bool
 6   Computed_similarity          42 non-null     float32
 7   Phase2_computed_similarity   42 non-null     float32
dtypes: bool(1), float32(2), int64(1), object(4)
memory usage: 2.3+ KB
```

0.5.9 Visualizing Resume Embeddings Against Ideal Clusters

Similarly to previous embedding plots, to better understand the semantic clustering of resumes with respect to an ideal cluster. We used the t-SNE algorithm to reduce the high-dimensional embeddings to a two-dimensional plot, which helps us interpret the relative 2D distance of the each resume to the ideal cluster.

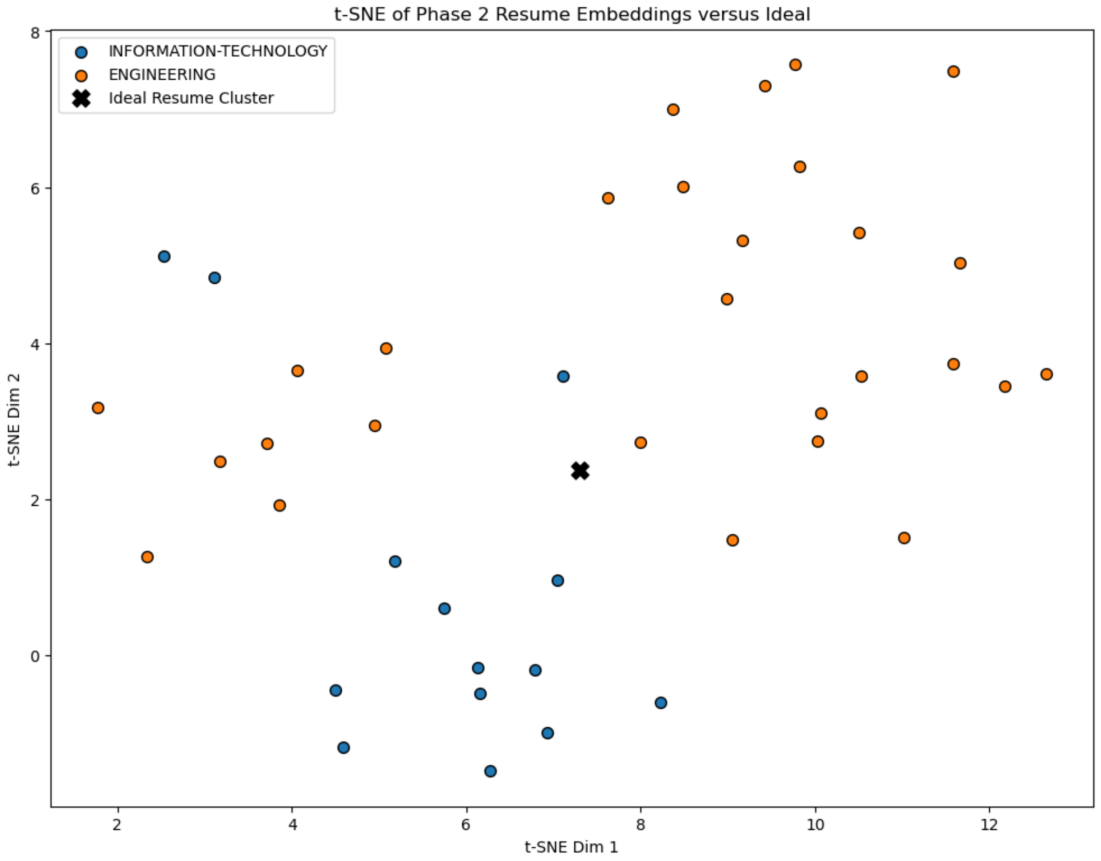

Figure 9: t-SNE of Tech Resume Embeddings to Ideal

This t-SNE plot shows us the relative positions of the Ideal Resume Cluster and the remaining resume embeddings in 2D space. Interestingly, we can see that the Ideal Resume cluster is located more centrally within the plot, whereas the job description is located near the corners. This suggests that the Ideal Resume cluster shares more semantic overlap with not only the job description but also the remaining resumes. Therefore, we have effectively created a closely knit group of relevant resumes.

Moreover, given the Ideal Resume cluster is roughly in the center, between both Engineering and IT embeddings, it suggests and corroborates the previous information that we found, that there can be a significant amount of semantic similarity between IT and Engineering resumes. It's important to note that the axes in the above plot are not of the same size as previous plots; while the embeddings are visually spread out, their semantic distances are quite small because of dimensionality reduction when plotting. Thus, these embeddings are extremely closely related to each other, especially compared to the previous plotting of the entire bucket.

0.5.10 Human Labeling

While automated semantic similarity scores are useful for first pass filtering, they may overlook subtleties, especially when many of the embeddings are very similar. Therefore, we decided to use human labeling for the last part of phase 2 to further refine our shortlist of candidate. Additionally, this would behave as a to generate labels to train the supervised model.

For the remaining 42 resumes, we manually reviewed the resumes based on their presence of key qualifications and their similarity score to the ideal cluster. Specifically, we marked them qualified (1), or not qualified (0) based on the following criteria and keywords

- **Programming Languages:** `C++`, `C`, `Python`
