

**DEBRE BERHAN UNIVERSITY**

**COLLEEGE OF COMPUTING**

**DEPARTMENT OF SOFTWAREENGINEERING**

Fundamentals of Machine Learning

**ML Individual ASSIGNMENT**

**NAME: Tesfaye Samson**

**ID NO : DBU1402506**

**SUBMITTED TO : MR DERBEW F**

**SUBMISSION DATE :February 2, 2017 EC**

****Personalized Machine Learning Project: Lung Cancer Prediction****

## ****1. Problem Definition & Data Acquisition****

### ****Problem Statement:****

Lung cancer is one of the most prevalent and deadly forms of cancer worldwide. Early detection significantly increases survival rates. The goal of this project is to develop a classification model that predicts whether an individual has lung cancer based on various risk factors such as smoking habits, alcohol consumption, chronic diseases, and genetic history.

### ****Dataset Source:****

* **Name:** Survey Lung Cancer Dataset
* **Source:** Kaggle (<https://www.kaggle.com/>)
* **License:** Publicly available for educational and research purposes
* **Dataset Format:** CSV
* **Size:** Contains a reasonable number of observations for a machine learning model

#### ****Dataset Structure and Features****

* The data set consists of multiple features related to risk factors. The target variable is **LUNG\_CANCER (1 = Yes, 0 = No)**.

| **Feature** | **Description** | **Type** |
| --- | --- | --- |
| GENDER | Gender (M = 0, F = 1) | Categorical |
| AGE | Age of the individual | Numerical |
| SMOKING | Smoking habit (1 = Yes, 0 = No) | Categorical |
| YELLOW\_FINGERS | Yellow fingers (1 = Yes, 0 = No) | Categorical |
| ANXIETY | Anxiety issues (1 = Yes, 0 = No) | Categorical |
| PEER\_PRESSURE | Influence of peer pressure to smoke (1 = Yes, 0 = No) | Categorical |
| CHRONIC DISEASE | Presence of chronic diseases (1 = Yes, 0 = No) | Categorical |
| FATIGUE | Fatigue levels (1 = Yes, 0 = No) | Categorical |
| ALLERGY | Allergy issues (1 = Yes, 0 = No) | Categorical |
| WHEEZING | Presence of wheezing (1 = Yes, 0 = No) | Categorical |
| ALCOHOL CONSUMING | Alcohol consumption (1 = Yes, 0 = No) | Categorical |
| COUGHING | Frequency of coughing (1 = Yes, 0 = No) | Categorical |
| SHORTNESS OF BREATH | Difficulty in breathing (1 = Yes, 0 = No) | Categorical |
| SWALLOWING DIFFICULTY | Difficulty swallowing (1 = Yes, 0 = No) | Categorical |
| CHEST PAIN | Presence of chest pain (1 = Yes, 0 = No) | Categorical |

### ****Load the Dataset****

We will load the dataset using **pandas** and display the first few rows.

import pandas as pd

|  |
| --- |
| # Load dataset  df = pd.read\_csv("path\_to\_your\_file.csv")  # Display first few rows  df.head() |

### ****Summarizing Data Distributions****

We will:  
✅ Use df.describe() for numerical features

|  |
| --- |
| # Summary of numerical columns  print(df.describe()) |

**Visualizing Distributions** :We will plot histograms for all features to understand their distributions.

|  |
| --- |
| import matplotlib.pyplot as plt  # Plot histograms for all numerical columns  df.hist(figsize=(12, 10), bins=20)  plt.tight\_layout()  plt.show() |
|  |

### ****Identifying Missing Values, Outliers, and Data Quality Issues****

Now, let’s examine our dataset for:  
✅ **Missing Values** (null values)  
✅ **Outliers** (extreme values)  
✅ **Data Quality Issues** (inconsistencies, duplicates, etc.)

