



University of Hyderabad

**Compute the Match Percentage of a Candidate who Fits
the Job Role Based on the Job Description**

By

Sanjay Chouhan

sc.chouhansanjay@gmail.com

Enrl. No. : 40AIML512-21/1

A project submitted in fulfilment for the
Diploma Degree

February 2022

DECLARATION OF AUTHORSHIP

I, Sanjay Chouhan, declare that this project titled, 'Compute the Percentage of a Candidate who Fits the Job Role Based on the Job Description' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a diploma degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given.
- With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Sanjay Chouhan

Date: 20/02/2022

ABSTRACT

In the IT industry there are lots of job portals and lots of candidates. When a company posts for a job role they receive many applicants. Sometimes 1000x more than the actual vacancies. Which becomes a problem for the company. If we can somehow automate the filtering process to choose the most suitable candidates. Or even if we can reduce the number of the candidates by removing the non compatible candidates. It will save a lot of the workforce of the company. In this project I have worked on a machine learning based system to Compute the Match Percentage of a Candidate who fits the Job Role based on the Job Description. If we can compute the match percentage with little margin of error. Then we can select top k candidates for further processing based on the Match Percentage. Here I have explored almost all of the machine learning models for this task, from simple one like linear regression to complex sota methods like BERT. And surprisingly the simple models are performing better than complex ones.

TABLE OF CONTENTS

1	Literature Survey & Data Acquisition	1
1	Problem Definition	1
2	Data Set Description	3
3	Key Performance Indicator (KPI) Metric	4
4	Real-World Challenges and Constraints	6
5	Literature Review	7
2	EDA and Feature Extraction	9
1	Basic EDA (Exploratory Data Analysis)	9
1.1	Job Description	9
1.2	Resumes	9
1.3	Univariate and Bi-Variate Analysis	10
2	Data Cleaning	11
3	Feature Extraction	12
3.1	Univariate Analysis	13
3.2	Bi-Variate Analysis	15
4	Feature Encoding	16
4.1	Binary Bow	17
4.1.1	Univariate and Bi-Variate Analysis	18
4.2	Average Word2vec	18
4.2.1	Univariate and Bi-Variate Analysis	19
5	High-Dimensional Data Visualization	19
3	Modelling and Error Analysis	20
1	Models with Bow Features	20
1.1	Train Test Split	20
1.2	Preprocessing	21

1.3	Modeling (Basic)	21
1.4	Error Analysis	22
1.5	Train Test Split and Preprocessing	22
1.5	Modeling	23
2	Models with Average Word2vec Features	24
2.1	Train Test Split	24
2.2	Preprocessing	24
2.3	Modeling (Basic)	24
2.4	Error Analysis	25
2.5	Train Test Split and Preprocessing	25
2.6	Modeling	25
3	Performances	26
4	Stacking Ensemble (Stack of Best Models)	27
4.1	Preprocessing	27
4.2	Finding the Meta Model	27
4.3	Train and Test Performance with the Best Model	28
5	Further Analysis of the Best Model	29
6	Advantages and Disadvantages of the Models	29
6.1	Linear Regression	29
6.2	Support Vector Machine	30
6.3	K-NN Regressor	30
4	Advanced Modeling and Feature Engineering	31
1	Feed Forward Neural Network with Bow Features	31
1.1	Dataset	31
1.2	Model Architecture	31
1.3	Training and Evaluation	32
2	Feed Forward Neural Network with BoW and Average W2V	33
Features		
2.1	Data Set	33
2.2	Model Architecture	33
2.3	Training and Evaluation	34
3	BERT with Feed Forward Neural Network	35

3.1	Data Set	36
3.2	Model Architecture	37
3.3	Training and Evaluation	38
4	Conclusion	38
5	Deployment and Productionization	39
1	Requirements	39
1.1	Applications	39
1.2	Python Libraries	39
2	Web Framework	40
2.1	Flask	40
2.1.1	Homepage	40
2.1.2	Predict Page	41
2.2	Flask Code	42
2.2.1	match.Py	42
2.2.2	helper.Py	42
2.2.3	models Folder	42
2.2.4	static Folder	42
2.2.5	templates Folder	42
2.2.6	uploads Folder	43
3	Hosting	43
3.1	WSGI Server	43
3.1.1	Gunicorn	43
3.2	Amazon EC2	44
4	Optimisation	44
5	Architecture Diagram	45
6	Scalability and Latency	45
6.1	Scalability	45
6.2	Latency	46
7	Links	46
6	Bibliography	47

Chapter 1

LITERATURE SURVEY & DATA ACQUISITION

1. PROBLEM DEFINITION:-

The main problem a company faces while hiring new candidates is the selection of suitable candidates based on their resumes and the job description for the first round of the interview process. It's like finding a needle in a haystack, especially for the big companies like MAANG or even other popular MNCs.

Many job search companies like naukri.com, google jobs, LinkedIn have made finding and applying for a job very easy, which has its own problem. Nowadays a company receives thousands or even tens of thousands of resumes against a single post. Traditionally all resumes require manual review by some HR managers for further process.

This means the big companies need to have a sufficient number of HR managers for this task. Also, there are different types of job posts in a company so these managers should have proper knowledge of the job descriptions. So that they can select the suitable resumes. It may also introduce human bias because some people are very strict and some are lenient.

The selection of the suitable candidates based on the job description and the resumes become a very time-consuming process when we scale it for large companies. Luckily with advancement of NLP (natural language processing), we can

solve this problem in a very less amount of time. Which will save the workforce and hence also some money for the company.

Let's say an HR manager can check a resume in 5 minutes and he/she works 8 hours per day. Then he/she will be able to check only 96 resumes in a whole day. Which is a work of a few seconds for a machine learning model.

According to inc.com, Google receives about 3 million resumes every year. Which according to the above estimate will take about 250 thousand hours of workforce to select candidates for the first round of the interview process. And a machine learning / NLP model will take a few minutes or hours to process based on the availability of compute process. Along with time, the NLP model will save a ton of money for the company.

There are some ATS (application tracking system) available in the industry which are few years old. But with the new NLP technologies such as transformer and attention based models we can develop far more powerful tools to solve this problem of candidate selection with much more accuracy.

In this project, We will be calculating the match percentage of the resumes for a job post. Based on the match percentage a company can ask top n candidates for the first round of the interview. Or the company can choose candidates with match percentage greater than a certain threshold.

The match percentage will indicate how fit a candidate is for the job role. A match percentage of 100 means that the candidate is perfectly fit for the job. Whereas a match percentage of 0 means that the candidate is not at all fit for the job. The higher the match percentage, the better it is for the candidate.

2. DATASET DESCRIPTION:-

To tackle this problem I will be using a dataset titled “A Perfect Fit” ^[1] which is available on Kaggle. Which was published by Mukund in 2021 under CC0: Public Domain license. This dataset originally belonged to HackerEarth's monthly machine learning challenge.

This dataset has only 1 job description/job role and 90 resumes which has output values i.e the match percentage. The job description and the resumes are of pdf file type. The resumes have different formats/templates, which is a nice thing because in the real world we see resumes of different styles.

There is a csv file too which has two columns “CandidateID” and “Match Percentage”. The “CandidateID” is same as the file name of the resume. And the “Match Percentage” is numeric value between 0 and 100. It represents how much percent a candidate is fit for the given role.

There are 60 more resumes in test folder. But we don't have access to the match percentage of those resumes. So we can't use them.

The dataset size is very small in our case. So obviously we will not face issues with processing the data. But it will create some overfit issues and the model might not generalize better. To get around this problem I can scrap LinkedIn for job description and public resumes but we won't have match percentage values. It will require manual labeling. I will do these steps to get more data if required in the future.

To process the csv data we will have to use pandas. Pandas is a very popular library to process tabular data. It is fast and reliable. Tabular data in pandas are called dataframe.

To read the pdf we can use one of the various libraries such as PyPDF or PDFMiner. These libraries can convert the well-formatted PDFs into text.

3. KEY PERFORMANCE INDICATOR (KPI) METRIC:-

There are mainly two types of metrics for machine learning use cases. One is the business metric which is mainly used by the business folks to determine how much profit will the company make. Or in our case how much money will the company save with reduction in the workload. We have discussed about the business metric in brief in the Problem Definition section.

The other type of metrics is the performance metric, which is used by the model while training. And also by the data scientists to understand and compare results of different models. When we say metric, we often mean the performance metric. The Key Performance Indicator (KPI) is the performance metric that is used for training the model.

