**Salary Prediction in the AI Job Market**

**Team 8**

**Yaswanth Ganapathi & Siva Pradeep Jarugumalli**

**MA 5790 December 2024**

**Abstract:**

The rising demand for artificial intelligence (AI) professionals has prompted industries to offer competitive salaries to secure top talent. This study investigates salary prediction within the AI job market, leveraging a dataset of 500 observations and 10 diverse variables. These include job-specific attributes such as role, industry, and experience level, as well as contextual data like company size and location.

By employing advanced machine learning technique both linear and non-linear model the study identifies key salary determinants and evaluates model performance using metrics like RMSE, R², and MAE. Among the models tested, Neural Networks emerged as the top performer, demonstrating superior capability in capturing intricate patterns within the data. This research provides actionable insights for employers, job seekers, and policymakers in navigating the evolving AI job market.



**1. Background:**

The artificial intelligence (AI) job market has witnessed exponential growth in recent years, driven by rapid advancements in machine learning, robotics, and automation. With organizations across industries integrating AI into their operations, the demand for skilled professionals has surged. Salaries in this domain vary significantly based on factors such as job role, industry, experience level, and geographic location, making salary prediction a complex yet crucial task.

This project seeks to address the need for data-driven insights into salary structures within the AI job market. The dataset used in this study encapsulates diverse variables such as **Job Title**, **Industry**, and **Experience Level**, reflecting the multifaceted nature of salary determination. By leveraging statistical and machine learning models, this research aims to uncover patterns and trends that influence salary outcomes.

Understanding these dynamics can benefit various stakeholders:

* **Employers** can use the insights to design competitive compensation packages.
* **Job seekers** can make informed decisions about career paths and negotiate better salaries.
* **Policymakers and analysts** can gain a deeper understanding of the evolving job market in AI to shape policies and predict future trends.

This study not only contributes to academic research but also holds practical implications for recruitment and workforce development in the AI sector.

**2. Variable Introduction and Definitions:**

**Goal of Study:**

The goal of this study is to predict salaries within the Artificial Intelligence (AI) job market using a combination of linear and non-linear machine learning models. By leveraging a dataset containing job-specific attributes (e.g., role, industry, experience level) and contextual data (e.g., company size, location), the study aims to:

1. Identify the key factors that influence salaries in the AI domain.
2. Develop predictive models capable of accurately estimating salaries based on the provided data.
3. Evaluate and compare the performance of various models to determine the most effective approach for salary prediction.
4. Provide actionable insights to employers, job seekers, and policymakers to aid in designing competitive compensation packages, making informed career decisions, and understanding market trends.

This study aims to bridge the gap between data-driven analytics and practical decision-making in the rapidly evolving AI job market.

The dataset consists of ten variables, categorized into predictors and the target variable. The target variable, **Salary (USD)**, represents annual compensation for roles within the AI sector. The predictors are divided into categorical variables, such as **Job Title** and **Industry**, which provide contextual data for salary prediction.

**Predictors and Their Descriptions:**

1. **Job Title**: Titles associated with various roles in AI (e.g., Data Scientist, AI Engineer).
2. **Industry**: Sectors employing AI professionals (e.g., Tech, Healthcare).
3. **Experience Level**: Categorization of job roles based on experience requirements (e.g., Entry-Level, Mid-Level).
4. **Location**: Geographical regions of employment.
5. **Education**: Minimum educational qualifications (e.g., Bachelor’s, Master’s).
6. **Company Size**: Size of the employing company (e.g., Small, Medium, Large).
7. **Remote Work**: Proportion of work performed remotely.
8. **Job Posting Date**: Period when the job was posted.
9. **Hiring Status**: Current hiring stage (e.g., Open, Closed).

The relationships between these predictors and salary will be analyzed to provide actionable insights

**3. Preprocessing Steps:**

Data preprocessing is a critical step in predictive modeling, ensuring the dataset is prepared for analysis and model training. For this project, the following preprocessing steps were undertaken to handle the dataset effectively:

**a. Handling Numerical and Categorical Variables:**

* The dataset contains one numerical variable (**Salary**) and nine categorical variables (e.g., **Job Title**, **Industry**, **Experience Level**).
* Categorical variables were encoded using **One-Hot Encoding**, transforming them into binary format for model compatibility.