- **System Experience:** Familiarity with RTOS (Real-Time Operating Systems), experience in building infrastructure, or working with HIL/SIL validation environments
- **Hardware Knowledge:** Ability to understand schematics, circuits, and perform low-level integration
- **Implementation & Integration:** Experience in implementing software/firmware systems or contributing to system integration pipelines
- **Soft Skills:** Teamwork, proactivity, ownership, and problem-solving under challenging conditions

Additionally, candidates resumes were evaluated based on their Phase 2 similarity score (to the ideal cluster) and their educational background, however, this was a more loose requirement. These scores behaved as a suggestion and helped us prioritize candidates who were not only well-qualified, but also semantically similar to the ideal resume clusters. This left us with a snippet of the dataset as shown below:

	ID	Resume_str	Category	Cleaned_str	education_level	matching_degree	Computed_similarity	Phase2_computed_similarity	Manual_Label
0	91635250	Christopher Townes Summa...	INFORMATION-TECHNOLOGY	christopher townes summary knowledgeable infor...	Unknown	True	0.445310	0.624749	1.0
1	15651486	DIRECTOR OF INFORMATION TECHNOLOGY ...	INFORMATION-TECHNOLOGY	director of information technology career over...	Associate	True	0.432384	0.608571	0.0
2	52246737	INFORMATION TECHNOLOGY PROVISIONING T...	INFORMATION-TECHNOLOGY	information technology provisioning technician...	Associate	True	0.440459	0.637156	0.0
3	24913648	INFORMATION TECHNOLOGY SPECIALIST ...	INFORMATION-TECHNOLOGY	information technology specialist professional...	Bachelor	True	0.423039	0.584070	0.0

Figure 10: Head of Final Data Frame with Qualification Labels

Additionally, calling `df.info()` gives us a little more insight into structure of the data frame.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42 entries, 0 to 41
Data columns (total 9 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   ID                          42 non-null     int64
 1   Resume_str                  42 non-null     object
 2   Category                    42 non-null     object
 3   Cleaned_str                 42 non-null     object
 4   education_level             42 non-null     object
 5   matching_degree             42 non-null     bool
 6   Computed_similarity         42 non-null     float64
 7   Phase2_computed_similarity  42 non-null     float64
```

```
 8   Manual_Label                38 non-null     float64
dtypes: bool(1), float64(3), int64(1), object(4)
memory usage: 2.8+ KB
```

