Resume Parsing Model

Ganesh Krishnan, Rodrigo Burberg

AAI-541-01-SU23 - Capstone Project

Abstract

The paper presents a LLM Embedded Model utilizing Hybrid Querying to improve resumes using Large Language Models (LLMs), combining LangChain and ChatGPT. The methodology includes data extraction, chunking, semantic vectorization, hybrid querying, and prompt engineering. High-quality data from the "Ultimate Tech Resume Guide" is used, and documents are split for efficient processing. The model retrieves information and generates responses using a hybrid approach, combining retrieval-based and generative techniques. Prompt engineering is employed for task definition and refinement. Proof of concept classification models predict job titles from a dataset of 155 resumes, showing promising results, these models would subsequently be used to improve the LLM model. Ethical concerns regarding data use and privacy are acknowledged. The approach offers insights into AI-driven resume enhancement and aligns with recent research trends.

Problem Statement

Job seeking is a full time job. Over the course of their job hunt, the typical applicant sends out 100 to 200 applications, occasionally even 400 (Resume Worded Editorial Team, 2023). A lot of questions are posed in a job seekers quest to find a full-time job, specifically:

- How closely does the resume fit the job description?
- Is the candidate qualified for the job at hand?
- How can the candidate avoid getting screened out from the filters set by HR?
- How can the candidate land an interview?

The issue at hand is how can we help modern day job seekers to see if they are a good fit for a position and how can we help them improve their resume so that they can stand out from the crowd.

We are tackling this problem by utilizing LLMs (Large Language Models) to help give suggestions to improve their resume. Furthermore, proof of concept classification models are used to identify the career by resume, to further improve the LLM model.

LLM Embedded Model - Methodology

We have developed a model that combines the power of the Large Language Model (LLM), LangChain, and ChatGPT to critically evaluate and improve resumes based on best practices and align applicants' skills with job descriptions. The methodology employed in creating this model, including data extraction, semantic vectorization, use of chroma vector DB, Hybrid Querying, and prompt engineering, holds substantial promise. By focusing on curating specific, relevant training data from an industry-standard guide - a "textbook-like" approach inspired by Mukherjee et al.'s "Textbooks Are All You Need" research - we have equipped our model to offer valuable, precise, and effective feedback on resume evaluation and optimization tasks.

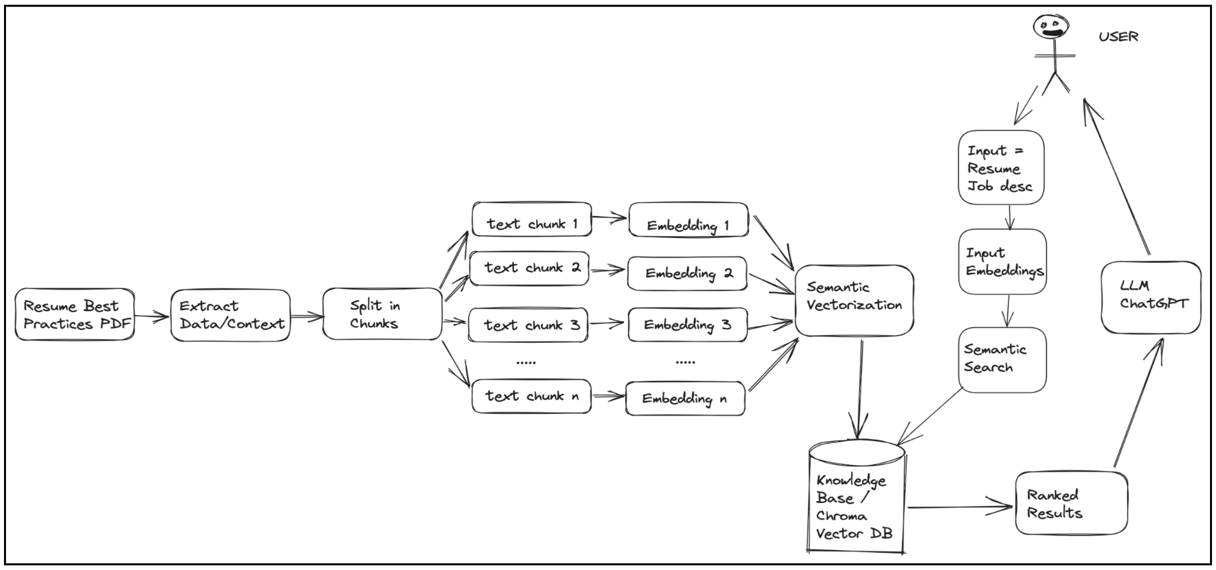


Figure 1

Resume Best Practices Data Ingestion:

To enable our model to provide effective resume critiques and optimizations, it was necessary to equip the model with expertise in resume best practices. To achieve this, we took inspiration from the groundbreaking paper, "Textbooks Are All You Need" by Mukherjee et al., which highlighted the dramatic effect of high-quality, textbook-like data on enhancing model performance while reducing the need for extensive compute and data resources.

