US Software Engineering Job's Analysis

Sai Ganesh Pendela, Akash Perni, Samba Sivesh Kurra, Venkata Phani Aditya Nutalapati
Maryland Applied Graduate Engineering
University of Maryland
Collge Park, MD, United States
spendela@umd.edu, aperni@umd.edu, shiv1818@umd.edu, vnutala1@umd.edu

Abstract—**This paper explores the application of machine learning algorithms in classifying the salaries of various software Engineering jobs based on different features like location, type of job, Company, Rating, etc. Four different machine learning algorithms are employed namely K-Nearest Neighbors, Logistic Regression, Decision Tree, and Random Forest, their results are compared based on accuracy, recall, precision, and cross-validation scores. Results indicate promising accuracy across all models except Logistic Regression, with Random Forest exhibiting remarkable accuracy and performance, providing a strong foundation for classifying salaries.**

Index Terms—**Software Jobs, Salaries, Logistic Regression, Decision Tree, K-Nearest Neighbors, Random Forest, accuracy, recall, precision, cross-validation.**

I. Introduction

The decision problem focuses on the classification of salaries into various ranges. In this paper, the US Software Engineering Jobs dataset is used to train and test different machine learning algorithms.

The dataset [1] comprises of 58,433 instances with 29 features. The objective is to predict the salary of a certain job based on its features. Here the dependent variable is ***salary*** and the independent variables are ***title, company, salary, rating, review_count, types, location, relative_time, hires_needed, hires_needed_exact, urgently_hiring, remote_work_model, snippet, dradis_job, link, new_job, job_link, sponsored, featured_employer, indeed_applyable, ad_id, remote_location, source_id, hiring_event_job, indeed_apply_enabled, job_location_postal, company_overview_link, activity_date, location_extras***.

The rest of the paper is organized as follows, Section II mentions the contributions of individuals in the project, the Methodology section describes the methodology to build Logistic Regression [2], Decision Tree [3], K-Nearest Neighbours [4], Random Forest [5]. and it also describes how to calculate various classification metrics. The Results section provides a comparison of the results of the models. In the Conclusion section, possible reasons for the Results obtained are explained.

II. Contributions

The teamwork of **Sai Ganesh Pendela, Akash Perni, Samba Sivesh Kurra, Venkata Phani Aditya Nutalapati** in our data science project has produced a thorough and impactful outcome. Akash Perni led the Exploratory Data Analysis, offering foundational insights, while Aditya excelled in Feature Engineering, enhancing the dataset's predictive capabilities. Sai Ganesh Pendela and Samba Sivesh Kurra collaborated on model building, hyperparameter tuning, and classification metrics, significantly boosting the system's accuracy and reliability. All four of us kept an helping hand on informative presentation and research paper, showcasing the project's goals and accomplishments.

III. Methodology

In this section, the necessary libraries required for the classification, the steps to load the dataset, preprocessing of the data, and building of the different models i.e, Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbours on the US Software Jobs dataset is explained. Also, the functions used to calculate the metrics is discussed.

A. Setting up and importing necessary packages

The Machine Learning models are built using **Google Colab** [7]. The exact versions of them are mentioned in table I

Software	**Version**
Google Colab	Colab Free Tier

TABLE I
SOFTWARE AND THEIR VERSIONS

In addition to Google Colab, it is advisable to use Python version 3.8.*, specifying the exact version for compatibility assurance. The development of classifiers necessitates the integration of essential Python packages, detailed in the provided table II. These packages play a crucial role in streamlining the development process, ensuring a conducive environment for effective classifier implementation.

Module	**Version**
Numpy [8]	1.24.4
Scikit-Learn [9]	1.3.0
Pandas [10]	2.0.3
Matplotlib [11]	3.7.2

TABLE II
MODULES USED AND THEIR VERSIONS

B. Loading the US Software Jobs dataset

The US Software Jobs dataset is readily available for download in CSV format from Kaggle. Once downloaded, you can import the dataset into Google Colab as a DataFrame [12] using the Pandas *read_csv()* [13] function. This function accepts the *file path* as an input and returns a DataFrame object. Utilizing a DataFrame to load the data is advantageous because it enables us to access the dataset's columns individually, unlike the conventional row-wise approach commonly used by most software.