**b. Addressing Skewness:**

* The **Salary** variable exhibited non-normal distribution, which was addressed using a **Box-Cox Transformation**. This transformation helped normalize the data, reducing skewness and stabilizing variance.
* Skewness before Box-Cox Transformation: -0.1592337.
* Skewness after Box-Cox Transformation: -0.01181303.

**A comparison of different colored bars

Description automatically generated**

**c. Near Zero Variance Features:**

* Features with little variance were identified and removed to avoid redundancy and reduce computational overhead, improving the efficiency and accuracy of the models.
* This step eliminates predictors that contribute minimal information to the model.

**d. Feature Scaling:**

* All numerical variables were standardized (centered and scaled) to ensure equal weighting across features, preventing features with larger scales from dominating the learning process.
* Standardization promotes better convergence and comparability in model training.

**4. Data Splitting:**

The dataset was divided into training and testing subsets to evaluate the predictive models effectively and ensure their generalizability. Key aspects of data splitting include:

1. **Purpose of Splitting**: Dividing the data ensures that the model learns patterns from one portion of the data (training set) while another portion (testing set) is kept unseen to evaluate its performance on new data. This mimics real-world scenarios where models predict unseen data.
2. **Training and Testing Proportions**: The dataset was split into **80% for training** and **20% for testing**. This ratio is a common standard in machine learning to balance the amount of data used for model learning and performance evaluation.

A blue and orange squares

Description automatically generated

1. **Stratified Sampling**: For the target variable (Salary\_BoxCox), stratified sampling was applied to ensure that the distribution of salary values was consistent across both subsets. This prevents bias in evaluation and ensures the testing set is representative of the overall data.
2. **Importance of Data Splitting**: A proper split ensures the testing set provides an unbiased evaluation of the model. The training set is used for model development and hyperparameter tuning, while the testing set helps assess the model's performance on unseen data.
3. **10-Fold Cross-Validation on Training Set**: During model training, 10-fold cross-validation was performed on the training set. This divides the training data into 10 parts, using 9 parts for training and 1 part for validation iteratively. It enhances model reliability, reduces overfitting, and ensures robust performance.

**5. Model Fitting:**

The training process aimed to fit both linear and non-linear models to the transformed salary (Salary\_BoxCox). Below is a detailed explanation of how the models performed, their characteristics, and insights into their applicability:

1. **Linear Models:**

Linear models are based on the assumption of a linear relationship between predictors and the target variable. These models are simple, interpretable, and computationally efficient but may struggle in the presence of non-linear patterns or interactions among features.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Models** | **Training** | | **Testing** | |
| **RMSE** | **R Square** | **RMSE** | **R Square** |
| Linear | 214051.6 | 0.04843276 | 193136.1 | -0.0768572 |
| Ridge | 212937.2 | 0.05421494 | 192579.9 | -0.07066369 |
| LASSO | 204852.6 | 0.05349881 | 184410.8 | 0.01824381 |
| E-Net | 201961.8 | 0.02203424 | 186281.6 | -0.001776928 |

1. **Nonlinear Models:**

Non-linear models excel in capturing intricate relationships and interactions but require careful tuning and often higher computational resources.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Models** | **Training** | | **Testing** | |
| **RMSE** | **R Square** | **RMSE** | **R Square** |
| NN | 200848.4 | 0.038053557 | 186613.2 | 0.004154275 |
| MARS | 202500.6 | 0.02697629 | 192579.9 | -0.005346685 |
| SVM | 203572.4 | 0.02680169 | 189918.4 | -0.04127449 |
| KNN | 201916.5 | 0.05712938 | 188620.6 | -0.02709186 |

**6. Conclusion:**

The analysis compared the performance of linear and non-linear models for predicting the transformed salary (Salary\_BoxCox). Among linear models, **Lasso Regression** emerged as the best performer due to its ability to select relevant features, reduce overfitting, and achieve a lower RMSE (184,410.8 on testing) and better Rsquare (0.0182 on testing) compared to other linear approaches like Ridge Regression and Elastic Net. However, all linear models struggled to capture the non-linear relationships in the data. Non-linear models demonstrated better adaptability, with **Neural Networks** showing the best overall performance due to their ability to model complex patterns, achieving an RMSE of 185,729.3 and Rsquare of 0.0042 on testing. Other non-linear models like MARS and SVM underperformed due to limitations in generalizing to unseen data. In conclusion, Lasso Regression is recommended for simplicity and interpretability, while Neural Networks are preferred for capturing complex relationships, provided computational resources are available. Further exploration with ensemble methods and addressing data limitations may enhance model performance.