Our data ingestion strategy was based on a hypothesis drawn from the Mukherjee et al. study. It suggested that just as a human learner might struggle to acquire skills from noisy, ambiguous, or incomplete data, language models might also face performance issues due to these problems. These issues could potentially reduce the quality and quantity of the signal that maps natural language. Therefore, we conjectured that a language model would benefit from a training set mirroring the qualities of a good "textbook": clarity, self-contained nature, instructiveness, and balance.

Our data source, the "Ultimate Tech Resume Guide", served this purpose effectively. This extensive guide, with its many examples and scenarios, embodies textbook-like quality. It offers clear, self-contained, instructive, and balanced information on resume creation. By ingesting this guide, our model was able to learn critical principles for resume development, enabling it to offer tailored feedback and improvements rooted in industry standards.This quality-over-quantity approach to data selection aligns with recent explorations into improving machine learning outcomes by focusing on data quality. Such improvements can alter the shape of scaling laws, potentially enabling the performance of large-scale models with leaner training/models. Notably, this approach can dramatically reduce the cost of large language models (LLMs) by requiring less training and smaller models. The successful ingestion of the "Ultimate Tech Resume Guide" reflects the potential for quality data to revolutionize the field of natural language processing, empowering models to reach new heights in performance.

Data Processing and Document Chunking:

To process the guide, we used a two-step approach involving data extraction and document chunking. The data extraction stage involved taking the content of the guide and preparing it for processing. This was achieved with the help of the LangChain library, a tool that excels in text processing, which converted the text into a machine-readable format, allowing the model to understand the content of the guide. Given the practical constraints of language models like GPT-3 and GPT-4, including a maximum input length, we had to ensure the guide's content was appropriately broken down for model processing. For GPT-3, for instance, the limit is 2048 tokens, with tokens ranging in size from one character to one word (in languages like English).

To conform to this limit, if a document, such as our resume guide, has more tokens than the model's limit, it must be split into smaller parts that fit the model's input length constraint. Failure to do this would either prevent the document's processing or necessitate part omission, potentially leading to the loss of crucial information.Therefore, document chunking was employed using a tool called RecursiveCharacterTextSplitter. This tool breaks the text into chunks of a specified size - in our case, 1000 characters. This size ensures that each chunk can be processed by the model without any information loss. The tool also includes an overlap of a certain number of characters between chunks to prevent loss of information at the boundaries of the chunks, preventing instances where a sentence gets cut in half, which could complicate the model's comprehension of the meaning.

Beyond enabling the model to process the entire document, splitting the document into chunks also enhances the efficiency and speed of training or inference. Smaller documents are processed faster, and by parallelizing the processing of multiple chunks, the model makes optimal use of computational resources. This combination of data extraction and document chunking plays a critical role in equipping our model with a comprehensive understanding of resume best practices and ensuring efficient operation.

Semantic Vectorization, Persistence, Embeddings and Chroma Vector DB:

Once the data from the guide had been adequately chunked and processed, the model moved onto the critical stage of semantic vectorization. In the broader realm of machine learning and natural language processing, this process, coupled with persistence and embeddings, plays a significant role, especially when handling large scale data or complex models.

Semantic vectorization transforms extracted text into meaningful numerical vectors, enabling the model to comprehend and establish relationships between different data points. This understanding forms the basis of the model's ability to provide optimal improvement suggestions. In the typical machine learning workflow, after converting documents into vector representations, or embeddings, these embeddings' state would usually be saved or persisted due to the computational cost of this operation. However, given the current scope of our project, with relatively small documents, we did not opt for persistence in our initial model design.

The Chroma Vector Database, a specialized database designed for the efficient storage and retrieval of high-dimensional vector data, was used for storing and managing the embeddings created. In scenarios where larger data volumes or more complex computations are involved, the persistent storage of this vector database would typically enable quicker reload into memory, even if the original session is terminated.

However, as our model evolves and scales up, embracing larger document sizes or preparing for deployment, we plan to incorporate persistence into our design. This future adjustment will prevent the loss of previous computations, facilitating more efficient use of computational resources, and enabling more robust handling of potential scaling. Once the vector database is persisted, it would then be used to initialize a document retriever, which is instrumental in tasks like information retrieval and question answering. A document retriever's ability to quickly locate documents semantically related to a given query makes it an indispensable tool in the realm of Natural Language Processing.

