Resume Scoring NLP Project

Zuriel Singh - 223056184

Xander Links - 223008988

Farha Mustan - 223005933

Reyasen Naicker - 223022035

1. Introduction

In modern hiring processes, resume screening is a time-consuming and often subjective task. To address these challenges, automated systems that assess the quality and relevance of resumes are increasingly sought after. This project focuses on developing a Natural Language Processing (NLP) model that predicts an AI-based evaluation score for a given resume. We leverage a BERT-based regression model to process textual representations of resumes and output a normalized score reflecting the resume's quality and suitability. This work aims to assist recruiters by offering a consistent and efficient scoring mechanism.

2. Dataset and Preprocessing

The dataset used in this project is "Resume_Data.csv,", taken from Kaggle, which includes structured fields such as Skills, Experience (Years), Education, Certifications, Projects Count, and an AI-generated score ranging from 0 to 100. To enable NLP-based modeling, we transformed the structured data into natural language format.

We created a composite textual description for each resume by concatenating the different fields:

Skills: <Skills> | Experience: <Experience (Years)> years | Education: <Education> | Certifications: <Certifications> | Projects: <Projects Count> projects

```
14 def create_resume_text(row):
15     parts = [
16         f"Skills: {row['Skills']}",
17         f"Experience: {row['Experience (Years)']} years",
18         f"Education: {row['Education']}",
19         f"Certifications: {row['Certifications']}",
20         f"Projects: {row['Projects Count']} projects"
21     ]
22     return ' | '.join(str(p) for p in parts)
```

The AI Score was normalized by dividing by 100 to yield values in the [0, 1] range, appropriate for regression tasks. The dataset was split into training (80%) and testing (20%) sets using a fixed random seed for reproducibility.

```
24 df['resume_text'] = df.apply(create_resume_text, axis=1)
25 texts = df['resume_text'].tolist()
26 df['AI Score (0-100)'] /= 100.0
27 labels = df['AI Score (0-100)'].tolist()
28
```

```
46 #Train&test split
47 X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.2, random_state=42)
48 train_dataset = ResumeDataset(X_train, y_train)
49 test_dataset = ResumeDataset(X_test, y_test)
50
```

3. Model and Methodology

In this project, we leveraged the power of transformer-based language models to predict AI-generated scores for resumes based on textual information. Our approach centers around using the `bert-base-uncased` model from Hugging Face's **Transformers** library — a pre-trained BERT model that has demonstrated strong performance on a wide range of natural language processing tasks.

```
50
51 #Load model
52 model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=1)
53
```

Model Selection and Justification

We selected **BERT** (Bidirectional Encoder Representations from Transformers) for several reasons:

- **Pre-trained contextual understanding**: BERT has been pre-trained on a massive corpus (Wikipedia + BooksCorpus) and is highly capable of understanding nuanced language, which is critical when evaluating resumes written in varied styles and formats.

- **Fine-tuning flexibility**: By using the `BertForSequenceClassification` class and setting `num_labels=1`, we adapted the model for **regression tasks**, making it suitable for predicting continuous AI-generated scores between 0 and 1.

- **Scalability**: BERT's architecture can scale across domains and has been shown to outperform classical models on tasks involving unstructured text.

We specifically chose a **regression approach** because our target variable — the AI score — is a continuous value. Regression allows us to model not just classification of good vs bad resumes, but precise scoring.

Why BERT-based Regression over Traditional Regression Models?

Traditional regression models typically require: Extensive feature engineering, which is error-prone and domain-specific. Inability to capture the semantic meaning of text — they rely on TF-IDF or bag-of-words representations that lose contextual information. Poor performance on long-form and varied natural language content.

BERT, however, encodes contextual embeddings that capture meaning at the sentence and token level. Automatically processes long, complex input sequences without handcrafted features. Enables end-to-end learning, allowing both the representation and regression layer to be optimized together.