### ****1. Checking for Missing Values****

We need to see if any column has missing values.

|  |
| --- |
| # Check for missing values  print(df.isnull().sum()) |

### ****Identifying Outliers****

We will use boxplots to detect outliers in numerical features.

|  |
| --- |
| import seaborn as sns  import matplotlib.pyplot as plt  # Plot boxplots for numerical features to detect outliers  plt.figure(figsize=(12, 8))  df.boxplot(rot=90) # Rotate labels for readability  plt.title("Boxplot of Features to Identify Outliers")  plt.show() |

### ****Checking for Data Quality Issues****

We will check for:

* **Duplicates**
* **Inconsistent or incorrect values**

|  |
| --- |
| # Check for duplicate rows  print("Number of duplicate rows:", df.duplicated().sum())  # Check for unique values in categorical columns (to find inconsistencies)  for col in df.columns:  print(f"Unique values in {col}: {df[col].unique()}") |

Visualizing Relationships Between Features and Target Variable

### ****Correlation Heat map****

A heat map shows **how strongly each feature is related** to lung cancer.

|  |
| --- |
| import numpy as np  plt.figure(figsize=(12, 8))  correlation\_matrix = df.corr()  sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)  plt.title("Feature Correlation Heatmap")  plt.show() |

* Features with **high correlation** to lung cancer (positive or negative) are important for our model.

### ****Exploratory Data Analysis (EDA) Observations****

After conducting a detailed EDA on the lung cancer dataset, here are our key findings:

### ****1. Data Distribution****

✅ The data set consists of both **categorical** (e.g., smoking, gender, coughing) and **numerical** (e.g., age) features.  
✅ The **target variable (LUNG\_CANCER)** is binary (0 = No, 1 = Yes).  
✅ Some categorical features have imbalanced distributions, which might affect model performance.

### ****2. Missing Values and Data Quality Issues****

✅ No missing values were detected in the dataset.  
✅ The dataset appears clean and does not contain inconsistencies.

### ****3. Outliers Analysis****

* **Age Distribution:**
* The majority of patients are between **50-70 years old**, with a few extreme outliers.
* Outliers in age may indicate incorrect data or genuinely rare cases.
* **Boxplot Observations:**
* Some categorical features show strong differences in distribution when split by cancer status.
* Features such as **SMOKING, YELLOW\_FINGERS, COUGHING, SHORTNESS\_OF\_BREATH**, and **CHEST\_PAIN** exhibit clear separation between 0 (No Cancer) and 1 (Cancer).

### ****4. Relationship Between Features and Lung Cancer****

✅ **Heat map Correlation Analysis:**

* **Strong positive correlation** between **smoking, coughing, wheezing, and lung cancer presence**.
* **Age has a weaker correlation** with lung cancer but still plays a role.
* Some features like **peer pressure and anxiety** show little correlation with lung cancer, meaning they might be less important predictors.

### ****5. Feature Selection Insights****

* Features like **smoking, coughing, shortness of breath, wheezing, chest pain, and yellow fingers** appear to be the most important predictors.
* Features like **peer pressure and anxiety** may have less predictive value.

**Data Preprocessing:**

### ****Checking for Missing Values****

**check if there are any missing values in the data set:**

|  |
| --- |
| # Checking for missing values  missing\_values = df.isnull().sum()  print("Missing values in each column:\n", missing\_values) |

****If there are missing values:****

* We will fill numerical features with the **median** to avoid biasing the data.
* For categorical features, we will fill missing values with the **mode (most frequent value).**

|  |
| --- |
| # Filling missing values  df.fillna(df.median(numeric\_only=True), inplace=True) # Fill numeric missing values with median  df.fillna(df.mode().iloc[0], inplace=True) # Fill categorical missing values with mode  print("Missing values handled successfully!") |

### ****Handling Outliers****

Outliers can distort our model, so we will use the **Inter quartile Range (IQR) Method** to detect and cap them.  
We'll do this only for **Age**, as other columns are binary (0/1).

|  |
| --- |
| Q1 = df['AGE'].quantile(0.25)  Q3 = df['AGE'].quantile(0.75)  IQR = Q3 - Q1  lower\_bound = Q1 - 1.5 \* IQR  upper\_bound = Q3 + 1.5 \* IQR  df['AGE'] = np.where(df['AGE'] < lower\_bound, lower\_bound, df['AGE'])  df['AGE'] = np.where(df['AGE'] > upper\_bound, upper\_bound, df['AGE'])  print("Outliers handled successfully!") |

## ****Encoding Categorical Features****

We already encoded **Gender** as:

* **Male → 0, Female → 1**

Since all other variables are binary (0/1), no additional encoding is needed

## ****Scaling / Normalizing Numerical Features****

Since **Age** has a wide range of values, we will **normalize it** using **Min Max Scaler**.

|  |
| --- |
| from sklearn.preprocessing import MinMaxScaler  # Initialize MinMaxScaler  scaler = MinMaxScaler()  # Apply scaling to Age  df['Age'] = scaler.fit\_transform(df[['Age']])  print("Age feature scaled successfully!") |

· This code **scales "Age"** between **0 and 1** using **Min-Max Scaling**.