Embedding vs. Fine-tuning:

Two primary methodologies exist for leveraging a Language Learning Model (LLM), to process and comprehend document data. These methodologies are embedding the LLM with a document for information querying, and fine-tuning the LLM with the document data. Each technique provides unique advantages, accompanied by certain trade-offs, and the choice between these largely depends on the task at hand, resources available, and the nature of the data.

Embedding an LLM with a document refers to representing a document or a set of documents as high-dimensional vectors using the LLM. This method relies on 'transfer learning', utilizing pre-trained models and applying them to a new task without further training. It's an efficient, flexible approach that's quick to implement and adapt to different documents. However, the trade-off is a potentially limited comprehension of the text's specific nuances and only retrieving existing documents instead of generating new responses based on the learned knowledge.
On the other hand, fine-tuning an LLM involves further training of a pre-trained LLM on a specific task or dataset. Fine-tuning provides a more nuanced understanding of data and enables the generation of new, context-specific responses. However, it is more resource and

time-intensive, with a risk of overfitting the model to the fine-tuning data and less flexibility if the target documents change. Our decision to implement an embedding approach with our Language Learning Model (LLM) was driven by several factors, primarily concerning the quality and availability of data, time constraints, and cost considerations.

Quality and Availability of Data: For our project, the crucial datasets are resumes, reports, and optimized resumes. While we had the option of synthetically generating these datasets, we faced challenges in assessing their quality due to a lack of domain expertise. In the field of resume critique and optimization, the quality of data plays a pivotal role in the effectiveness of the model. Thus, it made more sense to rely on a self-contained guide of best practices instead of synthetically generated data. Embedding allows us to leverage such guides directly and efficiently without requiring extensive additional training on potentially subpar synthetic data.

Time Constraints: Time is a critical factor in any machine learning project. Fine-tuning an LLM is often a more time-consuming process than embedding, as it involves additional training on specific datasets, which we do not have. By contrast, the embedding approach allowed us to implement a functional model in a shorter timeframe, as it does not require this additional training phase.

Cost Considerations: The financial cost of fine-tuning an LLM is often significantly higher than embedding, due to the computational resources required for additional training. Embedding, on the other hand, offers a more cost-efficient approach, as it allows for effective utilization of pre-trained models without incurring the expenses associated with further training.

Hybrid Querying:

Our model adopts a mechanism called hybrid querying to tackle the challenges inherent in parsing and understanding large documents, thus making it proficient at extracting pertinent information and providing insightful responses. Hybrid querying is a strategy that merges the virtues of both retrieval-based and generative techniques. This approach involves embedding documents into a Vector Database using the Chroma class and deploying a pre-trained Language Learning Model (LLM), in this case ChatGPT. This dual methodology was chosen to exploit the complementary benefits these methods offer.

Document Retrieval via Vector Embeddings: With this approach, we utilize Chroma to create and store high-dimensional vector representations for our documents. This embedding process converts the documents into an optimized format for swift and precise retrieval based on semantic similarity. This retrieval-based method is exceptionally effective when dealing with vast quantities of data, significantly reducing search times and enhancing the overall efficiency.

Response Generation via LLM: On the generative side, we harness the prowess of a state-of-the-art Language Learning Model (LLM), specifically ChatGPT, to generate contextually suitable responses. This LLM is versatile, adapting its responses to fit the specific prompts it receives. This generative aspect ensures the model generates insightful, context-aware responses, surpassing mere similarity-based document retrieval.

Together, these two processes create a synergy within our model, enabling it to provide tailored and contextually-aware responses. For a given query or prompt, the model uses document embeddings to retrieve the most semantically relevant information. It then taps into the LLM's power to generate a response customized to the particular context, merging efficiency, versatility, and context sensitivity.

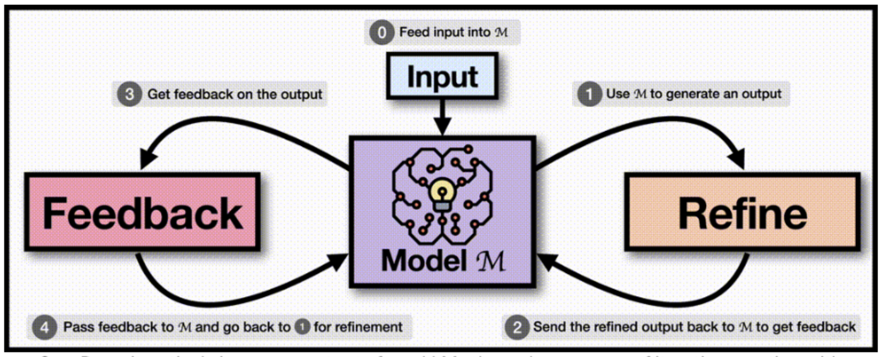
Prompt engineering: Task Definition and Self-Refinement

Prompt engineering lies at the core of our model's performance, aiding in precise task definition and shaping the model's response. Our strategy leverages two levels of prompt engineering: initial task definition and subsequent refinement.

