Advance Data Analytics Algorithm, Machine Learning

Data Science Salary Prediction

Report by – Aryan Bansal

Table of Contents

Abstract:

The goal of this project is to create a machine learning model and create a salary prediction model through a dataset which contains the salaries of Data Scientists of different countries. This has been possible by taking into consideration about different variables like job title, employment type, expertise level, company size, location, etc.

I implemented different types of models like decision tree, random forest and XGBooster, and based on the stats I selected XGBooster as my model for prediction.

Google Colab Link:

https://colab.research.google.com/drive/1ojXN6RJi1uwWXuzWxSXwYGQEguH3XDYG?usp=sharing

ChatGPT Link:

https://chatgpt.com/c/670bbce6-071c-8000-bf6a-64f92521440b

Dataset:

The dataset has been taken from Kaggle and the owner of the dataset is Mr. Sourav Banerjee.

This dataset provides a valuable insight into the trends in salaries for the data scientists, across the globe. Through this dataset, we can know about the salary trend and factors influencing the salaries across different job titles, experience level, geographic location, etc.

The structure of the dataset is as follows:

- Job title – The role of the employee like, Data Analyst, Computer Vision Engineer, ML Engineer, etc.
- Employment type – Nature of Employment (Like: Full-Time, Part-Time or casual)
- Experience Level – Level of Experience (Like: Entry, Mid, intermediate or Senior)
- Expertise Level - Level of Expertise (Like: Intermediate or Advanced)
- Company Size – The size of the company (like: Small, Medium or Large)
- Company Location – Geographic Region of the country (India, US, UK, etc)
- Year – The year that the data represent.

Job Title	Employment Type	Experience Level	Expertise Level	Salary	Salary Currency	Company Location	Salary in USD	Employee Residence	Company Size	Year
Data Scientist	Full-Time	Entry	Junior	120000	Australian Dollar	Australia	83171	Australia	Medium	2022
Computer Vision Engineer	Full-Time	Senior	Expert	102000	Brazilian Real	Brazil	18907	Brazil	Medium	2021
Data Scientist	Full-Time	Mid	Intermediate	50000	British Pound Sterling	United Kingdom	61520	United Kingdom	Medium	2023
Data Analyst	Full-Time	Entry	Junior	40000	British Pound Sterling	United Kingdom	49216	United Kingdom	Medium	2023
Data Science Engineer	Full-Time	Senior	Expert	159500	Canadian Dollar	Canada	127221	Canada	Large	2021
Machine Learning Engineer	Full-Time	Senior	Expert	105000	Euro	Germany	113366	Germany	Large	2023
Data Scientist	Full-Time	Senior	Expert	36000	Euro	Spain	38868	Spain	Medium	2023
Data Engineer	Full-Time	Entry	Junior	57000	Euro	Netherlands	59888	Netherlands	Large	2022
Big Data Engineer	Full-Time	Mid	Intermediate	1672000	Indian Rupee	India	22611	India	Large	2021
Data Scientist	Full-Time	Mid	Intermediate	2500000	Indian Rupee	India	33808	India	Medium	2021

Problem Statement:

Salaries in any field can vary significantly and can be based on multiple factors. To be able to predict what salary a person can seek for or what salary an employer should provide to its employee could be beneficial to both of them so as to negotiate and to create a fair deal. For employers, setting competitive salaries ensures they can attract and retain talent in a competitive job market.

Project Purpose:

The purpose of this project is to: -

- Accurately predict the Employee salaries for various job roles throughout the ML Model.
- Assisting job seekers and employers in finding the trend and expectations across different job roles.
- Identifying factors affecting the salary.

Motivation:

The motivation behind choosing this project was to address a real-life solution as it used to concern me a lot.