Different types of tasks such as classification and regression have different types of performance metrics. Since we are predicting numeric values, this problem comes under the regression task. Regression models have various interesting performance metrics such as mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), mean absolute percentage error (MAPE), coefficient of determination (R^2), Huber loss, etc. Along with the traditional performance metrics, we can also view the distribution/PDF of the errors. Each of the mentioned performance metrics have their own pros and cons.

The MAE is linear whereas MSE is quadratic in nature. The MSE penalizes more if the errors are large. The MAPE does not work well when we have very small output values. MAPE is used when we could expect large difference between predicted and actual output for higher values and small difference for lower values.

I have decided to use mean squared error (MSE) as the KPI metric for this task. We will also keep track of R-squared and MAE because these are easy to understand. And we will also view the PDF of the errors wherever it is necessary to check for any anomalies.

MSE is a very simple and most commonly used performance metric for regression tasks. MSE is average of square of difference between predicted and actual outputs. MSE is always greater than equal to zero. MSE of zero means the actual outputs and predicted outputs are the same. However, we don't see MSE of zero in practice. The lower the MSE value the better it is. There is no upper bound for MSE. This make MSE a little bit hard to understand. However, we can get around this problem by comparing the MSE with a mean predictor as in R-squared. MSE is used by default for most of the regression problems. Unless a specific performance metric is required, we use MSE.

The formula for MSE is,

$$MSE = 1/n \sum_{i=1}^n \left(y_i - \hat{y}_i \right)^2$$

The formula for MAE is,

$$MAE = 1/n \sum_{i=1}^n \left(| y_i - \hat{y}_i | \right)$$

The formula for R-squared is,

$$R^2 = 1 - \frac{RSS \text{ (Sum of squares residuals)}}{TSS \text{ (Total sum of squares)}}$$

$$R^2 = 1 - \frac{\sum_{i=1}^n \left(y_i - \hat{y}_i \right)^2}{\sum_{i=1}^n \left(y_i - \bar{y} \right)^2}$$

where,

y_i = actual output

\hat{y}_i = predicted output

\bar{y} = mean of actual output

Instead of MSE we could have used RMSE or MAE. RMSE is square root of MSE, so there is not much difference, we could use either one of them. I used MSE

instead of MAE because I wanted to penalize more if the error is large. That means we are okay with multiple small errors but we would like to avoid large errors.

The advantage of MSE is that it avoids large errors. That means we can be sure that we are not making any huge mistakes in prediction. This advantage becomes a disadvantage if we have a large number of outliers. Because the outliers will affect the MSE value a lot because of the effect of squaring.

4. REAL-WORLD CHALLENGES AND CONSTRAINTS:-

In the real world, we don't have any strict latency requirements. We can calculate the percentage of match between the resumes and the job description every night, every hour, etc. Or we can calculate the percentage as soon as the candidate uploads the resume. There is no strict latency requirement because we are not showing the match percentage to the candidate and it will be used by the employer.

One challenge is that we can manually filter out few of the candidate for whom the model output a larger match percentage. But there should not be large number of such cases because it will also require some manual labour.

The most concerning challenge we could face with the model is that we could miss out on some good candidates, if the model outputs a very small value for a fit candidate. This does not only affect the candidate but also the employer. Because finding candidates is easy but finding good candidates based on the job role is not so easy.

So basically we need to avoid large errors. Hence I have used MSE as the key performance indicator.

Scalability is not a big concern here because there is no such strict latency requirement. So if the number of resumes submitted increases, the employer has to wait a little longer for the match percentage.

Interpretability of the model is not a must but a good to have feature. For interpretability, we could extract the keywords from the resume. From these keywords, one can easily understand the justification behind the predicted output value.

There are multiple challenges with generalization. There are multiple types of job role in a industry. We would want our model to work for all types of job roles. Another concern is that there are multiple templates/formats of resumes. This is a very big concern because people write resumes very differently from one another. The same resume will look very different when it follows different templates. Also choosing of words differ from person to person. Some people write explanations and some just write in very brief. We will have to tackle these generalization issues while building the model.

Till now we have discussed the constraints and challenges in productionization of the model. But there are some challenges in the training the model too. Currently, we have a very small dataset for training. We could scrap the internet for job descriptions and resumes. But we have to manually set the output values. Those output values will certainly have some human biases. These biases will make the dataset a bit noisy. Also, the current dataset contains only one job description, so we can't create a fully generalized model.

5. LITERATURE REVIEW:-

Before discussing how others have tackled this problem let me share what initial solutions we discussed in our group discussion. The first cut and the simplest solution that we could think of was using one-hot encoding and then some similarity based metric like cosine similarity. We could use this as a baseline model and build on top of it.

We could also try some simple such as Linear regression with BoW or TF-IDF or Word2Vec representations. Or advanced machine learning techniques such as XGBoost regression, Random Forest regression, etc.

Then we could also try deep learning based advanced techniques such as BERT to get embeddings of the resumes and job descriptions. And train a few simple MLPs with the embeddings.

Most of the blog posts and research papers ^[2,3] have used similar approach as our first cut solution. Where they have done preprocessing steps like tokenization, stop word removal, lemmatization, etc. and then used text featurization techniques like BoW and TF-IDF. Then they have used cosine similarity to sort the candidates based on relevance.

One more interesting approach Pradeep Kumar Roy et al. ^[3] has discussed is to first categorise the resumes in different categories. And then apply similarity technique on most relevant category of resumes. For this we would also require to categorise the job based on the description.

In another blog ^[4] , Dennis de Voogt has used spaCy tool to calculate the match percentage with a interesting and human like approach. After preprocessing, the author has used NER (named entity recognition) to extract the technical skills of the candidate from the resume text. The NER is used to extract important relevant information from text. These information could be phone number, country, etc. And in this case it is the skills of the candidate and skills mentioned in job description. After getting the entities we can use similarity based techniques.

Chapter 2

EDA AND FEATURE EXTRACTION

1. BASIC EDA (EXPLORATORY DATA ANALYSIS)

In basic eda, we will explore the given data only. That means we check the high-level stats of the given data.

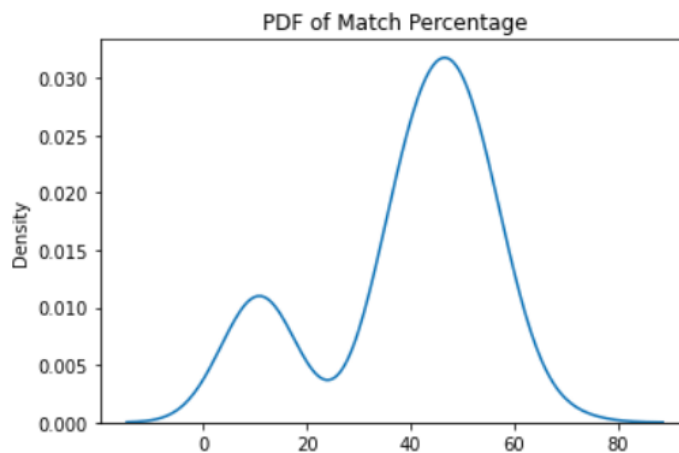
1.1. Job Description

- The job description pdf is of 2 pages.
- It has required work experience, education qualification and must have and nice to have skills mentioned.
- This is a typical job description for machine learning engineer role.

1.2. Resumes

- We have 90 data points.
- All the resumes are in pdf format.
- We have the resumes and their corresponding match percentage.
- There are two columns in the csv file,
 - CandidateID : which same as the resume file name.
 - Match Percentage : it is between 0 to 100.

- There is no missing data.
- There is no duplicate data.
- Minimum match percentage is 4.81
- Maximum match percentage is 69.21
- Mean of match percentage is 39.645
- Median of match percentage is 44.65
- There are no irregularities or outliers with match percentages.
- Match percentage is a bimodal distribution.
- There is no resume with Match percentage between 15 and 35.



- There are some spaced words like “D A T A S C I E N C E”, which we need to take care.
- Also there are short forms mostly of education qualifications like B.Tech.

1.3. Univariate and Bi-variate Analysis

- With the PDF of resumes's word length and character length, we can be certain that there are no irregularities in the resumes.
- There are minimum of 529 and maximum of 1543 characters in a resume.

- Character length is approximately gaussian distributed.
- There are minimum of 73 and maximum of 213 words in a resume.
- Word length is approximately gaussian distributed.
- There is very weak correlation between Resume character length and Match Percentage.
- There is very weak correlation between Resume word length and Match Percentage but it is better than character length.