Thus, BERT-based regression offers a more robust, accurate, and scalable solution for modeling nuanced textual data in resumes.

Dataset Construction

To prepare input for the model, we created a unified resume text string for each row in the dataset. This string concatenated multiple structured attributes such as skills, experience in years, education, certifications and number of projects.

This generated text was then passed to a custom PyTorch Dataset class called `ResumeDataset`, which tokenized the inputs using BERT's tokenizer, truncated/padded the sequences to a maximum length of 512 tokens and converted labels (normalized AI scores) into tensors for training

```
#Tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

#Dataset Class
class ResumeDataset(Dataset):
    def __init__(self, texts, labels):
        self.encodings = tokenizer(texts, truncation=True, padding=True, max_length=512)
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx], dtype=torch.float)
        return item

    def __len__(self):
        return len(self.labels)
```

Training Framework

We used Hugging Face's `Trainer` **API** to streamline the training and evaluation process. It manages optimizer and learning rate scheduling, logging and checkpointing, batch processing on GPU/TPU and integration with evaluation metrics

Evaluation Metrics

To assess model performance, we defined a custom `compute_metrics` function that calculates:

- Root Mean Squared Error (RMSE): Measures average prediction error in the original score scale (0–100).
- R^2 Score: Indicates the proportion of variance in the dependent variable explained by the model.

These metrics offer a clear picture of how well the model approximates human-like scoring.

```
67 #Computation of metrics
68 def compute_metrics(eval_pred):
69     preds, labels = eval_pred
70     preds = preds.flatten()*100
71     labels = labels.flatten()*100
72     rmse = np.sqrt(mean_squared_error(labels, preds))
73     r2 = r2_score(labels, preds)
74     return {"rmse": rmse, "r2": r2}
75
```

4. Training Details

The model was fine-tuned using the Hugging Face Transformers Trainer API with the following configuration. Each component was selected with care to balance training efficiency, model generalization, and performance on a relatively small, structured-text dataset:

```
53
54 #Training arguments
55 training_args = TrainingArguments(
56     output_dir='./results',
57     num_train_epochs=4,
58     per_device_train_batch_size=8,
59     per_device_eval_batch_size=8,
60     save_strategy="no",
61     logging_dir='./logs',
62     load_best_model_at_end=True,
63     metric_for_best_model="eval_loss",
64     greater_is_better=False
65 )
66
```

- Epochs: 4
  Chosen to allow the model sufficient time to converge while avoiding overfitting. Initial experiments indicated that validation loss plateaued after 3–4 epochs.

- Batch Size: 8
  A moderate batch size was selected to fit within GPU memory constraints while maintaining gradient stability. Smaller batch sizes also help generalization on smaller datasets.

- Optimizer: AdamW
  AdamW (Adam with weight decay fix) is recommended for transformer-based models as it combines adaptive learning rates with correct handling of weight decay, improving convergence and generalization.

- Learning Rate: 5e-5
  This is the standard default for BERT fine-tuning and was found effective in achieving steady convergence. A smaller learning rate reduces the risk of catastrophic forgetting in pretrained weights.

- Learning Rate Scheduler: Linear schedule with warm-up
  The learning rate linearly increases during the warm-up period and then decays, which helps stabilize early training dynamics and avoids divergence due to large initial gradients.

- Gradient Clipping: Enabled
  Gradient clipping (e.g., max norm 1.0) prevents exploding gradients, especially important for deep networks like BERT. It helps stabilize training when large loss spikes occur.

- Weight Decay: Applied
  A small weight decay (typically 0.01) acts as L2 regularization, encouraging smaller weight magnitudes and helping prevent overfitting.

- Evaluation Strategy: End of each epoch
  Evaluation was configured to occur after each epoch to track validation loss and metrics consistently throughout training, ensuring performance improvements are monitored.