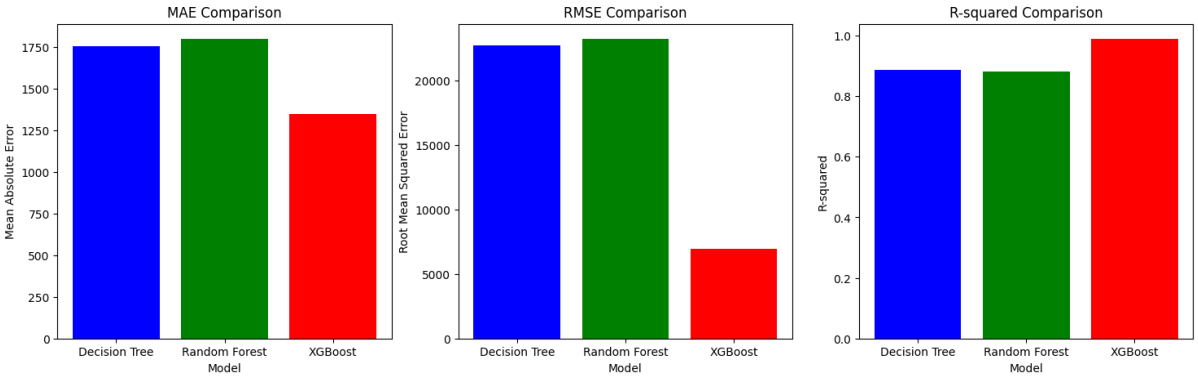
This project aims to bridge the gap between the job seekers and the employers by giving them the market expected salary trend vs the realistic insights for both sides.

Model Overview:

I tested 3 different models to have a better understanding of the predicting model in order to get the most accurate result. The three models that I used were Decision Tree, Random Forest and XGBooster.

Out of the three models, you'll only see XGBooster working and not the others. I trained those models, got my answers and then worked upon creating the predicting model on the best suited.

Providing with the evidence to it, I created a bar graph showing MAE, RMSE and R^2 for all the three models and comparing the results.



From the bar graphs, the MAE and RMSE for XGBooster is lowest among the three and the R^2 is the highest. Since, XGBooster was very compelling and aligning with the dataset, I chose to go with XGBooster.

Overview on XGBoost:

XGBoost stands for Extreme Gradient Boosting, a library proposed by some researchers from University of Washington. It is an optimized distributed gradient boosting library designed for efficient and scalable training of the ML Models. It provides a parallel tree boosting and is one if the most efficient Machine Learning library.

It's key feature is to handling the missing values efficiently, which allows it to handle the real-world data as it is likely to be inconsistent and have irregularities.

Main Code:

```
# Load the dataset

from google.colab import drive
drive.mount('/content/drive')

dataset_path = '/content/drive/MyDrive/v7_Latest_Data_Science_Salaries.csv'
df = pd.read_csv(dataset_path)
```

Uploading the dataset which is taken from Kaggle and mounting the drive so that Google Colab can access the files.

Dataset_path is the command which will allow the Colab to access and find the datasheet saved in the Drive.

Data Preprocessing:

```
# Identify categorical and numerical columns
categorical_columns = ['Job Title', 'Employment Type', 'Experience Level',
                       'Expertise Level', 'Salary Currency', 'Company Location',
                       'Employee Residence', 'Company Size']
numerical_columns = ['Year']

# Separate features (X) and target (Y)
X = df.drop('Salary in USD', axis=1)
y = df['Salary in USD']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature Engineering

# One-hot encode categorical features
encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
X_train_encoded = encoder.fit_transform(X_train[categorical_columns])
X_test_encoded = encoder.transform(X_test[categorical_columns])
```

Machine Learning model needs numerical input because of which categorical variables like Job Title, company location has been changed to numerical format. This has been done through categorical and numerical columns:

Categorical Columns: These are columns with non-numeric values, such as job titles, employment types, or company locations. Since machine learning models cannot work with text directly, we need to encode these categorical values into numeric format.