2. DATA CLEANING

In real-world, we do not find data in a clean format. So we do various data cleaning operations to make sure that we have good quality data. Because if we do not have good quality data then whatever model we make, it will perform poorly.

- I have converted everything to lower case so that words like “Data”, “data” and “DATA” are treated the same.
- I have replaced unusual quotes like ' with ' (quote).
- I have replaced the new lines with spaces.
- I have concatenated words like "D A T A S C I E N C E" to get "DATA SCIENCE".
- I have also removed hyperlinks.
- I have converted education degrees like B.Tech or BTech to a specified form.
- I have converted skills with special symbols to words like C++ to words cplusplus
- I have replaced non-alphanumeric characters with space.

- I have removed the stop words of english.
- I have removed inflections with the word lemmatizer.

3. FEATURE EXTRACTION

Feature extraction is the art part of data science and it is the most useful skill. If we do good feature extraction then a simple model might outperform a complex model. I have created 12 interesting features.

Since I have only one job description, I did not create a few more features that could have been related to the job description and resume. Like difference of word length between job description and resume.

I have created the following features,

- *resume_word_num* : total number of words in resume
- *total_unique_word_num* : total number of unique words in job description and resumes
- *common_word_num* : total number of common words in job description and resumes
- *common_word_ratio* : total number of common words divided by total number of unique words combined in both job description and resumes
- *common_word_ratio_min* : total number of common words divided by minimum number of unique words between job description and resumes
- *common_word_ratio_max* : total number of common words divided by maximum number of unique words between job description and resumes
- *fuzz_ratio* : fuzz.WRatio from fuzzy wuzzy library

- *fuzz_partial_ratio* : fuzz.partial_ratio from fuzzy wuzzy library
- *fuzz_token_set_ratio* : fuzz.token_set_ratio from fuzzy wuzzy library
- *fuzz_token_sort_ratio* : fuzz.token_sort_ratio from fuzzy wuzzy library
- *is_fresher* : wheather a candidate is fresher or experienced
- *from_reputed_college* : wheather a candidate is fresher from reputed college

3.1. Univariate Analysis

For univariate analysis, we have various options like visualization of pdf, boxplot, violin plot, histogram. I have used pdf to get insight of each features. Along with the pdf I have also printed some stats like min, max and mean value to understand better about the distribution.

- resume_word_num is approximately log normally distributed with mean 102.789
- Minimum resume_word_num is 63 and maximum resume_word_num is 168
- total_unique_word_num is distributed widely with mean 194.067
- Minimum total_unique_word_num is 172 and maximum total_unique_word_num is 227
- common_word_num is approximately gaussian distributed with mean 19.71
- Minimum common_word_num is 9 and maximum common_word_num is 36
- common_word_ratio is approximately gaussian distributed with mean 0.10
- Minimum common_word_ratio is 0.045 and maximum common_word_ratio is 0.17
- common_word_ratio_min is approximately gaussian distributed with mean 0.245

- Minimum common_word_ratio_min is 0.118 and maximum common_word_ratio_min is 0.245
- common_word_ratio_max is approximately gaussian distributed with mean 0.148
- Minimum common_word_ratio_max is 0.068 and maximum common_word_ratio_max is 0.27
- fuzz_ratio has bi-modal distribution with mean 74.356
- Minimum fuzz_ratio is 48 and maximum fuzz_ratio is 86
- PDF of fuzz_partial_ratio is falling slowly and has mean 44.956
- Minimum fuzz_partial_ratio is 43 and maximum fuzz_partial_ratio is 48
- fuzz_token_set_ratio is approximately gaussian distributed with mean 52.867
- Minimum fuzz_token_set_ratio is 41 and maximum fuzz_token_set_ratio is 61
- fuzz_token_sort_ratio is approximately gaussian distributed with mean 50.878
- Minimum fuzz_token_sort_ratio is 40 and maximum fuzz_token_sort_ratio is 59
- There are about 28 freshers out of 90 candidates.
- For both freshers and experienced the pdf of match percentage peak at same points.
- That means the match percentage is not affected by wheather a candidate is fresher or experienced.
- There are about 14 candidates are from reputed colleges out of 90 candidates.
- For both type of candidates the pdf of match percentage peak at same points.
- That means the match percentage is not affected by wheather a candidate is from reputed or non reputed college.

- This can be because candidates from different domain might have also applied for the job. Otherwise, we could have seen some high match percentage for experienced and/or candidates from reputed colleges.

3.2. Bi-variate Analysis

For bi-variate analysis I have plotted a scatter plot between each feature and the output. And also I have used the SRCC (Spearman Rank Correlation Coefficient) to check for correlation between the features and the output. The advantage of the scatter plot is that we can see the correlations clearly. Also, the SRCC help in quantizing the correlation. Since this is a regression task there are not many options for multi-variate analysis.

- There is very weak correlation between resume_word_num and Match Percentage. And the SRCC (Spearman Rank Correlation Coefficient) is 0.285
- The SRCC between total_unique_word_num and Match Percentage is -0.02. Since it is very close to zero we can say that there is not any correlation between total_unique_word_num and Match Percentage.
- There is some correlation between common_word_num and Match Percentage and the SRCC (Spearman Rank Correlation Coefficient) is 0.512
- The SRCC between common_word_ratio and Match Percentage is 0.541 which is a good.
- The SRCC between common_word_ratio_min and Match Percentage is 0.491
- The SRCC between common_word_ratio_max and Match Percentage is 0.512
- The SRCC between fuzz_ratio and Match Percentage is -0.19. There isn't much correlation. But if we ignore the fuzz ratios above 80 then we can see a good correlation.
- The SRCC between fuzz_partial_ratio and Match Percentage is 0.344 which is again a very weak correlation.

- The SRCC between fuzz_token_set_ratio and Match Percentage is also very small and is 0.354.
- The SRCC between fuzz_token_sort_ratio and Match Percentage is 0.501, which is good enough.
- The features related to common words have a good correlation with the output.
- But the common word features are also correlated with each other.
- Specially the common_word_num, common_word_ratio and common_word_ratio_max have very strong correlation.
- So I have removed common_word_ratio and common_word_ratio_max columns.
- So we have 10 features.
- The fuzzy wuzzy features except fuzz_token_sort_ratio does not show any promising correlation.

4. FEATURE ENCODING

For feature encoding we have various options like BoW, TF-IDF, Word2Vec, Bert based, etc. For the initial phase I have opted binary bag of words and average word2vec.

We will do advanced feature encoding like sentence bert later in this project. To check if it can show a better result than the base model.

What is Binary BoW?

- In Binary BoW, we create vector, based on presence or absence of a word. If the word is present then the corresponding cell will be 1 and if not then it will

be 0. The encodings will be very sparse and high dimensional. The count BoW has counts of occurrences of a word in the document.

Why Binary BoW?

- I think the Binary BoW is a very good option because the count or the frequencies are not as important as whether a word (which could be skill) is present or not. I believe that it may outperform even bert based encodings for this task.

4.1. Binary BoW

- I have used uni-gram, bi-gram and tri-gram to get some sequence information as well.
- And I have used both job description and resume text to create the vocabulary.
- The minimum document frequency is 4. So it help in removing some non-useful words like names of candidates.
- The maximum document frequency is 99%. That means word which are very frequent will be ignored.
- The vocab size is 716. So we will get 716 dimensional output for both job description and resumes.
- We have created two new features cosine_similarity and euclidean_distance.
- cosine_similarity : It represents cosine similarity score between sentence embeddings (based on BoW) of job description and resume.
- euclidean_distance : It represents euclidean distance between sentence embeddings (based on BoW) of job description and resume.
- Now we have total of **12** extracted features.
- And the total feature dimension is $12+716+716 = \mathbf{1444}$

4.1.1. Univariate and Bi-variate Analysis

- cosine_similarity is approximately gaussian distributed with mean 0.230
- Minimum cosine_similarity is 0.117 and maximum cosine_similarity is 0.355
- PDF of euclidean_distance is falling very sharply on both side and has mean of 12.724
- Minimum euclidean_distance is 11.532 and maximum euclidean_distance is 13.928
- The SRCC between cosine_similarity and Match Percentage is 0.506, which is good.
- But the SRCC between euclidean_distance and Match Percentage is only -0.135. So we can say that there is no correlation.