C. Data Preprocessing

Before constructing the models, the dataset undergoes a check for null values using the ***isnull()*** [15] function. The analysis reveals the presence of null values, prompting the filling of missing values in the salary column with the mean salary determined by the job title, company, and location combined. Following the null value check, categorical columns are converted into numerical representations using the ***Word2Vec()*** [16] function. Word2Vec encoding transforms words into numerical vectors, capturing their meaning for machine learning models to effectively understand and process textual data.

After Word2Vec encoding, a check for duplicate values is conducted using the ***drop_duplicate()*** [17] function, revealing the absence of duplicates in the dataset. Subsequently, standardization is applied using the ***StandardScaler()*** [18] function. This process transforms the data to have a mean of 0 and a standard deviation of 1, ensuring features are comparable and suitable for certain machine learning algorithms.

In the feature engineering phase, new dimensions were introduced to enhance the dataset's informativeness. Specifically, 'years of experience' and 'relative_time' of job postings were incorporated. The 'snippet' column, encapsulating brief job descriptions, provided valuable insights into the requisite experience level for prospective applicants. To gauge the recency of job postings, the number of days was derived from the 'relative_time' column. Furthermore, annual salaries were computed by considering the provided salary information in various units, such as months and days. In instances of salary ranges, the midpoint was chosen as the estimated annual salary.

D. Data partitioning

Data partitioning is done to separate a dataset into distinct subsets for training and testing machine learning models, enabling model evaluation on unseen data and assessing generalization performance. To perform data partitioning, the ***train_test_split()*** [19] function from Scikit-Learn is used. The function takes *data*, *ground truth* and *test_size* as input to return the *X_train*, *X_test*, *y_train* and *y_test*.

Data partitioning results in four variables:

X train (for training data), X test (for testing data), y train (ground truths for training), and y test (ground truths for evaluation).

A *20%* test split and *80%* train split are chosen, providing an adequate number of samples for both training and testing. Following this data split based on these variables, machine learning models are constructed, and subsequent model evaluations are conducted.

E. Building models

Scikit-Learn [20]provides various classes and functions to build Machine learning models. Here a total of four models are built and their performance is evaluated.

To construct a Logistic Regression model using the ***LogisticRegression()*** [3] function from the Scikit-learn library, initiate the model by invoking the ***fit()*** method. This involves incorporating the training data and the corresponding target variable. Following this training phase, the model becomes primed for making predictions on new data through the ***predict()*** method, leveraging the acquired patterns from the training process. Logistic Regression, a linear classification algorithm, is particularly useful for binary and multiclass classification tasks, providing an effective approach for modeling and predicting outcomes based on learned patterns.

To build a decision tree model, ***DecisionTreeClassifier()*** [3] function in the Scikit-learn library is used. Decision Tree is a non-linear algorithm that recursively splits the data based on features to make decisions. Here, the criterion is set to Entropy and random_state is set to 42.

To build a K-Nearest Neighbour model, ***KNeighborsClassifier()*** [4] function in the Scikit-learn library is used. K-Nearest Neighbors is a non-parametric algorithm that classifies data points based on the majority class among their k-nearest neighbors. Here, the number of neighbours is 4.

To build a Random Forest Classifier model, ***RandomForestClassifier()*** [4] function in the Scikit-learn library is used. The Random Forest Classifier, a versatile ensemble algorithm, generates multiple decision trees to determine predictions collectively. It simplifies classification tasks by aggregating insights from these trees, ultimately yielding accurate predictions based on the most common outcome observed across the ensemble. Here, the number of estimators is set to 100.

F. Evaluating the Performance

The scikit-learn (sklearn) library, a well-known Python machine learning toolkit, has the *sklearn.metrics* module and *sklearn.model_selection*. These modules offer a number of classes and functions for assessing how well machine learning models work. To calculate the accuracy, recall and precision as well as to display the confusion matrix and classification report, import the

accuracy_score() [20], ***recall_score()*** [21], ***precision_score()*** [22], ***confusion_matrix()*** [23],***ConfusionMatrixDisplay()***, ***cross_val_score()*** [24]functions.

The genuine target values (ground truth) from the dataset and the forecasted target values produced by a machine learning model are the two necessary arguments for the ***confusion_matrix()***.

A minimum of two arguments are needed to construct the ***classification_report()*** [17], which is normally provided and offers an extensive summary of multiple performance indicators for assessing the model's predictions. These arguments are the true target values and the anticipated target values.