Numerical Columns: These are columns with numeric values, such as the "Year". These can be used directly by machine learning models without encoding

Features (X): This is the dataset without the target column (Salary in USD). It contains all the relevant information, such as job title, employment type, experience level, etc., that the model will use to predict salaries

Target (y): This is the "Salary in USD" column, which we want the model to predict

The data is split to ensure the model is trained on one part (80%) and tested on another part (20%). This helps evaluate the model's performance on unseen data. Reproducibility is ensured by setting random_state=42.

train_test_split(): This function splits the data into training and testing sets.

Training Set (80%): Used to train the machine learning model.

Testing Set (20%): Used to evaluate the performance of the trained model on unseen data.

test_size=0.2: This indicates that 20% of the data is reserved for testing.

random_state=42: This ensures the split is reproducible—the same data will be used for training and testing every time you run the code

One-hot encoding converts categorical features into a binary format (0 or 1) to make them suitable for machine learning. **handle_unknown='ignore'** ensures that unseen categories in the test set won't cause errors. The training set is used to fit the encoder, and the same encoding is applied to the test set for consistency.

OneHotEncoder: This converts categorical values (like "Job Title" or "Company Location") into binary numerical values (0s and 1s). Each category becomes a separate column with a 1 or 0 indicating its presence.

handle_unknown='ignore': This ensures that any unknown category in the test data (not seen during training) will be ignored, preventing errors.

All these codes have been performed so as to ensure the data is properly structured for machine learning models. Splitting the data into training and test sets prevents overfitting and allows us to evaluate the model on unseen data and encoding categorical variables makes them compatible with the machine learning algorithms, ensuring the model can use all relevant features to predict salaries.

sparse_output=False: This makes sure the output is in a dense matrix format (rather than sparse), which is easier to work with in further steps.

fit_transform(): This learns the encoding from the training data and applies it to transform the categorical columns into numeric format.

transform(): This applies the learned encoding from the training data to the test data to ensure consistency between both datasets.

Model Training and Evaluation Process

```
# Helper Function to Evaluate Model Performance
def evaluate_model(model_name, y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred, squared=False)
    r2 = r2_score(y_true, y_pred)
    print(f"--- {model_name} ---")
    print(f"Mean Absolute Error (MAE): {mae:.2f}")
    print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
    print(f"R-squared (R²): {r2:.4f}")
    print("-" * 20)

# Model Training
xgboost_model = xgb.XGBRegressor(random_state=42)
xgboost_model.fit(X_train_processed.values, y_train)
```

This function evaluates how well the model has predicted salaries by calculating key error metrics. This step is essential to understand the accuracy and reliability of the model's predictions

Evaluating Performance: The evaluate_model() function calculates metrics (MAE, RMSE, R^2) to determine the accuracy and reliability of the predictions.

The three metrics that are being used, that is:

- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)
- R-Squared (R^2)

Evaluating ensures you can compare different models (like Decision Tree, Random Forest, and XGBoost) using consistent metrics.

Helps you choose the best-performing model for salary prediction.

These steps were crucial in the machine learning pipeline as it ensures the model is both trained effectively and evaluated consistently, allowing you to compare it with other models and select the best one

Data Preprocessing and Scaling

```python
# Function to Preprocess User Input
def preprocess_user_input(user_data, encoder, scaler, full_feature_list):
    """
    Preprocesses user input to match the encoded format used during model training.
    """
    # Get categorical features present in user_data
    categorical_features_present = [col for col in categorical_columns if col in user_data.columns]

    # Create a DataFrame with all categorical columns, filling missing ones with a default value
    user_data_categorical = pd.DataFrame(columns=categorical_columns)
    for col in categorical_columns:
        if col in user_data.columns:
            user_data_categorical[col] = user_data[col]
        else:
            # Fill missing categorical columns with a placeholder (e.g., 'Unknown')
            user_data_categorical[col] = ['Unknown']

    # Apply one-hot encoding
    user_data_encoded = encoder.transform(user_data_categorical)

    # Scale numerical features
    user_data_scaled = scaler.transform(user_data[numerical_columns])

    # Combine encoded categorical features and scaled numerical features
    user_data_processed = pd.concat([pd.DataFrame(user_data_encoded), pd.DataFrame(user_data_scaled)], axis=1)

    # Ensure all columns from the original training set are present
    user_data_processed = user_data_processed.reindex(columns=full_feature_list, fill_value=0)

    return user_data_processed
```