The initial task revolves around analyzing and optimizing resumes. The model is prompted to act as a resume expert, interpreting given resumes, job descriptions, and special circumstances to provide a comprehensive report. The goal is to generate an optimized resume based on the report's recommendations. The prompt is structured with explicit instructions to ensure that the model addresses all essential points such as experience level, optimal job fit, recommendations for improvement, and special circumstances.This initial prompt, implemented through the PromptTemplate, sets the stage for the model's performance. The placeholders ({resume}, {job_description}, and {special}) are filled with respective inputs, and the resume_qa model generates an initial output.

The second level of our strategy involves refining the initial output. Here, we introduce a peer-review concept where the model revises its peer's work. This process serves two purposes: it allows the model to self-evaluate its initial output, and it adds another layer of scrutiny to ensure that no aspect of the resume review and optimization task has been overlooked. The refinement prompt is designed to add rigor to the model's reasoning process. It adds constraints such as the current date to prevent the model from suggesting career gaps and instructs the model to output the resume as a Python dictionary, reinforcing the model's ability to generate structured outputs.

Our iterative strategy of prompt engineering proves to be effective and aligns well with the 'Self-Refine' method outlined in the paper "SELF-REFINE: Iterative Refinement with Self-Feedback". This method allows large language models like GPT-4 to improve their outputs at test-time using a simple, standalone approach. By revisiting and refining initial outputs, models can learn from their mistakes, make adjustments, and significantly improve their performance. In fact, the paper reports that this method led to approximately 20% absolute improvement on average in task performance across all evaluated tasks, showcasing the value of this iterative refinement approach.


0 Feed input into $\mathcal{M}$
Input
3 Get feedback on the output
1 Use $\mathcal{M}$ to generate an output
Feedback
Model $\mathcal{M}$
Refine
4 Pass feedback to $\mathcal{M}$ and go back to 1 for refinement
2 Send the refined output back to $\mathcal{M}$ to get feedback

SELF-REFINE iteratively improves outputs from LLMs through a process of iterative creation with feedback description.

By asking the model to revisit its initial output and make improvements based on further instruction, we stimulate a form of internal dialogue within the model. This process allows for progressively refined output, increased understanding of the task at hand, and improvement over time. The strategy is also robust against potential omissions in the first instance, given the second layer of scrutiny and the opportunity to incorporate feedback directly into the model's iterative learning process. Our two-level approach to prompt engineering exemplifies the power of thoughtfully crafted instructions in leveraging the capabilities of large language models, echoing the value of self-feedback mechanisms for enhanced model performance.

LLM Embedded Model - Results:

The evaluation of Large Language Models (LLMs), particularly when applied to complex and ***subjective*** tasks like resume optimization, necessitates a multifaceted approach that combines automatic metrics, human evaluations, iterative refinement strategies, and possibly tracking real-world outcomes.

In our opinion, Standard metrics for evaluating LLMs, such as Perplexity, BLEU (Bilingual Evaluation Understudy), ROUGE (Recall-Oriented Understudy for Gisting Evaluation), and METEOR (Metric for Evaluation of Translation with Explicit ORdering), are not suitable for this use case. Other non-suitable metrics are perplexity, which measures a model's uncertainty about the next token in a sequence, which isn't as relevant for a task that requires understanding, reasoning, and giving context-specific advice. Likewise, metrics like BLEU, ROUGE, and METEOR, which evaluate the overlap between a model's output and a human reference, are less relevant for this task because the goal isn't to generate text that closely matches a given reference, but to provide personalized and context-aware recommendations.

Recognizing the limitations of traditional metrics, we propose an evaluation strategy that includes human evaluations, tracking of real-world outcomes, and task-specific custom metrics. The model's outputs can be assessed based on relevance, context-awareness, and the practicality of the recommendations it provides. Real-world outcomes such as the number of interviews or job offers a candidate receives after using the optimized resume could be valuable indicators of the model's effectiveness; we could assess this via an A/B test. Additionally, task-specific custom metrics like "Pieces of Information Missed per 100 Resumes" or "Unaddressed Points in Special Circumstances" or "Hallucinations per 100 Resumes” could provide insights into the model's thoroughness and attention to detail.