Thus, by using human labeling with specific keywords, we are able to create a list of strong labels for the supervised models, and also cut down the applicant pool further. Additionally, our final selection is balanced through both automation and human intervention, giving us a greater and more controlled final selection.

0.5.11 Phase 3 & Supervised Learning

For the supervised learning, we decided to use a Support Vector Machine (SVM), one from our we designed ourselves, and another from the `sklearn.svm` library.

A Support Vector Machine (hereinafter refereed to as SVM) is a supervised machine learning algorithm commonly used for classification and/or regression tasks. An SVM operates by finding the optimal hyperplane that best separates the data points belonging to the different classes in a high-dimensional space. We decided to use an SVM, because it works really well with high-dimensional data (like the SBERT embeddings), and has good generalization skills.

0.5.12 Support Vector Machine (SVM) – SciKit Implementation

We used the `SciKit-Learn SVC` class with an RBF kernel to train a binary classifier that can distinguish between qualified and unqualified resumes. The model is trained on the features of the SBERT embeddings from the resumes, education level as a one-hot encoding. One-hot encoding is essentially a way for categorical data to be converted into binary for machine learning algorithms. Additionally, after pre-processing and scaling the dataset was split into an 80-20 train-test split.

Moreover, we decided to use an RBF kernel, because RBF kernel's are able to generate complex non-linear decision boundaries, an absolute necessity for high-dimensional data. This is because of their innate ability to map data into high-dimensional space and find boundaries or approximations that are not exactly possible in a lower-dimension. This idea is shown in figure 11 below:

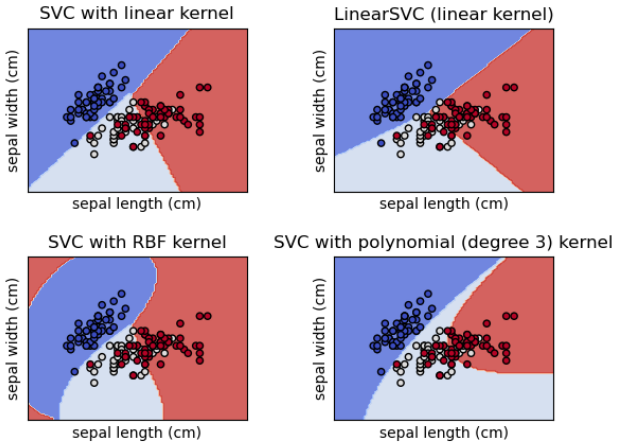

Figure 11: Support Vector Machine with differing Kernels

Next, to evaluate the model's generalization, we used 5-fold cross validation giving us a mean accuracy of 65%. This can be to the test-sets accuracy of 78% and the other performance metrics shown as follows:

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           1       0.75      1.00      0.86         6

    accuracy                           0.78         9
   macro avg       0.88      0.67      0.68         9
weighted avg       0.83      0.78      0.74         9
```

SVM from Scikit Library Confusion Matrix

Additionally, we graphed a confusion matrix for the test set, shown by figure 12.

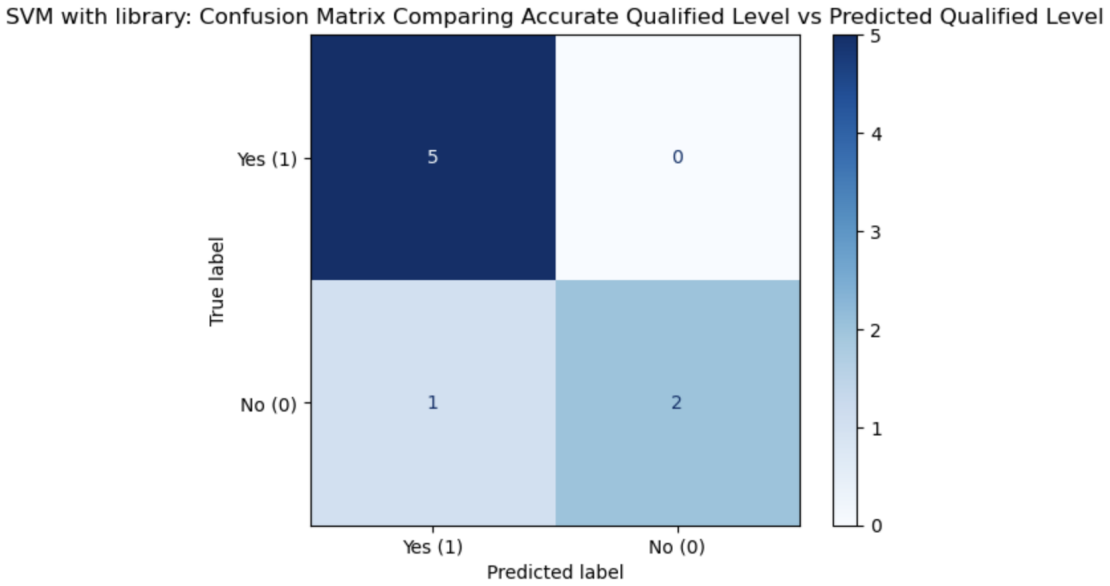

Figure 12: Confusion Matrix for Scikit SVM

When the model using Support Vector Classification (SVC) is ran, the Radial Basis Function (RBF) kernel is used to account for non-linear patterns in our data, the C parameter is assigned 0.1 penalization for misclassifications to prevent over-fitting, and class weight is balanced to prevent bias in the model. Additionally, in order to confirm accurate measures for the classification model cross validation was performed five times for an average cross validation score of 0.6556 or 65.56%. This indicates there are some varying accuracy rates when the model is ran multiple times and compared, which may indicate some overfitting.

When the model is ran for prediction, the final accuracy value is 77.78%, which means that the model accurately predicted the qualifications of 77.78% of resumes. This is neither a high or low accuracy value, but is closer to higher accuracy value ranges of 80% to 90%. However, it is still lower than the accuracy value for the SVM model made by scratch, which suggests

that that using linear regression for our model was preferred over radial basis function to give a higher accuracy score. Overfitting also could have caused some problems when testing with the entire dataset at the end. Additionally, the results of the heat map show that 5 of the 8 resumes tested in the dataset where predicted as qualified when actually qualified.

0.5.13 Support Vector Machine (SVM) – Custom Implementation

We also decided to build our own SVM model from scratch, implementing linear SVM rather than a RBF. This was done suing the NumPy libraries, and the implementation follows a process of hinge loss with regularization. Moreover, we trained the model using a gradient descent approach across 5000 iterations.

The same feature set and pre-processing as the Scikit SVM were reused for a fair comparison. For classification, we converted the target labels to {-1, 1}, which is necessary for the SVM. However, later on for consistency among the confusion matrices, the labels were set back to {0, 1}.

SVM from Scratch Algorithm