4.2. Average Word2Vec

What is Average Word2Vec?

- The word2vec produces dense word embedding usually of small size like 300 dimensions. I have used the pre-trained model on google news data. I did not have much data that's why I did not train my own word2vec model. After getting the word embeddings of each word in the document. I have summed them and divided by the number of total words to get average word2vec representation.

Why Average Word2Vec?

- The Word2Vec is used to be the state of the art for word embeddings. I have used it to compare with the BoW. It would have performed better if it was trained on similar dataset
- We have created two new features cosine_similarity and euclidean_distance.

- cosine_similarity : It represents cosine similarity score between word embeddings (based on w2v) of job description and resume.
- euclidean_distance : It represents euclidean distance between word embeddings (based on w2v) of job description and resume.
- Now again we have total of **12** extracted features.
- And the total feature dimension is $14+300+300 = \mathbf{612}$

4.2.1. Univariate and Bi-variate Analysis

- cosine_similarity is approximately gaussian distributed with mean 0.171
- Minimum cosine_similarity is 0.083 and maximum cosine_similarity is 0.291
- PDF of euclidean_distance is falling very sharply on both side and has mean of 15.578
- Minimum euclidean_distance is 14.036 and maximum euclidean_distance is 17.493
- The SRCC between cosine_similarity and Match Percentage is 0.495, which is good.
- But the SRCC between euclidean_distance and Match Percentage is only -0.03. So we can say that there is no correlation.
- cosine_similarity and euclidean_distance features are very similar for both feature encodings.

5. HIGH-DIMENSIONAL DATA VISUALIZATION

Since this is a regression task. The high-dimensional data visualization does not make much sense.

We could have used PCA or t-SNE to reduce the dimensionality to visualize. But I have ignored it because it is not relevant with this problem.

Chapter 3

MODELLING AND ERROR ANALYSIS

1. MODELS WITH BOW FEATURES

I have created the input and output data by combining extracted features and bow representations of resumes and job descriptions.

- We have 90 data points.
- We have 1444 features.

1.1. Train Test split

Outputs of both test and train data have the same distribution.

At first, I had divided the data into train, cv, and test data with 60%, 20%, and 20% respectively in each group.

But the models were not showing very different results when I changed the random state for the split.

This is because we have a very small dataset.

So I decided to use K-fold cross-validation for hyperparameter tuning. And I have divided the data as follows,

- Train data is 70% and has 63 data points.
- Test data is 30% and has 27 data points.

1.2. Preprocessing

In preprocessing I have done standardization which means mean centering and variance scaling. So now all columns will have mean of 0 and variance of 1.

1.3. Modeling (Basic)

Initially, I have tried only two models. One is Linear Regression with L2 regularizer which is a simple model. And the other is KNN Regression which is a little bit of complex model.

After hyper-param tuning with hyperopt, I found the best alpha for the linear regression model is 0.109. But when we calculate the MSE errors we found that there is huge gap in train and test mse errors. The mse on train data is 4.79 whereas the mse on test data is 91.48.

So, I plotted the test and train mse loss for different alpha values. From the plot, we can see that the model is overfitting. The difference between train and test mse decreases for higher alpha values. But in that case, both mse losses will increase significantly.

Since Linear regression is a simple and high bias model we should not see such overfitting. I had also tried linear regression with L1 regularizer and it was showing a very similar results.

Similarly for KNN regression, I found the best params (`n_neighbors=7`, `metric='cosine'`). But the KNN is performing very badly on both train and test data. The mse on train is 157.02 and test is 148.5.

So I chose the Linear regression model with L2 regularizer for error analysis.

1.4. Error Analysis

For BoW, I had used a minimum document frequency of 2, but the models were overfitting so I chose `min_df = 4`. Which improved the result a little bit but not much. I checked the data points which showed higher errors and compared different resumes.

I also tried the linear regression model multiple times and checked the feature importance. It showed some traces of collinearity.

These collinearities or multi-collinearities could be because we do have short and long forms of some skills and degrees. And also because we have unigram, bigram, and trigram.

So I did forward feature selection with `SequentialFeatureSelector` to select 500 features. After that when I fitted a linear regression. And I found that only less than 100 weights were non-zero.

So again did forward feature selection to select only the top 100 features. So now we have reached from 1444 features to 100 features, which means we have only 6.93% of the original features.

When we view the selected features we see that we have some of the extracted features. Also, we have more number of features or words from the resume than the jd. That's because we have only one jd for the whole dataset.

1.5. Train Test split and Preprocessing

Now again I have done train test split with the same random state. So both the data will contain the same data point as earlier. I have done the mean centering and variance scaling. And also I have saved the data points.

1.6. Modeling

Now again I have done hyper-param tuning for L2 Linear Regression.

- The best alpha is 1.659.
- The train mse and r-squared are 1.9844 and 0.9924 respectively.
- The test mse and r-squared are 52.1386 and 0.7784 respectively.

For KNN Regression,

- The best params are n_neighbors=7 and metric='cosine'
- The train mse and r-squared are 130.7427 and 0.5014 respectively.
- The test mse and r-squared are 113.766 and 0.5166 respectively.

For Decision Tree Regressor,

- The best params are max_depth=5, min_samples_split=2
- The train mse and r-squared are 21.9602 and 0.9163 respectively.
- The test mse and r-squared are 164.5762 and 0.3006 respectively.

For Support Vector Regression (RBF Kernel),

- The best param is C = 253.341
- The train mse and r-squared are 0.01 and 1 respectively.
- The test mse and r-squared are 90.842 and 0.614 respectively.

For Support Vector Regression (Linear Kernel),

- The best param is C = 1.429
- The train mse and r-squared are 3.0771 and 0.9883 respectively.
- The test mse and r-squared are 49.5258 and 0.7895 respectively.

For Random Forest Regressor,

- The best params are n_estimators = 50, min_samples_split = 5, max_depth = 15
- The train mse and r-squared are 27.0421 and 0.8969 respectively.
- The test mse and r-squared are 125.082 and 0.4685 respectively.

For XGBoost Regressor,

- The best params are `n_estimators = 150`, `max_depth = 2`, `learning_rate = 0.7`, `reg_lambda = 38.924`
- The train mse and r-squared are 3.2299 and 0.9877 respectively.
- The test mse and r-squared are 133.6595 and 0.432 respectively.

Here we can see that the best model is Linear kernel based SVR followed by the Linear regression with l2 regularizer.

2. MODELS WITH AVERAGE WORD2VEC FEATURES

I have created the input and output data by combining extracted features and average word2vec representations of resumes and job descriptions.

- We have 90 data points.
- We have 612 features.

2.1. Train Test split

Again we have done a train-test split just like before. Since we have used the same random state. The data points will be same as before in train and test splits.

2.2. Preprocessing

Again I have done column standardization with mean centering and variance scaling.

2.3. Modeling (Basic)

I have used Linear Regression with l2 regularizer and KNN Regression for initial analysis.

After hyperparameter tuning for Linear Regression, I found the best $\alpha=88.674$.

And train and test mse are 17.155 and 111.87 respectively.

For KNN Regression the best hyperparameters are `n_neighbors=7` and `metric='cosine'`.

And train and test mse are 155.137 and 100.862 respectively.

We can see the results are not very promising.

2.4. Error Analysis

From the previous experience and from the fact that we have only one `jd`, I had a hunch that we can get better results after doing forward feature selection.

I used the linear regression for forward feature selection and selected 200 features out of 613 features.

2.5. Train Test split and Preprocessing

Just like before, we have done train test split. Again we have preprocessed the data to have mean of 0 and variance of 1. And save them in file.

2.6. Modeling

For L2 Linear Regression.

- The best alpha is 8.287.
- The train mse and r-squared are 4.4046 and 0.9832 respectively.
- The test mse and r-squared are 59.5001 and 0.7472 respectively.

For KNN Regression,

- The best params are `n_neighbors=7`, `metric='cosine'`
- The train mse and r-squared are 105.3122 and 0.5984 respectively.
- The test mse and r-squared are 83.3849 and 0.6457 respectively.

For Decision Tree Regressor,

- The best params are `max_depth=10`, `min_samples_split=5`
- The train mse and r-squared are 2.4671 and 0.9906 respectively.
- The test mse and r-squared are 389.3863 and -0.6546 respectively.

For Support Vector Regression (RBF Kernel),

- The best param is $C = 399.718$
- The train mse and r-squared are 0.01 and 1 respectively.
- The test mse and r-squared are 53.8473 and 0.7712 respectively.