Each model output should be meticulously evaluated by humans. Any corrections, comments, or identified mistakes should not only contribute to performance metrics but also guide model improvements. For instance, if the model overlooked addressing a career break in its initial output, incorporating iterative refinement techniques like SELF-REFINE could allow the model to identify and appropriately address such significant details in its revised outputs.

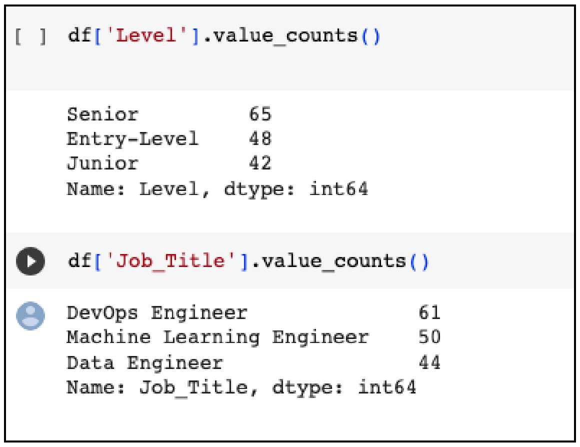
For scenarios like career-switching, the model's ability to discern this from the input and provide relevant recommendations can be an important performance metric. These insights can guide the creation of synthetic data or specific prompt chains, which can help the model better tackle these scenarios. Additionally, it is important to ensure the model doesn't hallucinate, or generate information not present in the input. In a task like resume optimization, hallucinations could result in incorrect or misleading advice, which could adversely affect a job seeker's prospects. In our model we introduced prompt engineering techniques to recognize any hallucinations present via the self-refine technique.

The model can also be fine-tuned with quality data, such as textbook-like data or outputs from a multiclass classification model. This can help it better understand the task and the expected output format. For example, if the model struggles with suggesting suitable job roles, it can be fine-tuned with data from a multiclass classification model, generating a list of the top ten jobs a candidate might be suitable for. Outputs from other models, specifically trained to perform tasks related to the main task at hand, can be used as a resource for fine-tuning. For instance, a multiclass classification model, trained to classify resumes into different job roles based on skills, experiences, and qualifications, can be a valuable source of information. The outputs from such a model can provide nuanced insights into the correlation between a candidate's profile and potential job fits.

Classification Models - Methodology:

For the classification models a customized dataset of 155 resumes. The resume collection process consisted of two phases. Initially, public sources such as LinkedIn were leveraged to obtain authentic resumes concentrated within three career tracks: Data Engineering, DevOps Engineering, and Machine Learning Engineering. Furthermore, resumes were gathered across experience levels including Entry-Level, Junior, and Senior roles. During LinkedIn data extraction, ethical considerations regarding candidate consent were addressed by anonymizing all personal information and company affiliations. The total number of LinkedIn-sourced resumes compiled was 75.

To augment the dataset, synthetic resume generation was subsequently pursued. Gretel facilitated the creation of approximately 80 synthetic resumes each for Data Engineering, DevOps Engineering, and Machine Learning Engineering roles. Between authentic and synthetic documents, the total customized resume dataset exceeded 155 samples. Before feeding into the models preprocessing was performed which consisted of: tokenization, removing stop words, stemming, removing punctuation and special characters. As seen in Figure 2 below, the dataset is more or less balanced.

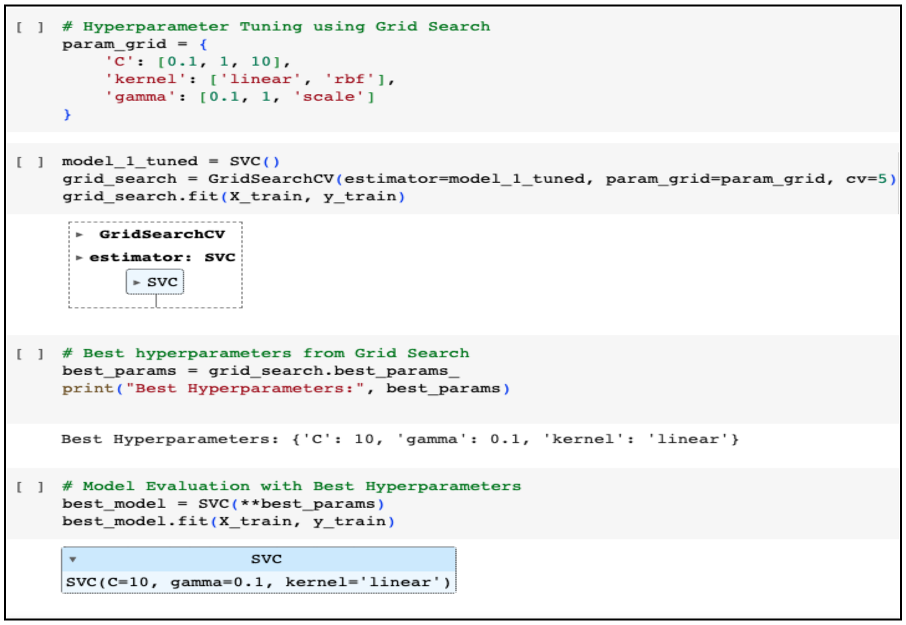


```
[ ]  df['Level'].value_counts()

     Senior         65
     Entry-Level    48
     Junior         42
     Name: Level, dtype: int64

     df['Job_Title'].value_counts()

     DevOps Engineer              61
     Machine Learning Engineer    50
     Data Engineer                44
     Name: Job_Title, dtype: int64
```