```
def fit(self, X, y):
        n_samples, n_features = X.shape          #number of samples(rows), features(columns)

        y_ = np.where(y <= 0, -1, 1)             #convert y to either -1 or 1
        self.weight = np.zeros(n_features)       #initilaize to 0
        self.bias = 0                            #initilaize to 0

        #training loop
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y_[idx] * (np.dot(x_i, self.weight) - self.bias) >= 1 #checks to current sample is correclty classified
                if condition:
                    self.weight -= self.lr * (2 * self.lambda_param * self.weight) #if correct onlt regulize
                else:
                    self.weight -= self.lr * (2 * self.lambda_param * self.weight - np.dot(x_i, y_[idx])) #if incorrected put both gradient and regulize
                    self.bias -=  self.lr * y_[idx]
```

Listing 2: Linear Support Vector Machine Data Fitting Function

Interestingly, much of an SVM is linear algebra, as seen in the code above, which will have a brief explanation: The `fit()` function trains using a gradient descent on the hinge loss function. Firstly, the labels are converted to {-1, 1}, which is important for SVM's. This is because it maintains symmetry compared to other labels such as {0, 1} or {-1, 0}. For every iteration, the algorithm checks if the current point satisfies the condition. If it does, only regularization is applied to the weight. However, if the condition is false, then the gradient is updated to penalize the models misclassifications. This process gradually updates the models weights and bias to hopefully minimize the classification error.

Our from scratch SVM model reached a test accuracy of 88.89%, thus outperforming the test accuracy of the SciKit model. Additionally, more information about the performance is listed below:

```
Accuracy: 88.89%
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.67      0.80         3
           1       0.86      1.00      0.92         6