For Support Vector Regression (Linear Kernel),

- The best param is $C = 385.101$
- The train mse and r-squared are 0.01 and 1 respectively.
- The test mse and r-squared are 109.3227 and 0.5355 respectively.

For Random Forest Regressor,

- The best params are $n_estimators = 50$, $min_samples_split = 5$, $max_depth = 20$
- The train mse and r-squared are 26.4767 and 0.899 respectively.
- The test mse and r-squared are 113.132 and 0.5193 respectively.

For XGBoost Regressor,

- The best params are $n_estimators = 100$, $max_depth = 2$, $learning_rate = 0.04$, $reg_lambda = 0.22$
- The train mse and r-squared are 9.3477 and 0.9644 respectively.
- The test mse and r-squared are 109.5243 and 0.5346 respectively.

The RBF kernel based SVR have the lowest mse for test data. But when we compare train and test mse for it then it is clear that the model is overfitting a lot.

The Linear regression will be the best choice among these models. Since it has lower mse and also it is not overfitting that much.

3. PERFORMANCES

The top 3 models based on MSE on test data are,

1. Support Vector Regression (Linear Kernel) model based on BoW feature

2. Linear Regression (L2 Regularizer) model based on BoW feature
3. Support Vector Regression (RBF Kernel) model based on Average Word2Vec feature

The worst performing models are Decision Tree Regressor models.

4. STACKING ENSEMBLE (STACK OF BEST MODELS)

I was not very impressed with the result. So I thought, can we somehow combine the best of both the features to create a better model.

That reminded me of the stacking ensemble. So I chose the following models for level-0 of stacking,

- Support Vector Regression (Linear Kernel) model based on BoW features
- Linear Regression (L2 Regularized) model based on Word2Vec features

4.1. Preprocessing

I have done similar preprocessing as above. I have done mean centering and variance scaling.

4.2. Finding the Meta model

The stacking gave very good results. Here's what I found for stacking ensemble models.

For L2 Linear Regression.

- The best alpha is 0.068.
- The train mse and r-squared are 2.1757 and 0.9917 respectively.
- The test mse and r-squared are 33.8341 and 0.8562 respectively.

For KNN Regression,

- The best params are n_neighbors=2, metric='euclidean'

- The train mse and r-squared are 1.5009 and 0.9943 respectively.
- The test mse and r-squared are 20.6566 and 0.9122 respectively.

For Support Vector Regression (RBF Kernel),

- The best param is $C = 271.031$
- The train mse and r-squared are 2.0037 and 0.9924 respectively.
- The test mse and r-squared are 30.6849 and 0.8696 respectively.

For Support Vector Regression (Linear Kernel),

- The best param is $C = 8.179$
- The train mse and r-squared are 2.661 and 0.9899 respectively.
- The test mse and r-squared are 44.1259 and 0.8125 respectively.

The KNN regressor seems to be performing the best as the meta-model for the stacking regressor.

4.3. Train and Test performance with the best model

With our best model which is the stacking ensemble model, where the base models or level-0 models are the *Support Vector Regression (Linear Kernel) model based on BoW features* and *Linear Regression (L2 Regularized) model based on Word2Vec features*, and the meta-model or level-1 model is a *KNN Regressor*.

We get the following results,

- Train MSE is 1.5009 and Test MSE is 20.6566
- Train R-Squared is 0.9943 and Test R-Squared is 0.9122
- Train MAE is 0.8253 and Test MAE is 3.7485

Which seems very impressive.

5. FURTHER ANALYSIS OF THE BEST MODEL

- The Support Vector Regression (Linear Kernel) model based on BoW features has 100 features out of which only 55 of them are non zero.
- The Linear Regression (L2 Regularized) model based on Word2Vec features has 200 features out of which only 86 of them are non zero.

Stats on test errors are,

- The minimum value is -9.91
- The maximum value is 9.39
- 75% of the errors are between -1.6875 and 4.6275

6. ADVANTAGES AND DISADVANTAGES OF THE MODELS

6.1. Linear Regression

Advantages:-

- It's a very simple model.
- It's a high bias model so there is less chance of overfitting.
- We can easily get feature importance with the absolute value of the weights.
- It's very fast because there aren't many operations involved.

Disadvantages:-

- It's very simple, so it can't capture complex patterns.
- Because it is a high bias model, it can underfit easily.
- It is affected by outliers. So we should use RANSAC.
- Feature importance can get affected because of collinearity.

6.2. Support Vector Machine

Advantages:-

- It only cares about the support vectors. So it is less affected by outliers.
- It works great for high-dimensional data.
- Because of the kernel, we can use it even when a similarity matrix is given as the data.

Disadvantages:-

- Choosing the right kernel is a hard task.
- It slows down heavily when the dataset is large.

6.3. K-NN Regressor

Advantages:-

- It is easy to understand.
- Even though it directly does not provide feature importance. It is interpretable when we view the neighbourhood data points.
- No training is required because it is instance-based.

Disadvantages:-

- It has high time complexity.
- It is affected by the curse of dimensionality.
- Since it is instance based we need to store the train data in memory.

Chapter 4

ADVANCED MODELING AND FEATURE ENGINEERING

1. FEED FORWARD NEURAL NETWORK WITH BOW FEATURES

1.1. Dataset

In advance modeling, first I started with the final BoW based features. The features which we got after applying forward feature selection.

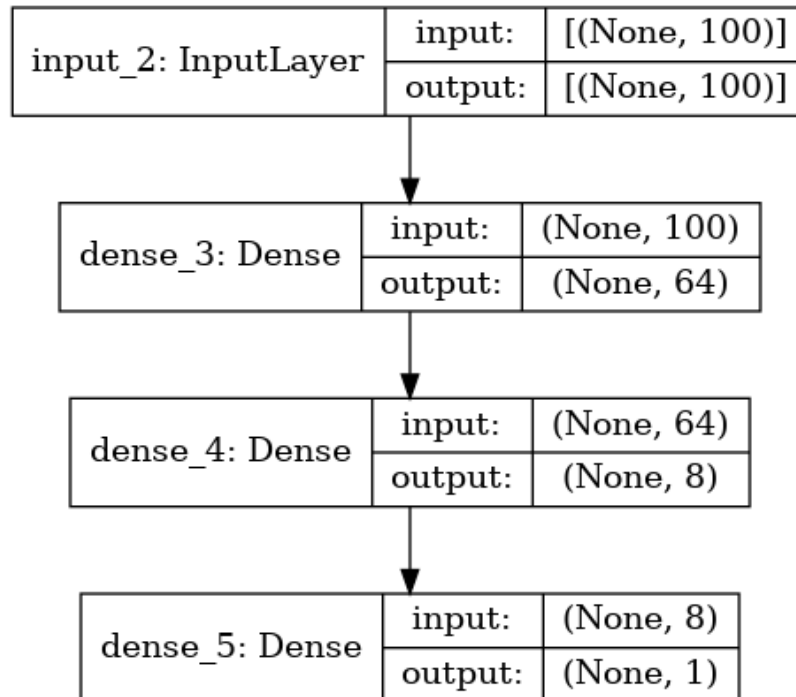
It only has a feature dimension of size 100. That means it only have 100 words from the whole corpus. Which is kind of better than NER (named entity recognition) because we have used the data and model to get these important words.

I have divided the data into three parts: train, cross validation (cv) and test.

- We have a total of 90 data points.
- We have 100 features.
- We have 53 data points for training.
- We have 10 data points for cross validation.
- We have 27 data points for testing.

1.2. Model Architecture

Then I created simple feed forward neural network as below,



After a lot of trials I settled with this architecture. It has a total of 6993 trainable parameters. And all three activation functions have ReLU activation function.

In regression tasks, we generally use linear activation functions. But since this is a match percentage prediction task, where there are no negative values. So it makes sense to use ReLU instead of linear activation function.

1.3. Training and Evaluation

During training I found that the model is overfitting a lot and the MSEs are almost constant for train and cv data. I even tried with a very small learning rate but without any luck.

- Train mse (mean squared error) is 1904.343
- Train mae (mean absolute error) is 40.7598
- CV mse (mean squared error) is 1219.7331
- CV mae (mean absolute error) is 30.2521
- Test mse (mean squared error) is 1800.1959
- Test mae (mean absolute error) is 39.575

From these performance metrics we can conclude that this model is very poor.