Figure 2 (Google Colab)

Support Vector Classifier

Grid Search was used to tune the hyperparameters of the SVC model. The hyperparameters that we are tuning are C, kernel, and gamma. Based on the hyperparameter tuning, it chose a value of 10 for C (regularization parameter) and gamma of 0.1. A high C value indicates that we have a smaller value hyperplane and a model with low bias but potentially high variance. A larger C value penalizes misclassified points in the model and could overfit. A low gamma value makes the model less sensitive to individual training examples and could result in underfitting (Yıldırım, 2021).



```
[ ] # Hyperparameter Tuning using Grid Search
    param_grid = {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf'],
        'gamma': [0.1, 1, 'scale']
    }

[ ] model_1_tuned = SVC()
    grid_search = GridSearchCV(estimator=model_1_tuned, param_grid=param_grid, cv=5)
    grid_search.fit(X_train, y_train)

      ▸ GridSearchCV
     ▸ estimator: SVC
          ▸ SVC

[ ] # Best hyperparameters from Grid Search
    best_params = grid_search.best_params_
    print("Best Hyperparameters:", best_params)

    Best Hyperparameters: {'C': 10, 'gamma': 0.1, 'kernel': 'linear'}

[ ] # Model Evaluation with Best Hyperparameters
    best_model = SVC(**best_params)
    best_model.fit(X_train, y_train)

    ▾                SVC
    SVC(C=10, gamma=0.1, kernel='linear')
```

Figure 3 (Google Colab)

Having a dataset of 155 resumes gave mixed results, however it should be noted that hyperparameter tuning benefits from larger datasets. Hyperparameter tuning didn’t yield any significant improvement in results.

Multinomial Naive Bayes Classifier

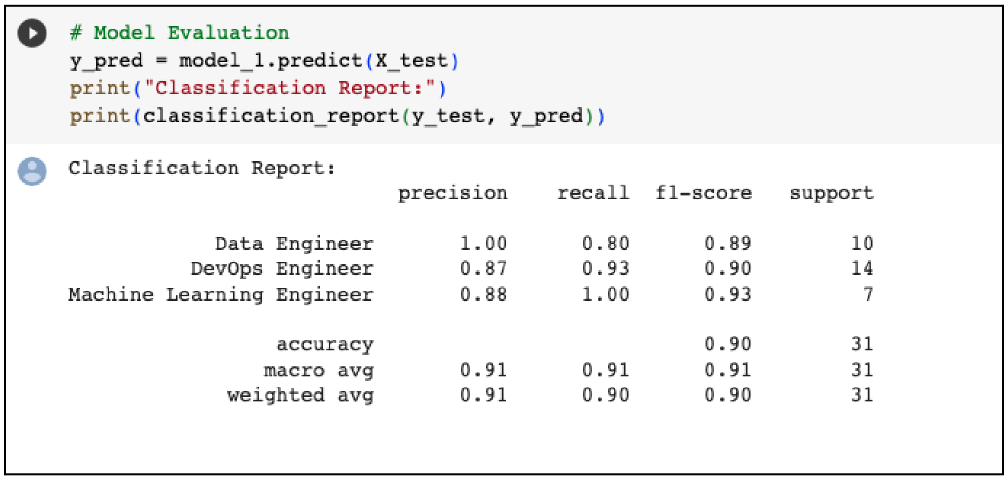
After fitting the model, its performance is evaluated by predicting labels for the test data and generating a classification report. Subsequently, hyperparameter tuning is undertaken using grid search and cross-validation. Grid search is a technique to identify optimal hyperparameter values. Specifically, the alpha parameter controlling smoothing in the model is being adjusted. The param_grid dictionary contains potential alpha values, while the cv argument specifies the number of cross-validation folds.