**Appendix 1: Linear Models tuning Plots**

1. **Ridge Regression:**

**A graph with a line

Description automatically generated**

1. **Lasso Regression:**

**A graph with a line

Description automatically generated**

1. **Elastic Net:**

**A graph of different colored lines

Description automatically generated**

**Appendix 2: Non-Linear Models Tuning Plot**

1. **Neural Networks:**

**A graph with numbers and lines

Description automatically generated**

1. **Multivariate Adaptive Regression Splines (MARS)**

**A graph with a line going up

Description automatically generated**

1. **Support Vector Machine (SVM)**

**A graph with a line

Description automatically generated**

1. **K-Nearest Neighbors (KNN)**

**A graph with a line going up

Description automatically generated**

**Appendix 3: Important Predictors for the best model**

1. **Best Model in Linear : Lasso**

**A graph with blue dots

Description automatically generated**

1. **Best Model in NonLinear: Neural Networks**

**A graph with blue dots

Description automatically generated**

**R Code:**

# Load necessary libraries

library(dplyr)

library(caret)

library(ggplot2)

library(glmnet)

library(earth)

library(kernlab)

library(nnet)

library(gridExtra)

library(e1071)

# Load the dataset (replace 'file\_path' with the actual file path)

data <- read.csv('ai\_job\_market\_insights.csv')

# Preprocessing: Handle missing values

data <- na.omit(data) # Remove rows with missing values

# Print skewness before transformation

cat("Skewness before Box-Cox Transformation:", skewness(data$Salary\_USD), "\n")

# Apply Box-Cox Transformation to target variable (Salary\_USD)

if (any(data$Salary\_USD <= 0)) {

data$Salary\_USD <- data$Salary\_USD + abs(min(data$Salary\_USD)) + 1

}

boxcox\_transform <- BoxCoxTrans(data$Salary\_USD)

data$Salary\_BoxCox <- predict(boxcox\_transform, data$Salary\_USD)

# Print skewness after transformation

cat("Skewness after Box-Cox Transformation:", skewness(data$Salary\_BoxCox), "\n")

# Visualize Box-Cox transformation

par(mfrow = c(1, 2))

hist(data$Salary\_USD, breaks = 30, col = "skyblue", main = "Before Box-Cox Transformation", xlab = "Salary (USD)")

hist(data$Salary\_BoxCox, breaks = 30, col = "orange", main = "After Box-Cox Transformation", xlab = "Salary (Box-Cox Transformed)")

# Convert categorical variables to factors and apply dummy encoding

data <- data %>%

mutate(across(where(is.character), as.factor)) %>%

model.matrix(~ . - 1, data = .) %>%

as.data.frame()

# Near Zero Variance (NZV) Check

nzv <- nearZeroVar(data, saveMetrics = TRUE)

data <- data[, !nzv$nzv]

# Split the data into training (80%) and testing (20%) sets

set.seed(123)

trainIndex <- createDataPartition(data$Salary\_BoxCox, p = 0.8, list = FALSE)

trainData <- data[trainIndex, ]

testData <- data[-trainIndex, ]

# Calculate counts for training and testing sets

split\_counts <- c(Training = nrow(trainData), Testing = nrow(testData))

# Create a bar plot

barplot(

split\_counts,

col = c("blue", "orange"),

main = "Distribution of Training and Testing Data",

xlab = "Data Subsets",

ylab = "Number of Observations",

ylim = c(0, max(split\_counts) + 50)

)

# Preprocessing: Centering and scaling predictors

preProc <- preProcess(trainData[, -which(names(trainData) %in% c("Salary\_USD", "Salary\_BoxCox"))], method = c("center", "scale"))

trainData\_scaled <- predict(preProc, trainData)

testData\_scaled <- predict(preProc, testData)

