Logistic Regression

Predicting AI Expectation in Job Roles

Kelly, Sabrina, Makenzie

October 15, 2025

from pyspark.sql import SparkSession, Row  
from pyspark.sql.functions import col, lit, concat\_ws, lower, regexp\_replace, when, trim, mean as \_mean  
from pyspark.sql.types import DoubleType  
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler, MinMaxScaler  
from pyspark.ml.classification import LogisticRegression  
from pyspark.ml.evaluation import BinaryClassificationEvaluator  
from pyspark.ml import Pipeline  
from pyspark.ml.functions import vector\_to\_array  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
# --- Start Spark ---  
spark = SparkSession.builder.appName("AI\_Job\_Comparison").getOrCreate()  
  
# --- Load data ---  
df = (spark.read.option("header", "true")  
 .option("inferSchema", "true")  
 .option("multiLine", "true")  
 .option("escape", "\"")  
 .csv("data/lightcast\_job\_postings.csv"))  
  
# --- Create text features for AI labeling ---  
text\_cols = ["TITLE\_RAW", "TITLE\_CLEAN", "BODY", "SKILLS\_NAME",  
 "COMMON\_SKILLS\_NAME", "SPECIALIZED\_SKILLS\_NAME",   
 "SOFTWARE\_SKILLS\_NAME"]  
  
df = df.fillna("")  
df = df.withColumn("text\_features",  
 concat\_ws(" ", \*[lower(regexp\_replace(col(c), r"[^a-zA-Z0-9 ]", "")) for c in text\_cols]))  
  
# --- AI job label ---  
ai\_keywords = [  
 r"\bAI\b", r"\bML\b", r"\bLLM\b", r"\bNLP\b",  
 "artificial intelligence", "machine learning", "deep learning",  
 "computer vision", "generative", "gen ai", "chatgpt", r"gpt-\d+",  
 "transformer", "bert", "prompt engineer", "reinforcement learning"  
]  
ai\_pattern = "|".join([f"(?i){k}" for k in ai\_keywords])  
df = df.withColumn("requires\_ai", when(col("text\_features").rlike(ai\_pattern), lit(1)).otherwise(lit(0)))  
  
# --- Create average salary ---  
df = df.withColumn(  
 "SALARY\_AVG",  
 when(col("SALARY") > 0, col("SALARY"))  
 .when(col("SALARY\_FROM").isNotNull() & col("SALARY\_TO").isNotNull(),  
 (col("SALARY\_FROM") + col("SALARY\_TO")) / 2)  
 .when(col("SALARY\_FROM").isNotNull(), col("SALARY\_FROM"))  
 .when(col("SALARY\_TO").isNotNull(), col("SALARY\_TO"))  
 .otherwise(None)  
)  
  
# --- Handle education level ---  
df = df.withColumn("MIN\_EDULEVELS", when(col("MIN\_EDULEVELS") == 99, 0).otherwise(col("MIN\_EDULEVELS")))  
  
# --- Select relevant features ---  
model\_df = df.select(  
 "requires\_ai",  
 "SALARY\_AVG",  
 "MIN\_YEARS\_EXPERIENCE",  
 "MIN\_EDULEVELS",  
 "STATE\_NAME",  
 "REMOTE\_TYPE"  
)

25/10/15 01:02:15 WARN Utils: Your hostname, Sabrinas-MacBook-Pro.local resolves to a loopback address: 127.0.0.1; using 10.0.0.170 instead (on interface en0)  
25/10/15 01:02:15 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
25/10/15 01:02:16 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
[Stage 0:> (0 + 1) / 1] [Stage 1:> (0 + 1) / 1]

# --- Handle numeric nulls ---  
numeric\_cols = ["SALARY\_AVG", "MIN\_YEARS\_EXPERIENCE", "MIN\_EDULEVELS"]  
for c in numeric\_cols:  
 model\_df = model\_df.withColumn(c, col(c).cast(DoubleType()))  
 mean\_val = model\_df.select(\_mean(col(c))).first()[0]  
 model\_df = model\_df.na.fill({c: mean\_val})  
  
# --- Handle categorical nulls ---  
categorical\_cols = ["STATE\_NAME", "REMOTE\_TYPE"]  
model\_df = model\_df.filter(  
 (trim(col("STATE\_NAME")) != "") &  
 (trim(col("REMOTE\_TYPE")) != "") &  
 col("STATE\_NAME").isNotNull() &  
 col("REMOTE\_TYPE").isNotNull()  
)  
  
# --- Encode categorical variables ---  
indexers = [StringIndexer(inputCol=c, outputCol=f"{c}\_IDX", handleInvalid="keep") for c in categorical\_cols]  
encoders = [OneHotEncoder(inputCol=f"{c}\_IDX", outputCol=f"{c}\_VEC") for c in categorical\_cols]  
  
# --- Assemble numeric features ---  
numeric\_assembler = VectorAssembler(inputCols=numeric\_cols, outputCol="numeric\_features")  
  