2. FEED FORWARD NEURAL NETWORK WITH BOW AND AVERAGE W2V FEATURES

2.1. Dataset

After the poor performance of the previous model. I thought of training a feed forward neural network with final BoW and final Average Word2Vec features.

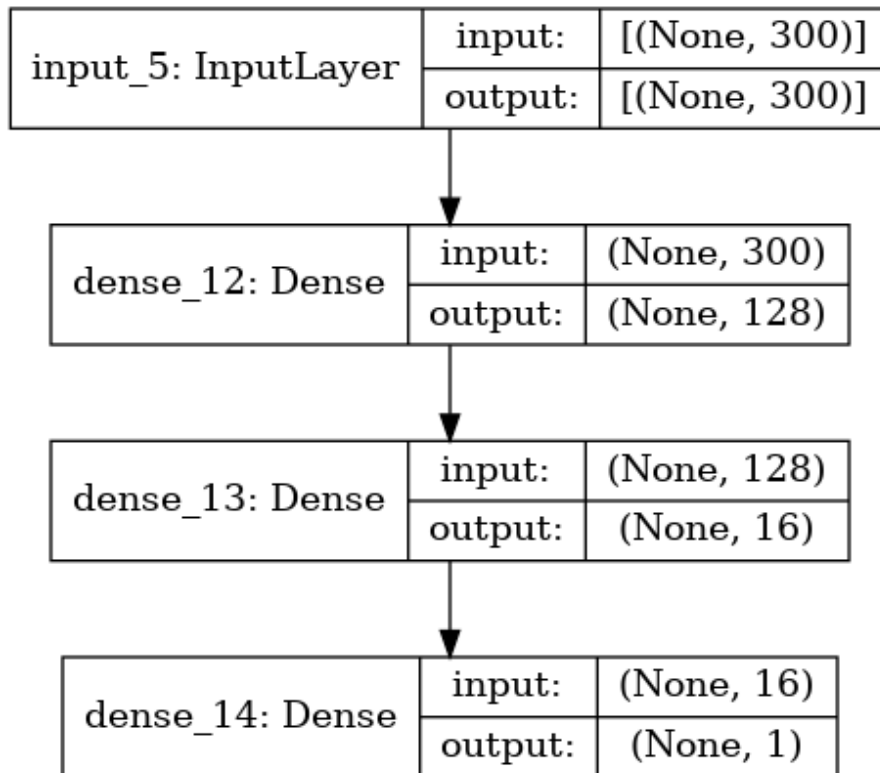
As I mentioned earlier we have 100 features for the final BoW. And for the final average word2vec we had 200 features. Which brings our total feature dimension to 300.

I have divided the data into three parts: train, cross validation (cv) and test.

- We have a total of 90 data points.
- We have 300 features.
- We have 53 data points for training.
- We have 10 data points for cross validation.
- We have 27 data points for testing.

2.2. Model Architecture

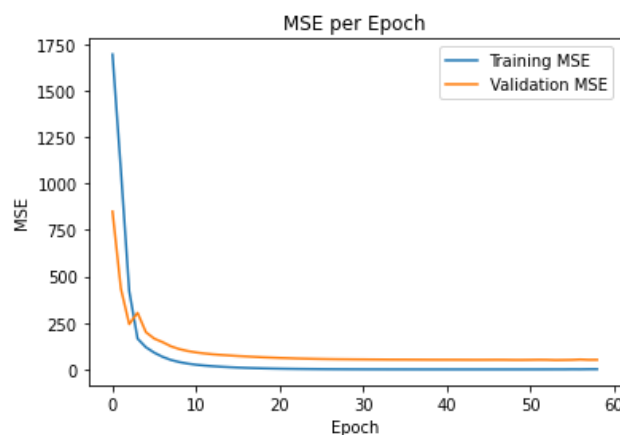
Since now we have more features I decided to use a few more activation functions than before. The model architecture is shown below,



It has a total of 40,609 trainable parameters. And all three layers have ReLU activation functions as before. I had to do quite a bit of tweaking to get the best architecture.

2.3. Training and Evaluation

At first, the model ran for 59 epochs and the performance was way better than the previous model. But it seems that the model is overfitting a lot.



We can see that the error is decreasing very fast but it becomes almost constant after a few epochs. Also the mse and mae of train and cv data is pointing that the model is overfitting.

So I decided to execute it only for 20 epochs. And we got a better result.

- Train mse (mean squared error) is 3.0928
- Train mae (mean absolute error) is 1.3764
- CV mse (mean squared error) is 72.9421
- CV mae (mean absolute error) is 7.0519
- Test mse (mean squared error) is 39.6974
- Test mae (mean absolute error) is 5.3252

This is very impressive. As we can see this model is very comparable to the stacking ensemble model, where the base models or level-0 models are Support Vector Regression (Linear Kernel) model based on BoW features and Linear Regression (L2 Regularized) model based on Word2Vec features and the meta model or level-1 model is a KNN Regressor.

This makes sense because both models are based on the same features.

3. BERT WITH FEED FORWARD NEURAL NETWORK

Till now we have used previous features with simple feed forward neural networks. But BERT (Bidirectional Encoder Representations from Transformers) is a SOTA (state of the art) technique for NLP tasks. It is based on transformer which usage encoder only structure. And as the name indicates it is bi-directional. BERT base can take a maximum of 512 worded input and it outputs 768 dimensional representation for each token.

BERT is trained on huge dataset of google books, wikipedia and other internet crawled datasets. It is trained with Masked Language Modeling task and Next Sentence Prediction task. So naturally it assumes the sequential nature of the text.

But here we do not have a sequential data because we have very few sentences in the resumes. So even though it is sota for nlp tasks it should not work.

BERT combined multiple research papers ideas such as,

I) Semi supervised sequence learning -

It does semi-supervised sequence learning with Masked Language Modeling and Next Sentence Prediction tasks.

II) Contextualised word embeddings -

It got the idea of contextualised word embeddings from ELMO paper. Where each word embedding is generated based on the whole sentence.

III) Transfer Learning -

It got the idea of transfer learning from ULM-FiT. We get the pre-trained BERT model and then we can do fine tuning as per our need.

IV) Encoder Only Transformer -

It got this idea from OpenAI transformer which is a decoder only transformer.

Here I have used DistilBERT, which is a simpler and faster version of BERT with a very minimal decrease in performance. I have loaded the distilbert-base-uncased that means the case does not matter.

3.1. Dataset

I have loaded the PDFs and then done encoding with DistilBERT based tokenizer with max length = 300 and zero padding if required. Zero padding is done so that we can perform batch training.

Then using the pretrained model I have extracted the output corresponding to CLS token which will be of 768 dimension. I have done this for both job description and resume.

I have divided the data into three parts: train, cross validation (cv) and test.

- We have a total of 90 data points.
- We have 1536 features. (768+768)
- We have 62 data points for training.

- We have 9 data points for cross validation.
- We have 19 data points for testing.

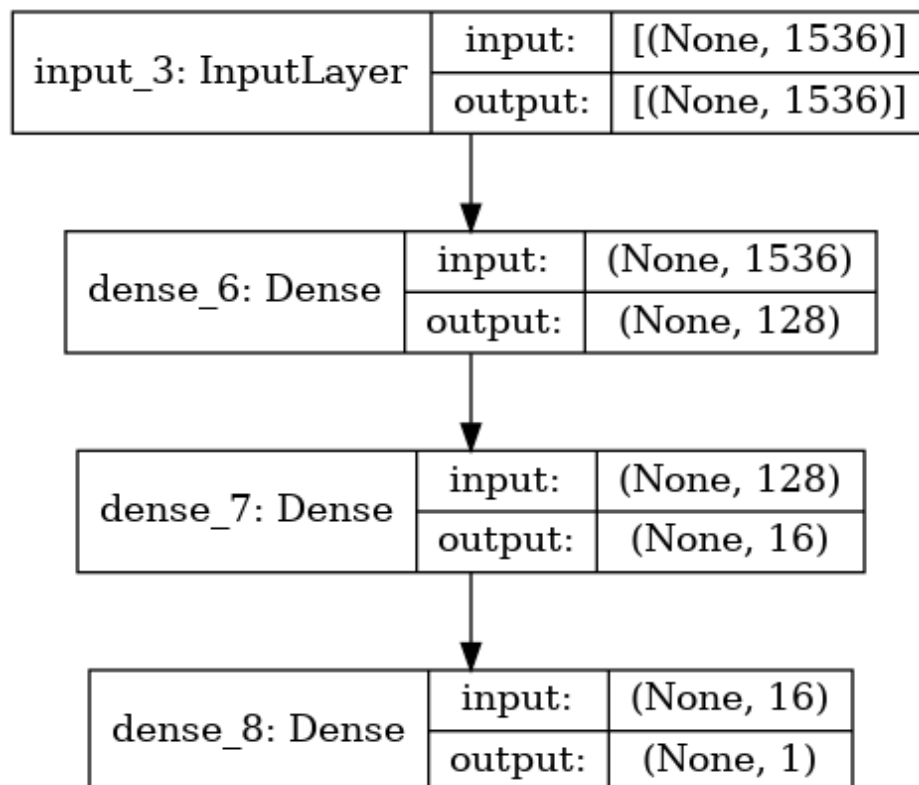
Distribution of output for all three data are almost overlapping. That means the data split is good.