Helps in improving **model performance** by normalizing the feature.

·

**Model Implementation and Training**

Since our task is **predicting lung cancer (Yes=1, No=0)**, the target variable is **categorical (binary: 0 or 1)**.

* **This means we need a Classification model.**

**Best Choice: Random Forest Classifier**

### ****Splitting the Data set into Training and Testing Sets****

We need to split our data set into **training (80%)** and **testing (20%)** sets. The **training set** is used to train the model, and the **testing set** helps us evaluate its performance.

We'll use train\_test\_split from sklearn.model\_selection.

|  |
| --- |
| from sklearn.model\_selection import train\_test\_split  # Define features (X) and target variable (y)  X = df.drop(columns=['Lung\_Cancer']) # Features  y = df['Lung\_Cancer'] # Target variable  # Split the dataset: 80% train, 20% test  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)  print(" Dataset successfully split into training and testing sets!") |

### ****Training the Random Forest Model****

Now, let's **train the Random Forest Classifier** using our training data:

|  |
| --- |
| from sklearn.ensemble import RandomForestClassifier  # Initialize the model  rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)  # Train the model  rf\_model.fit(X\_train, y\_train)  print("✅ Model training completed!") |

### ****Hyperparameter Tuning****

Instead of using default settings, we can **optimize hyper parameters** to improve accuracy.  
We'll use **Grid Search CV** to find the best values for n\_estimators, max\_depth, etc

Key Hyperparameters for RandomForestClassifier:

* **n\_estimators** → Number of trees in the forest.
* **max\_depth** → Maximum depth of each tree.
* **min\_samples\_split** → Minimum number of samples needed to split a node.
* Grid Search CV is a method in **scikit-learn** that automatically finds the best combination of hyper parameters by trying different values.

|  |
| --- |
| from sklearn.model\_selection import GridSearchCV  # Define hyperparameter grid  param\_grid = {  'n\_estimators': [50, 100, 200],  'max\_depth': [None, 10, 20],  'min\_samples\_split': [2, 5, 10]  }  # Initialize GridSearchCV  grid\_search = GridSearchCV(RandomForestClassifier(random\_state=42), param\_grid, cv=5, scoring='accuracy')  # Fit the grid search to find the best parameters  grid\_search.fit(X\_train, y\_train)  # Get the best parameters  best\_params = grid\_search.best\_params\_  print(f"✅ Best Hyperparameters: {best\_params}")  # Train the best model  best\_rf\_model = RandomForestClassifier(\*\*best\_params, random\_state=42)  best\_rf\_model.fit(X\_train, y\_train) |

**Hyperparameter Tuning Documentation**

### ****Objective****

The goal of hyperparameter tuning is to optimize the performance of our **Random Forest Classifier** by selecting the best combination of hyperparameters. This prevents overfitting, improves generalization, and maximizes classification accuracy.

### ****Hyperparameters Considered:****

* n\_estimators: Number of trees in the forest.
* Higher values can improve performance but increase computation time.
* We tested **50, 100, and 200**.
* max\_depth: Maximum depth of each tree.
* Prevents overfitting by limiting tree growth.
* We tested **None (unlimited), 10, and 20**.
* min\_samples\_split: Minimum number of samples required to split an internal node.
* Prevents overfitting by ensuring splits happen only with enough data.
* We tested **2, 5, and 10**.

### ****Approach: Grid Search with Cross-Validation****

We used **GridSearchCV**, which performs an exhaustive search over all possible hyperparameter combinations using **5-Fold Cross-Validation**.

* Best Hyperparameters: {'max\_depth': None, 'min\_samples\_split': 5, 'n\_estimators': 200}

***THIS MEANS*** :This means the trees in the Random Forest **grow until all leaves are pure** (or until they contain less than min\_samples\_split samples).A node must have **at least 5 samples** before splitting.The Random Forest will use **200 decision trees** instead of the default 100.