# --- Scale numeric features to 0-1 ---  
scaler = MinMaxScaler(inputCol="numeric\_features", outputCol="numeric\_scaled")  
  
# --- Assemble final feature vector (scaled numeric + categorical vectors) ---  
final\_assembler = VectorAssembler(  
 inputCols=["numeric\_scaled"] + [f"{c}\_VEC" for c in categorical\_cols],  
 outputCol="features"  
)  
  
# --- Logistic Regression classifier ---  
from pyspark.ml.classification import LogisticRegression  
lr = LogisticRegression(featuresCol="features", labelCol="requires\_ai", maxIter=50, regParam=0.0, elasticNetParam=0.0)  
  
# --- Build pipeline ---  
pipeline = Pipeline(stages=indexers + encoders + [numeric\_assembler, scaler, final\_assembler, lr])  
  
# --- Split train/test and fit ---  
train\_df, test\_df = model\_df.randomSplit([0.8, 0.2], seed=42)  
lr\_model = pipeline.fit(train\_df)

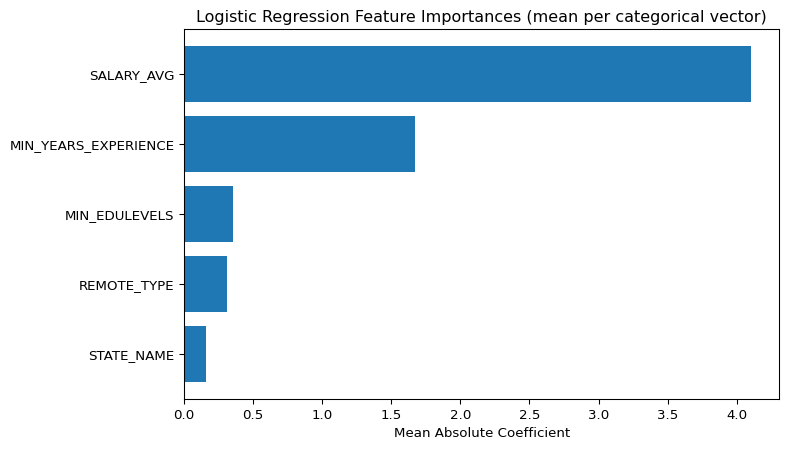
[Stage 2:> (0 + 1) / 1] [Stage 5:> (0 + 1) / 1] [Stage 8:> (0 + 1) / 1] [Stage 11:> (0 + 1) / 1][Stage 11:> (0 + 1) / 1] [Stage 14:> (0 + 1) / 1][Stage 14:> (0 + 1) / 1] [Stage 17:> (0 + 1) / 1][Stage 17:> (0 + 1) / 1] [Stage 20:> (0 + 1) / 1][Stage 20:> (0 + 1) / 1] 25/10/15 01:08:43 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS  
[Stage 21:> (0 + 1) / 1][Stage 21:> (0 + 1) / 1]

# --- Predict & evaluate ---  
predictions = lr\_model.transform(test\_df)  
evaluator = BinaryClassificationEvaluator(labelCol="requires\_ai", metricName="areaUnderROC")  
roc\_auc = evaluator.evaluate(predictions)  
print(f"ROC-AUC: {roc\_auc:.3f}")  
  
# --- Extract logistic regression stage ---  
lr\_stage = lr\_model.stages[-1]  
  
# LogisticRegressionModel in Spark stores coefficients in a dense vector  
coefficients = lr\_stage.coefficients.toArray()  
  
# --- Compute feature sizes ---  
# Number of numeric features  
num\_numeric = len(numeric\_cols)  
# One-hot vector sizes for categorical features  
features\_vec = lr\_model.transform(test\_df).select("features").first()[0]  
vector\_size = len(features\_vec)  
num\_categorical = len(categorical\_cols)  
categorical\_vector\_sizes = [(vector\_size - num\_numeric)//num\_categorical]\*num\_categorical  
  
# --- Aggregate coefficients by feature ---  
feature\_sizes = [1]\*num\_numeric + categorical\_vector\_sizes  
feature\_names = numeric\_cols + categorical\_cols  
  
agg\_importances = []  
start = 0  
for size in feature\_sizes:  
 # Take mean of absolute values of coefficients for categorical features  
 agg\_importances.append(np.mean(np.abs(coefficients[start:start+size])))  
 start += size  
  
# --- Create DataFrame & plot ---  
importance\_df = pd.DataFrame({  
 "Feature": feature\_names,  
 "Importance": agg\_importances  
}).sort\_values("Importance", ascending=False)  
  
plt.figure(figsize=(8,5))  
plt.barh(importance\_df["Feature"], importance\_df["Importance"])  
plt.xlabel("Mean Absolute Coefficient")  
plt.title("Logistic Regression Feature Importances (mean per categorical vector)")  
plt.gca().invert\_yaxis()  
plt.show()

[Stage 32:> (0 + 1) / 1][Stage 32:> (0 + 1) / 1]

ROC-AUC: 0.623

[Stage 41:> (0 + 1) / 1][Stage 41:> (0 + 1) / 1]



ROC-AUC was used instead of accuracy because it is more suited for binary classifications such as AI vs non-AI.