# Define training control for cross-validation

train\_control <- trainControl(method = "cv", number = 10)

# Function to evaluate models on testing set and print results

evaluate\_and\_print <- function(model, testData, model\_name) {

predictions <- predict(model, newdata = testData)

actual <- testData$Salary\_BoxCox

rmse <- sqrt(mean((predictions - actual)^2))

r2 <- 1 - sum((predictions - actual)^2) / sum((actual - mean(actual))^2)

mae <- mean(abs(predictions - actual))

cat(model\_name, "Testing Results:\n")

cat("RMSE:", rmse, "\n")

cat("R-squared:", r2, "\n")

cat("MAE:", mae, "\n\n")

}

# Linear Models

# 1. Linear Regression

set.seed(123)

model\_lm <- train(Salary\_BoxCox ~ . - Salary\_USD, data = trainData\_scaled, method = "lm", trControl = train\_control)

cat("Linear Regression Training Results:\n")

print(model\_lm)

evaluate\_and\_print(model\_lm, testData\_scaled, "Linear Regression")

# 2. Ridge Regression

set.seed(123)

model\_ridge <- train(Salary\_BoxCox ~ . - Salary\_USD, data = trainData\_scaled, method = "ridge", trControl = train\_control, tuneLength = 15)

cat("Ridge Regression Training Results:\n")

print(model\_ridge)

plot(model\_ridge)

evaluate\_and\_print(model\_ridge, testData\_scaled, "Ridge Regression")

# 3. Lasso Regression

set.seed(123)

model\_lasso <- train(Salary\_BoxCox ~ . - Salary\_USD, data = trainData\_scaled, method = "lasso", trControl = train\_control, tuneLength = 15)

cat("Lasso Regression Training Results:\n")

print(model\_lasso)

plot(model\_lasso)

evaluate\_and\_print(model\_lasso, testData\_scaled, "Lasso Regression")

# 4. Elastic Net

set.seed(123)

model\_enet <- train(Salary\_BoxCox ~ . - Salary\_USD, data = trainData\_scaled, method = "glmnet", trControl = train\_control, tuneLength = 15)

cat("Elastic Net Training Results:\n")

print(model\_enet)

plot(model\_enet)

evaluate\_and\_print(model\_enet, testData\_scaled, "Elastic Net")

# Non-Linear Models

# 1. Neural Networks

set.seed(123)

model\_nn <- train(Salary\_BoxCox ~ . - Salary\_USD, data = trainData\_scaled, method = "nnet", trControl = train\_control, tuneLength = 15, linout = TRUE, trace = FALSE)

cat("Neural Networks Training Results:\n")

print(model\_nn)

plot(model\_nn)

evaluate\_and\_print(model\_nn, testData\_scaled, "Neural Networks")

# 2. Multivariate Adaptive Regression Splines (MARS)

set.seed(123)

model\_mars <- train(Salary\_BoxCox ~ . - Salary\_USD, data = trainData\_scaled, method = "earth", trControl = train\_control, tuneLength = 15)

cat("MARS Training Results:\n")

print(model\_mars)

plot(model\_mars)

evaluate\_and\_print(model\_mars, testData\_scaled, "MARS")

# 3. Support Vector Machines (SVM)

set.seed(123)

model\_svm <- train(Salary\_BoxCox ~ . - Salary\_USD, data = trainData\_scaled, method = "svmRadial", trControl = train\_control, tuneLength = 15)

cat("SVM Training Results:\n")

print(model\_svm)

plot(model\_svm)

evaluate\_and\_print(model\_svm, testData\_scaled, "SVM")

# 4. K-Nearest Neighbors (KNN)

set.seed(123)

model\_knn <- train(Salary\_BoxCox ~ . - Salary\_USD, data = trainData\_scaled, method = "knn", trControl = train\_control, tuneLength = 15)

cat("KNN Training Results:\n")

print(model\_knn)

plot(model\_knn)

evaluate\_and\_print(model\_knn, testData\_scaled, "KNN")

Imp\_var1 <- varImp(model\_lasso)

plot(Imp\_var1)

Imp\_var2 <- varImp(model\_nn, )

plot(Imp\_var2)