This code is used to convert raw user input into the same format used during training to ensure consistency between the user input and the trained model, preventing errors during prediction.

Key Points:

Handles missing columns: If some categorical features are missing from the input, they are filled with a placeholder (e.g., 'Unknown').

Applies one-hot encoding: Converts categorical features into a binary format.

Scales numerical features: Ensures numerical inputs are on the same scale as the training data.

Aligns with the training set: Ensures all columns match the original feature set used during model training.

User_data_categorical and pd.dataframe has been used to create a data frame for categorical columns, ensuring all necessary columns are included. Filling missing columns with a default value, such as 'Unknown', to prevent encoding errors.

This code has been performed as some columns may be missing from the user input, so the function ensures consistency with the original feature set used during training.

Prediction and Visualization (Test vs Training)

```
# Function to Predict Salary
def predict_salary(model, user_data_processed):
    """
    Uses the XGBoost model to predict the salary based on user input.
    """
    predicted_salary = model.predict(user_data_processed)
    return predicted_salary[0]

#Creating a Scatter plot

import matplotlib.pyplot as plt

# Ensure X_test_processed is properly formatted and passed
y_pred_test = xgboost_model.predict(X_test_processed.values)  # Predict salaries

# 1. Actual vs Predicted Salary (Scatter Plot)
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_test, alpha=0.5, color='b')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.xlabel("Actual Salary")
plt.ylabel("Predicted Salary")
plt.title("Actual vs Predicted Salary (Scatter Plot)")
plt.show()
```

The below function takes the trained model and processed user input to predict the salary.

Model.predict(): This method uses the XGBoost model to generate predictions based on the input data.

Return predicted_salary[0]: Returns the predicted salary value

It is important to run this function as this function allows the trained model to make real-time predictions for new user input, simulating how the model would perform in production. It also also returns the first element ensures that a single prediction is extracted from the returned array (XGBoost typically returns an array of predictions).

Scatter Plot:

I have also made a scatter plot using importing the matplotlib and using the test data so that it can show a trend line between the predictive values and the actual values. Scatter plot was important as it helps in visualizing how closely the predicted salaries match the actual salaries.

By adding the reference line to it, it provides a baseline for comparison-points close to the line indicate better predictions, while points far from the line indicate prediction errors.

Model Evaluation: This plot allows you to identify trends and errors (e.g., if the model underpredicts or overpredicts salaries).

Predictive Model

```
# Main Program to Run the Interactive Prediction Model
if __name__ == "__main__":
    # Get the full feature list from the training data
    full_feature_list = X_train_processed.columns

    print("Welcome to the Salary Prediction Model!")

    while True:
        # Collect User Input
        job_title = input("Enter the job title (e.g., 'Data Scientist'): ").strip()
        employment_type = input("Enter the employment type (e.g., 'Full-Time'): ").strip()
        experience_level = input("Enter the experience level (e.g., 'Senior'): ").strip()
        expertise_level = input("Enter the expertise level (e.g., 'Expert'): ").strip()
        company_size = input("Enter the company size (e.g., 'Large'): ").strip()
        company_location = input("Enter the company location (e.g., 'United States'): ").strip()
        year = int(input("Enter the year (e.g., 2024): "))
```

This code is part of the main program that runs the interactive salary prediction model. The user inputs job-related information, and the program processes it to predict a salary using the trained model