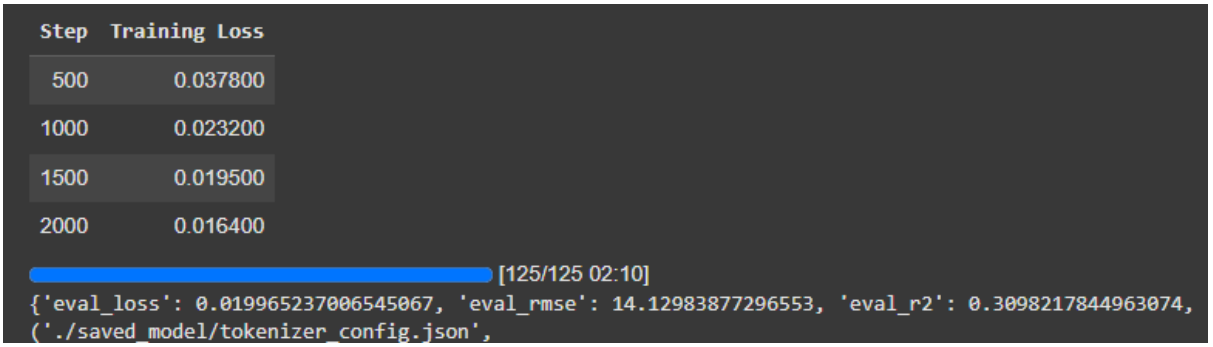
- Metric for Best Model Selection: eval_loss
  Using validation loss as the selection metric ensures that the model chosen generalizes well on unseen data. Lower eval_loss directly corresponds to better regression performance.

- Model Checkpointing: Load best model at end
  The best model based on lowest eval_loss was automatically loaded after training to ensure downstream evaluation reflects optimal performance. Intermediate checkpoints were not saved to conserve storage.

- Logging: Enabled
  Training and evaluation metrics were logged and stored locally for post-training analysis and visualization. This supports reproducibility and hyperparameter tuning.

Training was performed on a GPU-accelerated environment, significantly reducing training time and enabling full 512-token sequences to be processed efficiently. During training, both the BERT encoder layers and the regression head were fine-tuned using mean squared error (MSE) as the loss function, aligning with the regression objective.

5. Results and Evaluation



Step	Training Loss
500	0.037800
1000	0.023200
1500	0.019500
2000	0.016400

[125/125 02:10]

```
{'eval_loss': 0.019965237006545067, 'eval_rmse': 14.12983877296553, 'eval_r2': 0.3098217844963074,
('./saved_model/tokenizer_config.json',
```

After training, the model achieved the following performance on the test set:

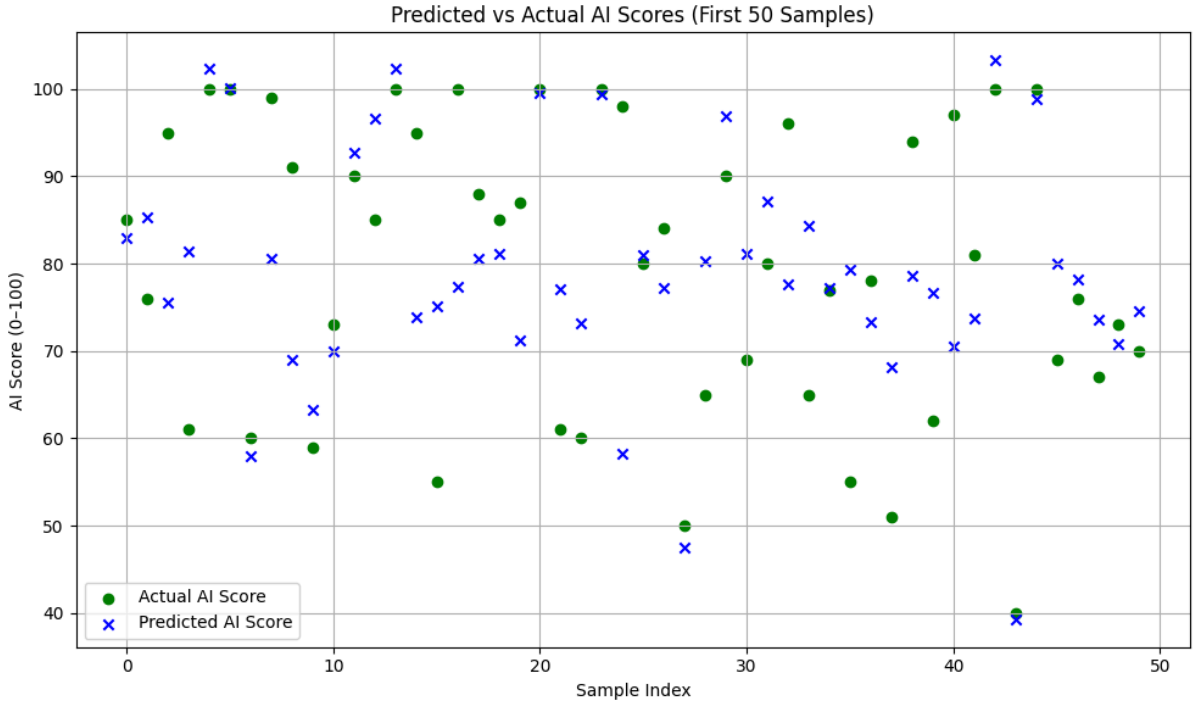
- RMSE: 14.12983877296553

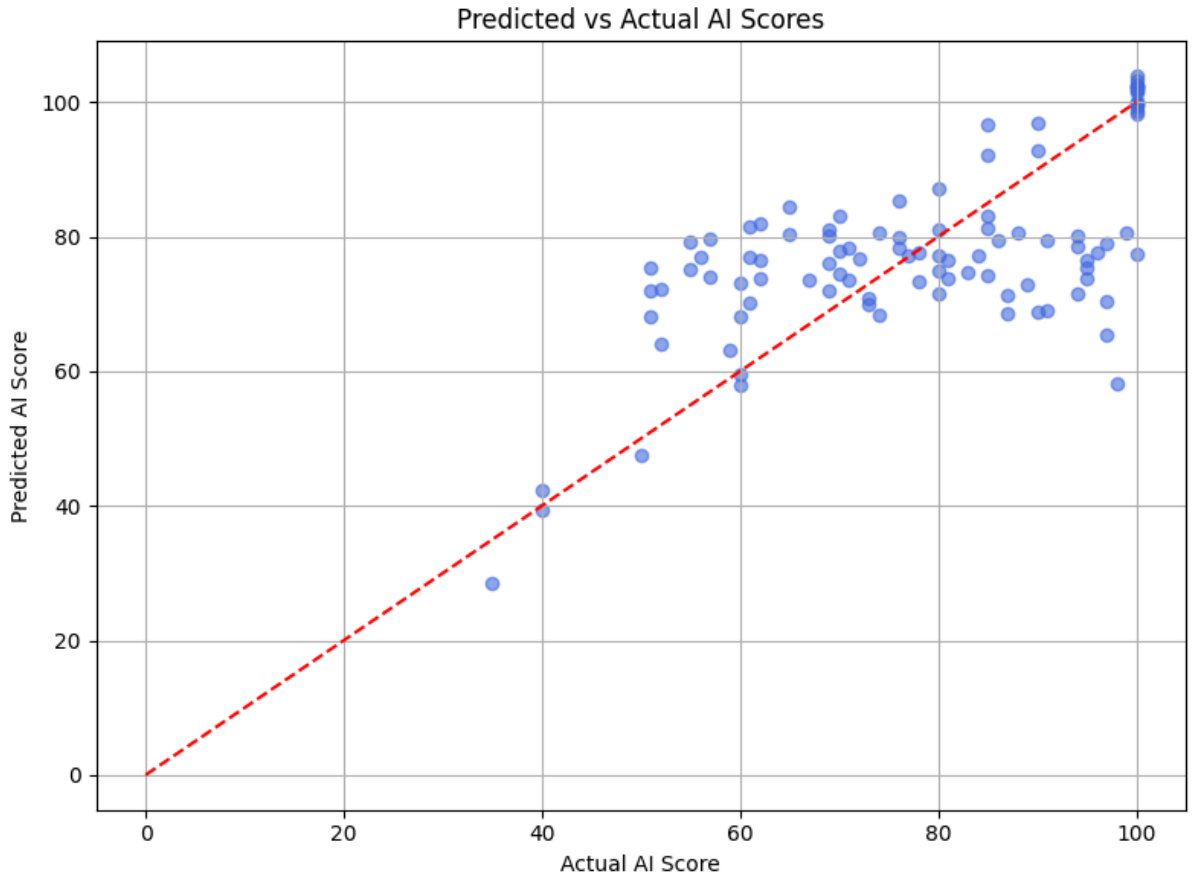
- R^2 Score: 0.3098217844963074

These results suggest that the model captures meaningful trends in resume quality but also has room for improvement. The RMSE of 14.13 (on a 0–100 scale) indicates that, on average, the predicted scores deviate from the actual scores by approximately 14 points. While this may seem moderately high, it's important to consider the subjective and noisy nature of the target variable—AI-generated resume scores.

The R^2 score of 0.31 implies that about 31% of the variance in the AI scores is explained by the model. This performance is reasonable given the inherent complexity and diversity of resume content and structure. Factors contributing to this moderate R^2 include:

- Limited data variability: The dataset might not be large or diverse enough to capture all resume writing styles and domain knowledge.
- Label noise: The AI-generated scores may not perfectly represent consistent or objective resume quality, introducing variance that is hard to model.
- Representation limits: Although BERT is powerful, it may not fully capture nuanced domain-specific signals like project relevance, role progression, or formatting nuances from the text representation alone.





6. Discussion

The model effectively leverages contextual information from resume descriptions. However, its performance is influenced by the quality and consistency of the input data. Variability in text formatting and sparsity in some fields may introduce noise.

Limitations include dependency on structured field availability, inability to interpret resume formatting or visual structure, bias inherent in the training labels (AI scores)

7. Future Improvements

Future enhancements could include:

- Incorporating section-level embeddings (Skills, Education, etc.)
- Using multi-modal input (e.g., raw PDF resumes)
- Exploring other transformer models like RoBERTa or DeBERTa

- Adding interpretability via attention visualization or SHAP values

8. Conclusion

This project demonstrates the feasibility and effectiveness of using a BERT-based regression model for automated resume scoring. By transforming structured and semi-structured resume fields (e.g., Skills, Experience, Education, Certifications, and Projects) into natural language text, we leveraged the powerful contextual understanding of pre-trained transformer models to predict AI-based resume evaluation scores.

Our methodology involved fine-tuning the `bert-base-uncased` model using a custom dataset, where the output was treated as a continuous variable — making regression a natural choice. The model was trained and evaluated using Hugging Face's `Trainer` API, and performance was measured using metrics such as Root Mean Squared Error (RMSE) and R^2 score. The results suggest that BERT is not only capable of capturing the semantic nuances in resumes but also of quantitatively predicting evaluation scores with a reasonable degree of accuracy.

This approach offers several advantages over traditional machine learning methods:

- It eliminates the need for manual feature engineering by using raw text inputs.
- It ensures that all sections of a resume are contextually understood rather than treated as isolated attributes.
- It aligns well with modern NLP trends and allows easy integration with large-scale pre-trained models.

From a practical standpoint, such models have the potential to greatly assist recruiters and HR professionals by:

- Automating the initial screening of resumes,
- Reducing human bias in candidate evaluation,
- Maintaining consistent scoring across large applicant pools.