AI vs. Non-AI Careers

Final Report

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()  
spark.sparkContext.setLogLevel("FATAL")  
  
# --- 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)))  
  
# --- 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",  
 "NAICS2\_NAME",  
 "MIN\_YEARS\_EXPERIENCE",  
 "MIN\_EDULEVELS",  
 "STATE\_NAME",  
 "REMOTE\_TYPE\_NAME"  
)  
  
# --- Handle numeric nulls ---  
numeric\_cols = ["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 = ["NAICS2\_NAME", "STATE\_NAME", "REMOTE\_TYPE\_NAME"]  
model\_df = model\_df.filter(  
 (trim(col("STATE\_NAME")) != "") &  
 (trim(col("REMOTE\_TYPE\_NAME")) != "") &  
 (trim(col("NAICS2\_NAME")) != "") &  
 col("STATE\_NAME").isNotNull() &  
 col("REMOTE\_TYPE\_NAME").isNotNull() &  
 col("NAICS2\_NAME").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)

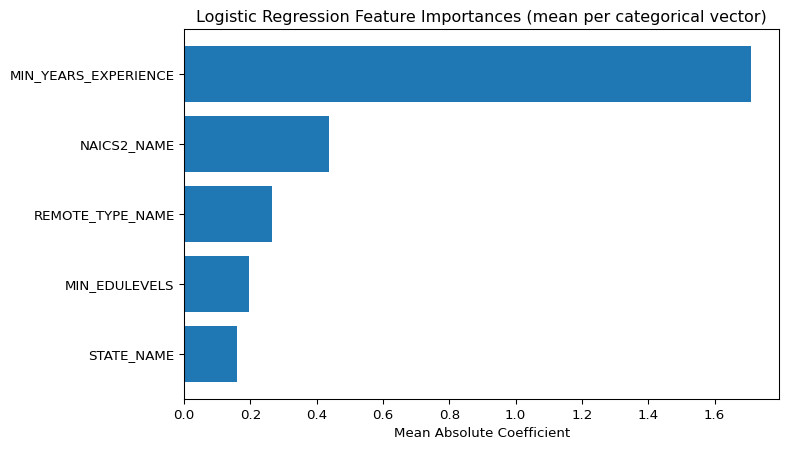
WARNING: Using incubator modules: jdk.incubator.vector  
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties  
25/10/15 13:57:48 WARN Utils: Your hostname, Kellys-MacBook-Air.local, resolves to a loopback address: 127.0.0.1; using 192.168.1.6 instead (on interface en0)  
25/10/15 13:57:48 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address  
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
25/10/15 13:57:49 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
25/10/15 13:57:49 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.  
[Stage 1:> (0 + 1) / 1] [Stage 2:> (0 + 1) / 1] [Stage 5:> (0 + 1) / 1] [Stage 8:> (0 + 1) / 1] [Stage 14:> (0 + 1) / 1] [Stage 20:> (0 + 1) / 1] [Stage 26:> (0 + 1) / 1] [Stage 29:> (0 + 1) / 1] [Stage 30:> (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.savefig("output/logistic\_regression.png", dpi=200, bbox\_inches="tight")  
plt.show()

[Stage 45:> (0 + 1) / 1]

ROC-AUC: 0.655

[Stage 54:> (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=15.0, NAICS2\_NAME="Professional, Scientific, and Technical Services", STATE\_NAME="New York", MIN\_EDULEVELS=3.0, REMOTE\_TYPE\_NAME="Remote"),  
 Row(MIN\_YEARS\_EXPERIENCE=5.0, NAICS2\_NAME="Finance and Insurance", STATE\_NAME="Texas", MIN\_EDULEVELS=2.0, REMOTE\_TYPE\_NAME="Remote"),  
 Row(MIN\_YEARS\_EXPERIENCE=10.0, NAICS2\_NAME="Real Estate and Rental and Leasing", STATE\_NAME="Minnesota", MIN\_EDULEVELS=0.0, REMOTE\_TYPE\_NAME="Hybrid Remote"),  
 Row(MIN\_YEARS\_EXPERIENCE=15.0, NAICS2\_NAME="Health Care and Social Assistance", STATE\_NAME="New York", MIN\_EDULEVELS=4.0, REMOTE\_TYPE\_NAME="Not Remote"),  
]  
  
sample\_df = spark.createDataFrame(sample\_jobs)  
  
# Ensure numeric columns are DoubleType  
for c in ["MIN\_YEARS\_EXPERIENCE", "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",  
 "NAICS2\_NAME",  
 "STATE\_NAME",  
 "MIN\_EDULEVELS",  
 "REMOTE\_TYPE\_NAME",  
 col("prob\_array")[1].alias("AI\_prob"),  
 "prediction"  
)  
  
results.show(truncate=False)

+--------------------+------------------------------------------------+----------+-------------+----------------+-------------------+----------+  
|MIN\_YEARS\_EXPERIENCE|NAICS2\_NAME |STATE\_NAME|MIN\_EDULEVELS|REMOTE\_TYPE\_NAME|AI\_prob |prediction|  
+--------------------+------------------------------------------------+----------+-------------+----------------+-------------------+----------+  
|15.0 |Professional, Scientific, and Technical Services|New York |3.0 |Remote |0.5524002949151843 |1.0 |  
|5.0 |Finance and Insurance |Texas |2.0 |Remote |0.17876889392822903|0.0 |  
|10.0 |Real Estate and Rental and Leasing |Minnesota |0.0 |Hybrid Remote |0.18450482855782802|0.0 |  
|15.0 |Health Care and Social Assistance |New York |4.0 |Not Remote |0.1876643082460473 |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, NAICS2 classification, state, minimum education level, and remote work type. These features were chosen because AI roles typically require more experience, are concentrated in certain industries over others, demand advanced degrees, and could be more prevalent in specific regions, while remote flexibility can influence applicant reach. The model revealed that years of experience was by far the most influential factor, followed by NAICS2, 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 years of experience is the most important feature when determining whether or not a role involves AI. However, the combination of all factors can lead to dramatically different results. For example, when using the model against a sample data set, it predicted that the 15-year remote listing in New York in “Professional, Scientific, and Technical Services” would be an AI job, but the 15-year on-site listing in New York in “Health Care and Social Assistance” 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. This tool should be consistently re-evaluated to maintain relevancy so that users can derive the most value from its results.