This code belongs to the User Interaction and Prediction Input Phase. Specifically:

- **Input Collection:** Gathers job-related details from the user in real time.
- **Feature Alignment:** Ensures the input matches the feature structure from the training data.
- **Interactive Execution:** Allows multiple predictions through a continuous loop

The code collects real-time input from the user to make a salary prediction. The interactive loop ensures that multiple predictions can be made without restarting the program. It also ensures the user input aligns with the feature set used during model training, reducing errors

```
# Create a DataFrame from user input
user_data = pd.DataFrame({
    'Job Title': [job_title],
    'Employment Type': [employment_type],
    'Experience Level': [experience_level],
    'Expertise Level': [expertise_level],
    'Company Size': [company_size],
    'Company Location': [company_location],
    'Year': [year]
})
```

This code creates a data frame from the user input collected in the previous step. A data frame is a tabular data structure used in Pandas.

pd.DataFrame(): Creates a pandas data frame from a dictionary of lists. Each key (e.g., 'Job Title', 'Year') becomes a column name, and the corresponding list values become the data in that column.

The purpose of this step is to structure the user input in a format that matches the input format of the machine learning model.

> Machine learning models, especially those trained using Pandas data frames, expect inputs in a data frame format.
>
> The column names must align with the feature names from the training data to avoid errors during prediction.

```
# Preprocess User Input
user_data_processed = preprocess_user_input(user_data, encoder, scaler, full_feature_list)

# Predict Salary
predicted_salary = predict_salary(xgboost_model, user_data_processed.values)

print(f"\nPredicted Salary: ${predicted_salary:.2f}")

another_prediction = input("Do you want to make another prediction (yes/no)? ").lower()
if another_prediction != 'yes':
    break
```

This code handles the salary prediction model's preparation, prediction, and user interaction. First, the user input is pre-processed with the preprocess_user_input() function, which guarantees that the input data matches the training data by applying one-hot encoding, scaling numerical features, and ordering the columns to fit the model's expectations. The processed input is then fed into the trained XGBoost model, which produces a salary forecast that is shown to the user in a structured output. After displaying the expected income, the application allows the user to make another forecast using an interactive loop. If the user decides not to continue, the loop breaks and the software terminates. This framework guarantees that the prediction process is accurate, streamlined, and user-friendly.

Conclusion:

In this project, we successfully created a salary prediction model utilizing the XGBoost, Random Forest, and Decision Tree algorithms. After evaluating their results, XGBoost was chosen as the final model because to its higher accuracy and capacity to handle complicated data interactions. The algorithm estimates salary based on a number of critical parameters,

including job title, employment type, experience level, firm size, region, and year. We verified that the inputs met the model's criteria by carefully preparing both training and user input data, resulting in solid predictions

This project delivers significant information for both job seekers and businesses by assisting users in understanding salary expectations based on various positions and qualities. The model's dynamic nature enables users to enter real-time data and investigate wage patterns, making it a useful tool for reviewing remuneration. Overall, the study highlights the relevance of data preparation, model validation, and feature engineering in developing an efficient machine learning solution, as well as the predictive model's potential in real-world settings. This model may be improved further by managing more complicated feature interactions and adding external data sources, allowing it to make even more exact predictions

References:

1. *What is XGBoost?* (n.d.). NVIDIA Data Science Glossary. https://www.nvidia.com/en-au/glossary/xgboost/

2. GeeksforGeeks. (2021, September 18). *XGBoost*. GeeksforGeeks. https://www.geeksforgeeks.org/xgboost/

3. *Latest Data Science Salaries*. (n.d.). Www.kaggle.com. https://www.kaggle.com/datasets/iamsouravbanerjee/data-science-salaries-2023/data

4. *Python Package Introduction — xgboost 1.5.2 documentation*. (n.d.). Xgboost.readthedocs.io. https://xgboost.readthedocs.io/en/stable/python/python_intro.html