# --- Create sample job listings ---  
sample\_jobs = [  
 Row(MIN\_YEARS\_EXPERIENCE=10.0, SALARY\_AVG=300000.0, STATE\_NAME="Massachusetts", MIN\_EDULEVELS=3.0, REMOTE\_TYPE="Remote"),  
 Row(MIN\_YEARS\_EXPERIENCE=5.0, SALARY\_AVG=75000.0, STATE\_NAME="Texas", MIN\_EDULEVELS=0.0, REMOTE\_TYPE="Onsite"),  
 Row(MIN\_YEARS\_EXPERIENCE=10.0, SALARY\_AVG=120000.0, STATE\_NAME="Minnesota", MIN\_EDULEVELS=2.0, REMOTE\_TYPE="Hybrid"),  
 Row(MIN\_YEARS\_EXPERIENCE=15.0, SALARY\_AVG=150000.0, STATE\_NAME="New York", MIN\_EDULEVELS=2.0, REMOTE\_TYPE="Remote"),  
]  
  
sample\_df = spark.createDataFrame(sample\_jobs)  
  
# Ensure numeric columns are DoubleType  
for c in ["MIN\_YEARS\_EXPERIENCE", "SALARY\_AVG", "MIN\_EDULEVELS"]:  
 sample\_df = sample\_df.withColumn(c, col(c).cast(DoubleType()))  
  
# --- Transform samples through the trained pipeline ---  
predictions = lr\_model.transform(sample\_df)  
  
# --- Extract probability of AI class ---  
predictions\_array = predictions.withColumn("prob\_array", vector\_to\_array(col("probability")))  
  
# --- Show results ---  
results = predictions\_array.select(  
 "MIN\_YEARS\_EXPERIENCE",  
 "SALARY\_AVG",  
 "STATE\_NAME",  
 "MIN\_EDULEVELS",  
 "REMOTE\_TYPE",  
 col("prob\_array")[1].alias("AI\_prob"),  
 "prediction"  
)  
  
#results.show(truncate=False)  
# ---------- Doc-friendly table (no changes to model/pipeline) ----------  
pdf = results.toPandas()  
  
# Light formatting  
pdf["AI\_prob"] = pdf["AI\_prob"].round(3) # 0.000–1.000  
pdf["MIN\_YEARS\_EXPERIENCE"] = pdf["MIN\_YEARS\_EXPERIENCE"].round(1)  
pdf["MIN\_EDULEVELS"] = pdf["MIN\_EDULEVELS"].round(1)  
pdf["SALARY\_AVG"] = pdf["SALARY\_AVG"].map(lambda x: f"${x:,.0f}") # $300,000  
  
print(pdf.to\_string(index=False))

[Stage 42:> (0 + 8) / 8]

MIN\_YEARS\_EXPERIENCE SALARY\_AVG STATE\_NAME MIN\_EDULEVELS REMOTE\_TYPE AI\_prob prediction  
 10.0 $300,000 Massachusetts 3.0 Remote 0.524 1.0  
 5.0 $75,000 Texas 0.0 Onsite 0.237 0.0  
 10.0 $120,000 Minnesota 2.0 Hybrid 0.329 0.0  
 15.0 $150,000 New York 2.0 Remote 0.498 0.0

# Logistic Regression Summary

This analysis skimmed the job listings for keywords to distinguish AI from non-AI job postings and used a logistic regression model trained on five features: minimum years of experience, average salary, state, minimum education level, and remote work type. These features were chosen because AI roles typically require more experience, offer higher compensation, demand advanced degrees, and are concentrated in specific regions, while remote flexibility can influence applicant reach. The model revealed that years of experience and salary were the most influential factors, followed by state, education, and remote type, reflecting the expected patterns of AI job characteristics.

In order to create the model, we encoded the categorical variables and used MinMaxScaler to size the continuous variables to a 0-1 scale, which allowed the training model to weigh each variable equally. After obtaining the model data, we then charted the feature importance, making sure to take the aggregate coefficients to avoid over-reporting the importance of the encoded variables. The model reveals that salary is the most important feature when determining whether or not a role involves AI. When using the model against a sample data set, it predicted that the $300,000 salary job in Massachusetts with 10 years of minimum experience would involve AI, but the $150,000 salary job in New York with 15 minimum years of experience would not.

In order to refine its predictive capabilities, this model could benefit from some fine-tuning; right now it is evaluating probability and deeming any data with over 50% chance as involving AI, but if we were to get more granular with the data, the ideal threshold to use in this process may be closer to 35% or 40%. Additionally, given the ever-changing landscape of AI and the continued trend of AI becoming commonplace in many jobs, that threshold may continue to lower with time.