3.2. Model Architecture

Just like before the model has three dense layer and all of them have ReLU activation unit. The total trainable params are 198,817.

In all the models I have used Adam optimiser. Because it is the best optimiser as of now.

The model architecture is as shown below,



3.3. Training and Evaluation

As expected the performance is not that good. Just like the initial model the loss is not decreasing at all.

We get following performances from the model,

- Train mse (mean squared error) is 1780.1329
- Train mae (mean absolute error) is 38.9871
- CV mse (mean squared error) is 1070.5638
- CV mae (mean absolute error) is 27.6278
- Test mse (mean squared error) is 1803.7161
- Test mae (mean absolute error) is 37.4089

4. CONCLUSION

The Feed Forward Neural Network with BoW and Average W2V features model is performing the best but still not as good as the stacking ensemble model.

The main reason for deep learning models to not perform well is because we do not have much data. And also there is no sequence structure even though it is a text data.

This is one of those cases where a simple model performs better than a complex deep learning model.

Chapter 5

DEPLOYMENT AND PRODUCTIONIZATION

1. REQUIREMENTS

1.1. Applications

First we need to install following applications,

- Python 3
- Python 3 PIP
- Gunicorn

1.2. Python Libraries

Then we need to install python libraries,

- Flask
- numpy
- pandas
- pdfplumber
- matplotlib
- fuzzywuzzy
- fuzzywuzzy[speedup]
- joblib
- gunicorn
- nltk
- gensim

- python-Levenshtein
- scikit-learn==0.24.2

2. WEB FRAMEWORK

The web framework allows us to easily create websites. There are multiple web frameworks that we can use for deployment like flask, django, etc. We can even build an API in python and create the web interface in another language.

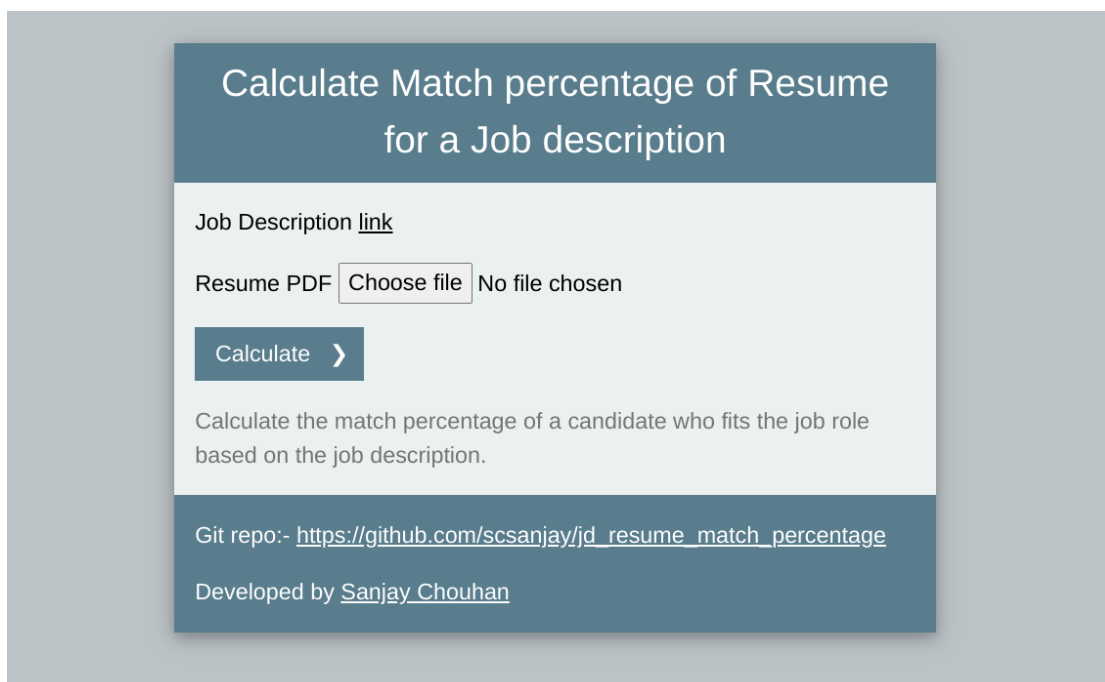
2.1. Flask

I have decided to use Flask which is a micro web framework in python.

It is generally used for creating APIs and building small websites. We will build a simple web-interface so we can use flasks for our task.

2.1.1. Homepage

I have built a homepage where the user can upload the resume. The interface is very simple as we can see below.



The screenshot shows a web application interface with a dark blue header containing the title "Calculate Match percentage of Resume for a Job description". Below the header, there is a light blue form area. The form contains a label "Job Description [link](#)" followed by a text input field. Below this, there is a label "Resume PDF" followed by a "Choose file" button and the text "No file chosen". A dark blue "Calculate" button with a right arrow is positioned below the file selection area. Underneath the button, there is a paragraph of text: "Calculate the match percentage of a candidate who fits the job role based on the job description." At the bottom of the form, there is a dark blue footer area containing the text "Git repo:- https://github.com/scsanjay/jd_resume_match_percentage" and "Developed by [Sanjay Chouhan](#)".

I have fixed the Job description to reduce complexity. And also because we trained the model with only one job description so it will not perform well with other job descriptions.

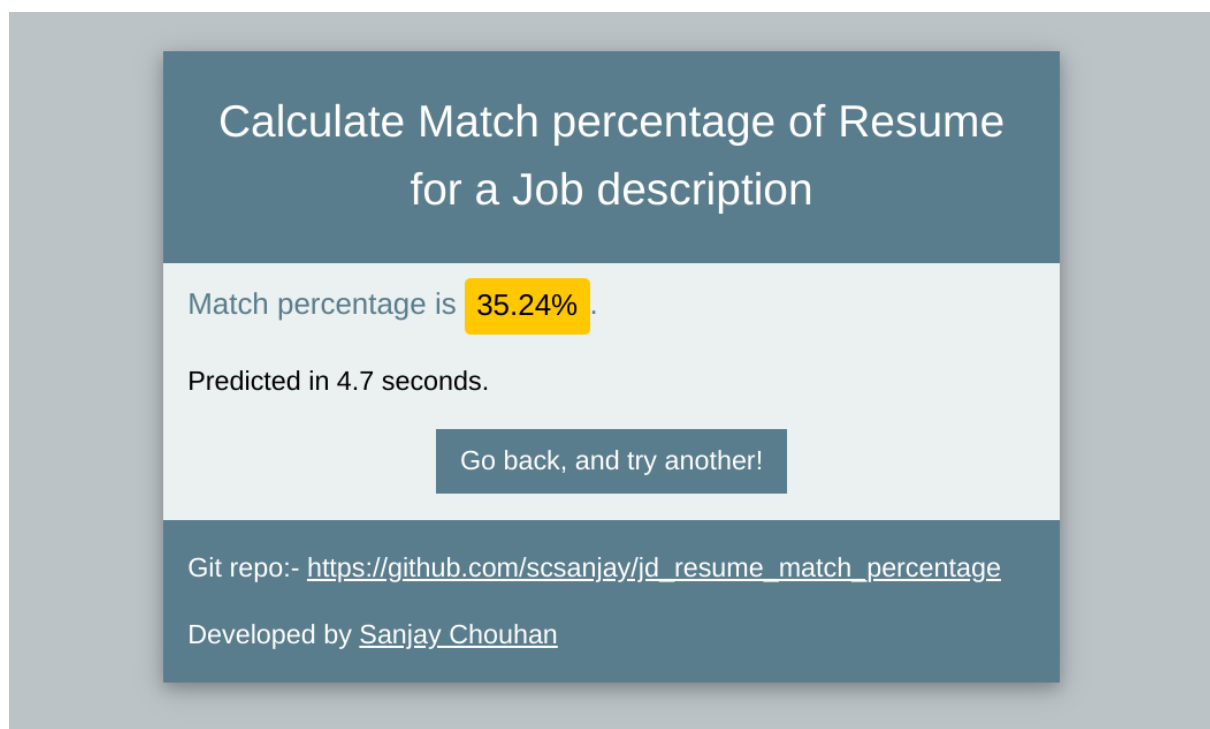
Then we have the input field to upload a resume. It will only accept pdf files of less than 3MB.