During cross-validation, the GridSearchCV class systematically explores different hyperparameters across folds. The model's performance on the cross-validation set aids in selecting the best hyperparameters. In the case of the Naive Bayes algorithm, alpha is a crucial hyperparameter. Low alpha values lead to high variance (overfitting), whereas large values result in high bias (underfitting) (Cheruku, n.d.). In this study, an alpha value of 1 strikes a balance between underfitting and overfitting. Despite these efforts, hyperparameter tuning did not significantly impact accuracy scores.

Classification Models - Results:

The results were impressive from both classification models. Support Vector Classifier had a weighted average of 90% precision, recall, and f1-scores when predicting the correct job title based on a given resume. The Multinomial Naive Bayes Classifier achieved an impressive weighted average of 94% precision, recall, and f1-scores without hyperparameter tuning. Keep in mind that only 155 resumes were used to run the classification models. The classification models at hand were proof of concept showing for future development, where we would create a model that outputs multiclass classification on at least ten jobs. These results would then be fed into the LLM model for fine-tuning purposes.

Support Vector Classifier

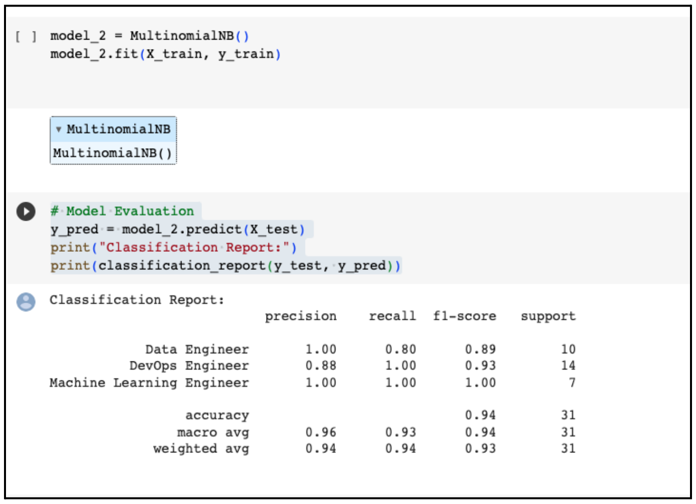


```
# Model Evaluation
y_pred = model_1.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

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

            Data Engineer       1.00      0.80      0.89        10
          DevOps Engineer       0.87      0.93      0.90        14
Machine Learning Engineer       0.88      1.00      0.93         7

                 accuracy                           0.90        31
                macro avg       0.91      0.91      0.91        31
             weighted avg       0.91      0.90      0.90        31
```

Figure 4 (Google Colab)

Multinomial Naive Bayes Classifier



```
[ ] model_2 = MultinomialNB()
    model_2.fit(X_train, y_train)
```

```
# Model Evaluation
y_pred = model_2.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

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

           Data Engineer       1.00      0.80      0.89        10
         DevOps Engineer       0.88      1.00      0.93        14
Machine Learning Engineer       1.00      1.00      1.00         7

                accuracy                           0.94        31
               macro avg       0.96      0.93      0.94        31
            weighted avg       0.94      0.94      0.93        31
```

Figure 5 (Google Colab)

Ethics

The use of existing human-created works such as resumes and textbooks to train AI systems raises multifaceted ethical concerns that require nuanced analysis:

The use of copyrighted materials such as textbooks to train AI systems gives rise to complex ethical questions surrounding consent, fair use, and attribution. On one hand, utilizing such texts without permission from rights holders could be construed as an inappropriate appropriation of others' intellectual property and effort. However, most academic and technical publications build upon a foundation of communal knowledge. The boundary between standing on the shoulders of giants and exploiting others' work is not always well-defined. Arguments exist that training AI on copyrighted content potentially infringes author rights, contingent on factors including the system's purpose, outputs, and monetization model. Verbatim reproduction may constitute infringement, while transformative applications could qualify as fair use depending on the specific context. Comprehensive guidelines remain absent given the lack of legal precedents on these issues.

Irrespective of permissions, it is advisable to acknowledge any training sources and exercise transparency regarding utilized data. This upholds ethical norms of integrity and enables accountability. Academia has a strong tradition of citing inspirations and reference materials. AI research would benefit by emulating such practices, especially when derivative works are involved. While building upon existing knowledge is foundational to human progress, due diligence must be taken to avoid undue exploitation.

The practice of harvesting resumes from sources like LinkedIn to train AI systems also warrants prudent ethical reflection. Utilizing resumes without consent could constitute an inappropriate appropriation of individuals' personal information and career histories. However, resumes are intended to showcase professional capabilities and credentials to potential employers. The boundary between public presentation and private data is ambiguous. Arguments exist that leveraging resumes to develop AI infringes on candidates' rights, contingent on how the training data is obtained, anonymized, and applied. Directly reproducing resume details in AI screening tools is ethically concerning, but transforming resume data into assistants that provide constructive feedback enables helpful innovations, assuming they are designed responsibly and with candidate rights as the priority.