The ***Cross validation*** [25] is a pivotal technique that aids in comprehensively evaluating the robustness and generalizability of machine learning models. It involves partitioning the dataset into subsets to validate and train the model iteratively. The method iterates through these partitions, using each subset as both the training and validation set at different stages. By systematically validating the model across diverse data segments, ***cross_val_score()*** [25] assists in uncovering potential issues related to overfitting or underfitting. It serves as a crucial tool in estimating a model's performance on unseen data, contributing significantly to the model selection process.

RESULTS

A. Exploratory Data Analysis

The below fig.1 With the names of the organizations displayed on the y-axis and the number of job listings displayed on the x-axis, the chart offers insights into the recruiting practices of various companies. The employment volumes of the various organizations in the dataset may be quickly compared thanks to this graphic.

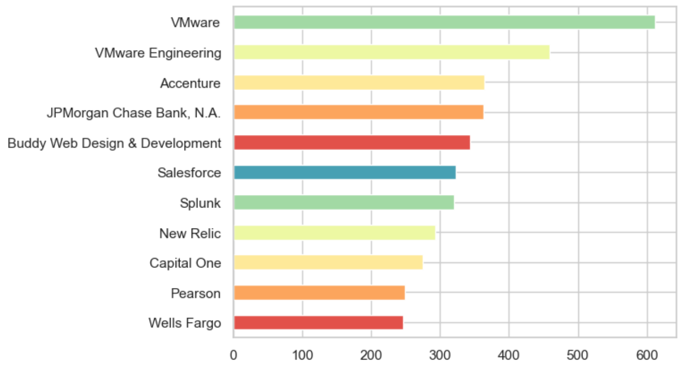

Fig. 1. Plot demonstrating the most hiring companies.

The below fig.2 offers insights into the distribution of top 10 job *titles* within the dataset. Each horizontal bar represents a specific job title, while the length of the bar corresponds to the frequency of that job title occurrence. This visualization aids in identifying the most prevalent job titles, providing a quick overview of the job distribution in the dataset.

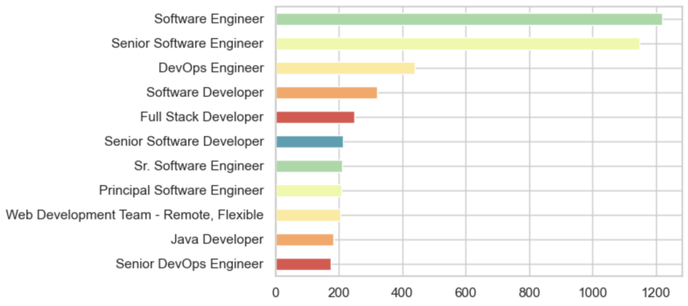

Fig. 2. Histogram based on job titles count.

In the *location* column of the DataFrame, the fig.3 shows the distribution of the top 10 least common locales. Every horizontal bar represents a distinct place, and the bar's length shows how frequently that location occurs. The color scheme is used for decorative effects.

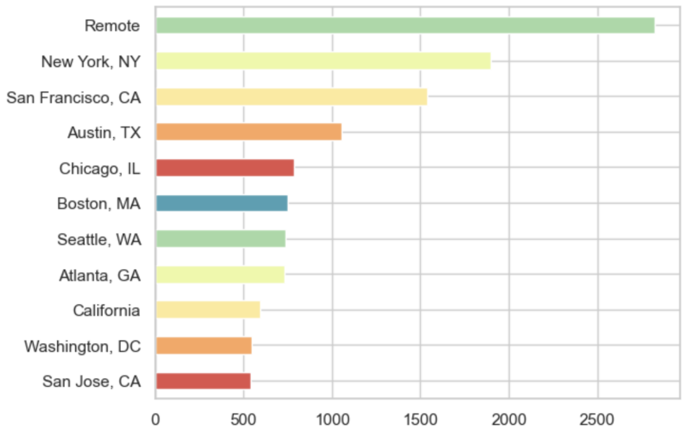

Fig. 3. Histogram based on job locations count.

The percentage distribution of full-time positions with and without sponsorship is shown in the pie chart 4 below. It also has comments to show the numbers of full-time sponsored and non-sponsored positions.

A pie chart that shows the distribution of *sponsored* and *unsponsored* internships is shown in fig.5 below. The proportion of each group is shown by the percentages that are shown in each section. The number of internships with and without funding is shown in the annotations.

The firms that have sponsored the most positions are shown visually in the plot below6. Every horizontal bar signifies a corporation, and the number of sponsored positions that each bar offers is indicated by the bar's length. The x-axis indicates the quantity of sponsored jobs, while the y-axis lists the names of the companies.