And we have the “Calculate” button to submit the form and show the result.

2.1.2. Predict page

On the predict page I have simply displayed the match percentage. And we have a back button to take back to the homepage so that the user can try again with a different resume.

Then we have the predict page to show the calculated match percentage and also the prediction time,



In an ATS system we would have a different approach. There we would have form which generally get filled automatically when we upload the resume. And the user is allowed to modify the data in case there are some mistakes. This way we get structured data and we can train better models.

Also in ATS we do not show the match percentage to the user but show the match percentage to the hiring manager in the backend.

2.2. Flask code

2.2.1. match.py

In match.py I have initialised flask. It will work as an entry point. Here I have mentioned the constants like upload folder, max size of the resume, etc.

I have created a route for the homepage which will be shown when the user browses either `/` or `/index`. For the homepage we have just rendered the template.

Another route is for the predict page (`/predict`) which only accepts POST requests. Here at first I have added some constraints which will redirect back to the homepage if not met.

Then I have uploaded the resume file to the server. And loaded the pdf to preprocess it. After preprocessing I have passed it through the models to get the prediction. And displayed the result on the prediction page.

2.2.2. helper.py

The helper.py contains all the preprocessing related functions. I have created this so that we do not clutter all the codes.

2.2.3. models folder

This contains all the preprocessors, models and some static values.

2.2.4. static folder

The static folder is for css, js and favicon.

2.2.5. templates folder

We have stored all the html templates in this folder.

2.2.6. uploads folder

This folder will be used to save the user uploaded resume files. This is a temporary folder so we should clean it from time to time.

3. HOSTING

3.1. WSGI Server

While lightweight and easy to use, Flask's built-in server is not suitable for production as it doesn't scale well. So for productionising a flask based website we need to use some other WSGI Server (Web Server Gateway Interface HTTP server).

There are various options productionisation of Flask apps.

Some of the hosted options are,

- Heroku
- Google App Engine
- Google Cloud Run
- AWS Elastic Beanstalk
- Amazon Lightsail Containers
- PythonAnywhere
- Azure App Service

And some of the self hosted options are,

- Gunicorn
- uWSGI
- CGI

3.1.1. Gunicorn

Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX. It provides a perfect balance of performance, flexibility, and configuration simplicity. With worker

parameters it allows us to manage a number of processes to run parallelly for scalability.

3.2. Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in AWS. EC2 is highly scalable and can be launched with only a few clicks. Amazon EC2 is IAAS (Infrastructure as a Service). Here we get a virtual computer with some RAM, CPU, hard disk, OS, etc. We can use this to host a website. And Amazon gives us full control over it.

First I launched a t2.micro instance of amazon EC2 which is a free tier instance. It has only 1 CPU and 1 GB RAM.

Then I created an Elastic IP address and assigned it to the EC2 instance. Because the IP address of the instance might change if we restart the system.

Also we need to create and assign a security group which allows TCP connection to the port on which we will be running the Gunicorn. Otherwise we won't be able to access the instance from the browser.

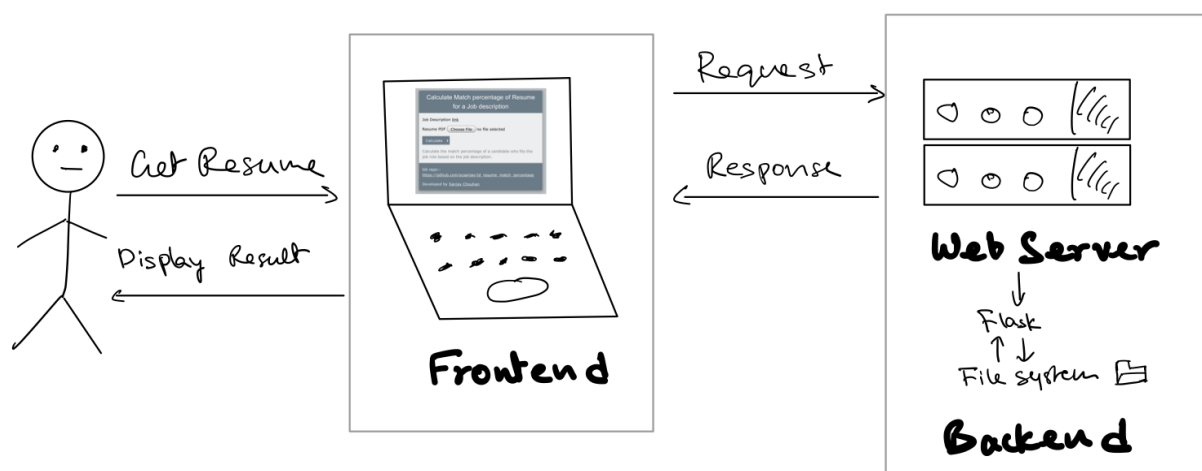
After the instance setup I transferred all the files of the web app to the EC2 with SFTP. Then I connected to the instance's terminal with SSH and ran the gunicorn command to serve the website.

4. OPTIMISATION

I have used some simple optimisation hacks to run the website on a small server. Since the job description is fixed, I have preprocessed the job description and calculated BoW and average word2vec representation of the job description. This will certainly improve the prediction time.

The pretrained word2vec is 3.7GB that means we won't be able to run it in a system with 1GB RAM. So I limited the vocab size to 3 lakh and also I changed the data type of the representation from float32 to float16. With this I was able to reduce the size to 190 MB.

5. ARCHITECTURE DIAGRAM



6. SCALABILITY AND LATENCY

6.1. Scalability

We can easily scale the website because we are hosting on EC2 instance and using gunicorn. So whenever required we can change the instance type and use a larger instance. Also we can spawn more instances of the application with gunicorn's worker parameter.

We can use caching for the homepage to make it faster. Also we can store the word and the corresponding word2vec representation in an indexed table. This will help

with RAM and the speed because then we don't have to load the word2vec model and we can get the representation in $O(1)$ time.

6.2. Latency

On the EC2 instance where we have lots of constraints the latency is 3.9 seconds including the file upload time. Which is good.

On local system where we don't have any constraints the latency is only 1.2 seconds including the file upload time.

So if we want to improve the latency we can increase the RAM and run more workers. And also we can increase the network bandwidth of the EC2 instance. With caching we can improve the load time of the homepage.

If we want more speed we can use C++ to implement the models.

7. LINKS

Github Repo - https://github.com/scsanjay/jd_resume_match_percentage

Blog Link - <https://sanjayc.medium.com/aa3adc7bfbb5>

Demo Link - <http://13.234.90.146:8080/>

BIBLIOGRAPHY

- Mukund. A Perfect Fit. <https://www.kaggle.com/mukund23/a-perfect-fit> (2021)
- AppliedRoots. [<https://www.appliedroots.com/>]
- Sanjay Chouhan. Towards data science. The Quora Question Pair Similarity Problem. 2021.
[<https://sanjayc.medium.com/the-quora-question-pair-similarity-problem-3598477af172>]
- FuzzyWuzzy. SeatGeek. [<https://github.com/seatgeek/fuzzywuzzy>]
- Scikit-learn. [https://scikit-learn.org/stable/user_guide.html]
- XGBoost doc. [<https://xgboost.readthedocs.io/en/stable/>]
- Tensorflow API Doc. [https://www.tensorflow.org/api_docs/python/tf]
- Transformers. Hugging Face. [<https://huggingface.co/docs/transformers/index>]
- Jay Alammar. The Illustrated BERT, ELMo, and co. 2018.
[<https://jalammar.github.io/illustrated-bert/>]
- Flask. [<https://flask.palletsprojects.com/>]
- Serve Flask with Gunicorn. Digitalocean.
[<https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-gunicorn-and-nginx-on-ubuntu-18-04>]