    accuracy                           0.89         9
   macro avg       0.93      0.83      0.86         9
weighted avg       0.90      0.89      0.88         9
```

SVM from Scratch Confusion Matrix

Additionally, we graphed a confusion matrix for the test set, shown by figure 13.

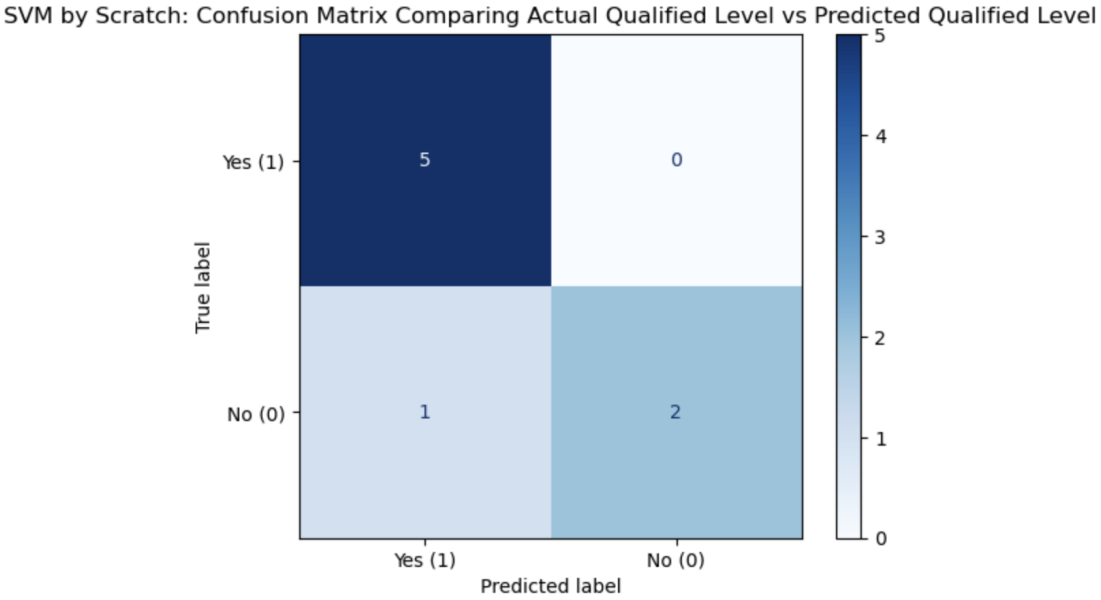

Figure 13: Confusion Matrix for Custom SVM

After making our model and manually labeling we trained our model to be able to recognize a qualified vs non-qualified depending on the job description's ideal resume. When comparing the manual vs SVM classification for the resumes we concluded that our model's training accuracy was 88.89%. Using the predicted classification from the training model and the human labeling gathered through our group's humanistic approach to classify we can note that we labeled 25 resumes as being qualified and 13 as being unqualified manually. On the contrary the model training came out with 32 resumes being qualified and 5 being unqualified. With the confusions matrix taking into account its own predictions with the 9 resumes tested we got 6 as being correctly labeled and qualified, 1 as being correctly predicted as not qualified and, 2 being incorrectly labeled as qualified by the training model.

0.5.14 SVM Conclusion

Overall, the SVM's had good performance, however, because the final dataset was so small the SVM models might have been overfitting the data, and not truly learning the patterns.

Thus, while the SVM's worked for a baseline test, for proper deployment a much larger dataset would be needed in order for the models to properly learn the patterns of a qualified resume and a not qualified resume.

Additionally, It is important to note: although the confusion matrices do look the same, with the same values, they are from the different models.

0.6 Conclusion

In this project we successfully designed and implemented an automation for the process of resume screening. By utilizing SBERT embeddings and cosine similarity we were able to filter down the 2400+ resumes that we started with to a manageable tech.csv with 42 resumes, that best matched our tesla job description. We then further validated our results by comparing two SVM models, one made from scratch and the other made from the scikit library. The model made from scratch achieved a 89% accuracy, whereas the scikit model achieved a 77% accuracy. These results show effectiveness of utilizing embeddings with supervised learning techniques like SVM in this case. Since this was using a small set of resumes we can't really know the true accuracy of the automation process, but it is good for a baseline idea. In the future if we wanted full deployment, we would need to train the models with a much larger dataset.

0.7 Contributions

Gokul Giridharan (20%) - Helped on Phase 2, Made the Project Presentation, Worked on the project Outline, Wrote the Conclusion
Henry Yost (20%) - Did the project report, Did Phase #1, Phase #2, Worked on Phase #3
Refugio Zepeda Jr. (20%) - Worked on the EDA graphs and analysis, Made the SVM model that was build from scratch and did the analysis
Riya Ashok (20%) - Made the SVM model that was made from the scikit library, and did the analysis
Angelina Jordan (20%) - Thought of the project idea, Checked for errors on the slides, Worked on the EDA analysis on the slides, Made sure everything was clear and concise.