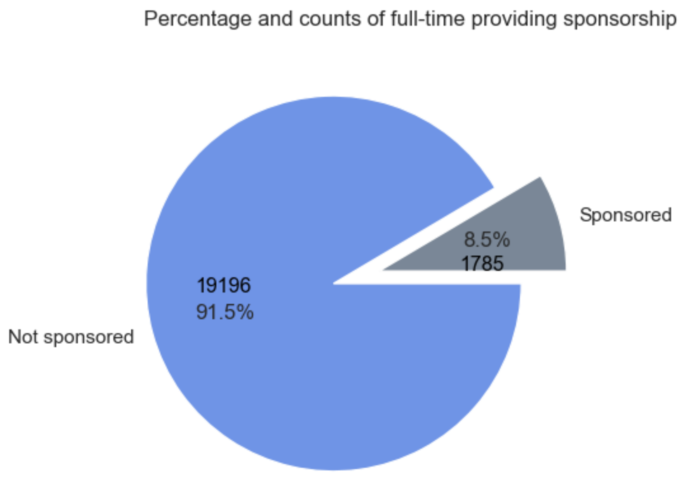

Fig. 4. pie chart demonstrating full-time jobs providing sponsorship.

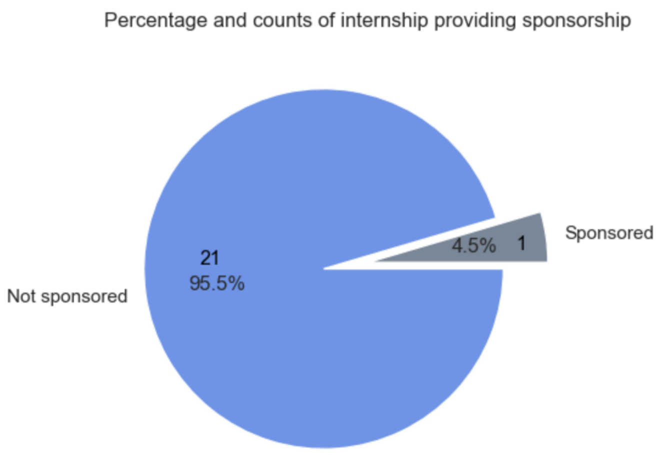

Fig. 5. pie chart demonstrating internships providing sponsorship.

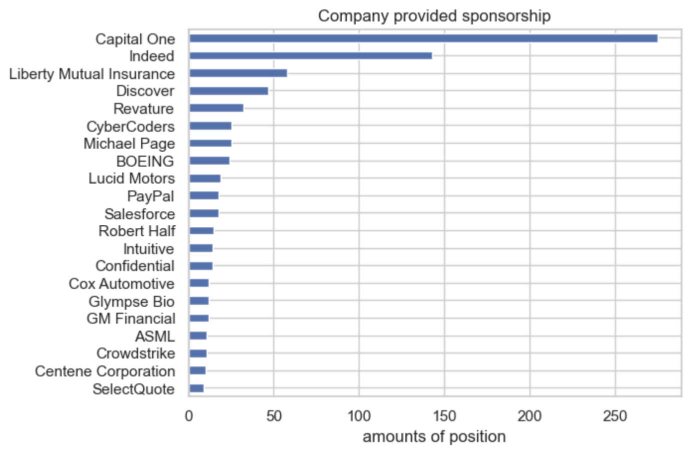

Fig. 6. Histogram demonstrating Companies Providing Sponsorship

B. Comparison based on classification metrics

The performance of various classification models on the US Software Jobs dataset is assessed based on metrics such as accuracy, precision, recall. Random Forest demonstrates robust classification with an accuracy of **93.07%**, showcasing a balanced trade-off between precision **95.58%** and recall **89.32%**. Decision Tree also performed well, achieving an accuracy of **90.6%**, with a precision **75.59%** and recall **89.07%**. KNN, with an accuracy of **90%**, exhibits a precision **67.68%** and recall **61.84%**. Logistic Regression produced a low accuracy of **69.01%**, with a precision of **17.25%** and recall **25.00%**. The variability in model performance highlights the importance of considering specific classification goals and trade-offs when selecting an appropriate model for salary classification.