It is advisable to anonymize any harvested resumes and exercise transparency regarding data provenance. This upholds norms of privacy and enables accountability. Data minimization and representativeness are also crucial to avoid perpetuating historical biases in hiring practices. Like academic research, AI development would benefit by emulating ethical data practices, especially when personal information is implicated.

Future work:

Techniques to improve the models performance and capabilities include:

1. Incorporating fine-tuned data from multiclass classification models can help the LLM better understand the task and expected output format. This approach can improve its ability to suggest suitable job roles and offer nuanced insights into the correlation between a candidate's profile and potential job fits.
2. Creating a feedback loop with human reviewers can continuously improve the model's responses. Integrating reviewer feedback into the training process can enhance the model's understanding and performance.
3. Continuing to apply iterative refinement techniques, similar to SELF-REFINE, can lead to improved outputs over time. By revisiting and refining initial outputs, the model can learn from its mistakes and make adjustments, enhancing its performance in providing tailored advice.
4. Expanding beyond tech-related resumes and incorporating textbook-like data from various fields. By utilizing domain-specific guides, we aim to provide tailored advice for resumes across different industries, addressing the needs of a broader range of job seekers.

Conclusion:

In this study, we introduced an LLM Embedded Model that utilizes Hybrid Querying to enhance resumes using Large Language Models (LLMs). We combined LangChain and ChatGPT to process and evaluate resumes based on best practices and align candidates' skills with job descriptions. Our methodology involved data extraction, semantic vectorization, hybrid querying, and prompt engineering. By curating high-quality data from the "Ultimate Tech Resume Guide" and using a hybrid approach, our model retrieved information and generated responses efficiently. The prompt engineering strategy aided in task definition and self-refinement.

We addressed the problem of helping job seekers improve their resumes and stand out from the crowd. Our model's performance was promising, and we developed proof of concept classification models to predict job titles from resumes. These models laid the foundation for improving the LLM model.

References

1. Cheruku, S. K. (n.d.). Naive Bayes classifier model in machine learning. *www.linkedin.com*. https://www.linkedin.com/pulse/naive-bayes-classifier-model-machine-learning-sunil-kumar-cheruku/
2. Gunasekar, S., Zhang, Y., Aneja, J., Mendes, C. C. T., Del Giorno, A., Gopi, S., Javaheripi, M., Kauffmann, P., de Rosa, G., Saarikivi, O., Salim, A., Shah, S., Behl, H. S., Wang, X., Bubeck, S., Eldan, R., Kalai, A. T., Lee, Y. T., & Li, Y. (2023, June 20). *Textbooks are all you need*. arXiv.org. https://arxiv.org/abs/2306.11644
3. Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Welleck, S., Majumder, B., Gupta, S., Yazdanbakhsh, A., & Clark, P. (n.d.). *SELF-REFINE: ITERATIVE REFINEMENT WITH SELF-FEEDBACK*. https://arxiv.org/pdf/2303.17651.pdf
4. Resume Worded Editorial Team. (2023). How Many Jobs Should You Apply For in 2023? A Breakdown. *Resume Worded*. https://resumeworded.com/how-many-jobs-should-i-apply-for-key-advice#:~:text=The%20average%20job%20seeker%20sends,company%20before%20you're%20successful.
5. Ratz, A. V. (2022, April 11). Multinomial Naïve Bayes' For Documents Classification and Natural Language Processing (NLP). *Medium*. https://towardsdatascience.com/multinomial-na%C3%AFve-bayes-for-documents-classification-and-natural-language-processing-nlp-e08cc848ce6
6. Sriram. (n.d.). Top 12 Commerce Project Topics & Ideas in 2023 [For Freshers]. *upGrad blog*. https://www.upgrad.com/blog/multinomial-naive-bayes-explained/

7. *Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi Quoc, E., Le, V., & Zhou, D. (n.d.). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models Chain-of-Thought Prompting.* *https://arxiv.org/pdf/2201.11903.pdf*
8. Yıldırım, S. (2021, December 14). Hyperparameter tuning for support vector machines — C and gamma parameters. *Medium*. https://towardsdatascience.com/hyperparameter-tuning-for-support-vector-machines-c-and-gamma-parameters-6a5097416167
9. *1.4. Support vector machines*. (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/svm.html

Appendix

1.0 Code: **https://github.com/burberg92/Resume_LLM**