Model	Accuracy	Recall	Precision
Decision Tree	0.90	0.89	0.75
Logistic Regression	0.69	0.25	0.17
K-Nearest Neighbors	0.90	0.61	0.67
Random Forest	0.93	0.89	0.75

TABLE III
PERFORMANCE OF ALL THE MODEL'S

Random Forest likely outperformed the other classification algorithms due to its ensemble learning approach, which combines multiple decision trees to improve accuracy and generalization. The model's ability to reduce overfitting, handle non-linear relationships, provide feature importance insights, and robustly handle noisy data and imbalanced classes contributed to its superior performance. .

C. Comparison based on cross validation results

The cross-validation results reveal the performance of various models in salary classification. Random Forest exhibits the highest average accuracy at **93.44%**, demonstrating its robust predictive capabilities with low variability **0.0055** standard deviation. Decision Tree achieves a solid accuracy of **90.25%** with low deviation **0.007602**, making it a reliable classifier. The KNN performs has a accuracy of **88.55%** and a standarad deviation of textbf0.011279. The LR has a low average accuracy of **69.11%**. The choice of k in Random Forest likely contributes to its superior accuracy, making it a standout model for salary classification, although other models also exhibit strong and consistent performance.

Model	Average Accuracy	Standard Deviation
Decision Tree	0.90	0.007602
Logistic Regression	0.69	0.000265
K-Nearest Neighbors	0.88	0.011279
Random Forest	0.93	0.005552

TABLE IV
CROSS VALIDATION RESULTS OF ALL THE MODEL'S

The strong performance of Random Forest in cross-validation results can be attributed to its robustness and ability to generalize well across different subsets of the dataset. Cross-validation involves repeatedly splitting the dataset into training and testing sets, ensuring that the model is evaluated on diverse subsets of the data. Random Forest's ensemble approach, which combines multiple decision trees with feature and data randomness, mitigates overfitting and enhances the model's adaptability to various data distributions.

Conclusion

The results of applying classification algorithms to the provided dataset, particularly the superior performance of the Random Forest model, are important for several reasons. Firstly, accurate classification of job-related data, such as salary categories, is crucial for both job seekers and employers. The ability to predict and categorize salaries based on various features allows for more informed decision-making in the job market. The practical significance lies in providing valuable insights to job seekers about potential earnings and aiding employers in efficient hiring processes. The Random Forest's success underscores the importance of ensemble learning and its applicability to complex datasets in the context of job-related information.

In the future, to enhance the effectiveness of this work, several avenues could be explored. Firstly, incorporating additional features or refining existing ones may further improve the models' predictive capabilities. Continuous updates to the dataset would ensure that the models remain relevant and adaptive to changing trends in the job market. Additionally, exploring more advanced techniques, such as deep learning approaches, could be considered to capture intricate patterns that may not be fully captured by traditional machine learning models. Expanding the dataset to include a more diverse set of job listings, industries, or regions could also enhance the models' generalizability. Regular evaluations and retraining of the models would be essential to maintain their accuracy over time. Overall, this work could be extended by embracing evolving technologies and continuously refining the models to better serve the dynamic nature of the job landscape.

References

[1] https://www.kaggle.com/datasets/mexwell/us-software-engineer-jobs
[2] https://en.wikipedia.org/wiki/Logistic_regression
[3] https://en.wikipedia.org/wiki/Decision_tree
[4] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
[5] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[6] https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/
[7] https://colab.research.google.com/?utm_source=scs-index
[8] https://numpy.org/doc/stable/
[9] https://scikit-learn.org/stable/tutorial/index.html
[10] https://pandas.pydata.org/docs/getting_started/index.html
[11] https://matplotlib.org/stable/tutorials/pyplot.html
[12] https://www.geeksforgeeks.org/python-pandas-dataframe/
[13] https://www.geeksforgeeks.org/python-read-csv-using-pandas-read_csv/.
[14] https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html
[15] https://www.w3schools.com/python/pandas/ref_df_isnull.asp
[16] https://en.wikipedia.org/wiki/Word2vec
[17] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop_duplicates.html
[18] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
[19] https://www.geeksforgeeks.org/how-to-split-the-dataset-with-scikit-learns-train_test_split-function/
[20] https://scikit-learn.org/stable/
[21] https://www.javatpoint.com/accuracy_score-in-sklearn
[22] https://en.wikipedia.org/wiki/Precision_and_recall
[23] https://www.w3schools.com/python/python_ml_confusion_matrix.asp
[24] https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f
[25] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html