Model Evaluation and Analysis

**evaluate the model's performance** using appropriate classification metrics:

**Make Predictions on Test Data**

|  |
| --- |
| y\_pred = best\_rf\_model.predict(X\_test)  y\_prob = best\_rf\_model.predict\_proba(X\_test)[:, 1] |

#### ****Compute Evaluation Metrics****

Since this is a **classification problem**, we will use:  
✔️ **Accuracy**: How many total predictions were correct?  
 **Precision**: Out of all positive predictions, how many were correct?  
**Recall**: Out of all actual positive cases, how many were correctly predicted?  
**F1-score**: A balance between precision and recall.  
**AUC-ROC**: Measures the ability of the model to differentiate between classes.

|  |
| --- |
| from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix, classification\_report  # Calculate metrics  accuracy = accuracy\_score(y\_test, y\_pred)  precision = precision\_score(y\_test, y\_pred)  recall = recall\_score(y\_test, y\_pred)  f1 = f1\_score(y\_test, y\_pred)  auc = roc\_auc\_score(y\_test, y\_prob)  # Print results  print(f"✅ Accuracy: {accuracy:.4f}")  print(f"✅ Precision: {precision:.4f}")  print(f"✅ Recall: {recall:.4f}")  print(f"✅ F1-score: {f1:.4f}")  print(f"✅ AUC-ROC: {auc:.4f}")  # Classification Report  print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))  # Confusion Matrix  import seaborn as sns  import matplotlib.pyplot as plt  cm = confusion\_matrix(y\_test, y\_pred)  plt.figure(figsize=(5, 4))  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Cancer", "Cancer"], yticklabels=["No Cancer", "Cancer"])  plt.xlabel("Predicted")  plt.ylabel("Actual")  plt.title("Confusion Matrix")  plt.show() |

#### ****Compare with a Baseline Model****

To see if our model actually improves upon **random guessing**, we compare it with a **Dummy Classifier** (which makes random predictions based on class distribution).

|  |
| --- |
| from sklearn.dummy import DummyClassifier  dummy\_clf = DummyClassifier(strategy="most\_frequent") # Always predicts the majority class  dummy\_clf.fit(X\_train, y\_train)  dummy\_pred = dummy\_clf.predict(X\_test)  # Evaluate Dummy Model  dummy\_accuracy = accuracy\_score(y\_test, dummy\_pred)  print(f"⚠️ Baseline Model Accuracy: {dummy\_accuracy:.4f}")  print(f"🚀 Our Model Accuracy: {accuracy:.4f}") |

### ****Interpretation of Metrics****

* **High accuracy** but low recall? → Model might be biased toward the majority class.
* **High precision but low recall?** → Model avoids false positives but misses many true positives.
* **AUC-ROC close to 1?** → The model is very good at distinguishing between the classes.

Deploy your trained **Random Forest model** as an API using **FastAPI**.

## ****Save the Model****

First, save the trained model using joblib:

|  |
| --- |
| import joblib  # Save the trained model  joblib.dump(best\_rf\_model, "lung\_cancer\_model.pkl")  print("✅ Model saved successfully!") |

### ****Detailed Steps to Deploy Your Lung Cancer Prediction Model API****

We will **deploy** the FastAPI application on **Render**, which is a free and easy-to-use cloud hosting platform.

## ****Save Your Trained Model****

Before deployment, **save** your trained **Random Forest model** using joblib

|  |
| --- |
| import joblib  # Save the trained model  joblib.dump(best\_rf\_model, "lung\_cancer\_model.pkl")  print("✅ Model saved successfully!") |

* ***install FastAPI and Uvicorn***

Open **Anaconda Prompt:**

|  |
| --- |
| **pip install fastapi uvicorn joblib numpy pydantic** |

## ****Create Your FastAPI App****

Now, create a **new Python file** called app.py and add the following code:

|  |
| --- |
| import numpy as np  import pandas as pd  from fastapi import FastAPI  import joblib  app = FastAPI()  # Load trained model  model = joblib.load("lung\_cancer\_model.pkl")  @app.post("/predict")  def predict\_lung\_cancer(data: dict):  try:  # Convert input data to NumPy array  input\_data = np.array([[  data['age'], data['smoking'], data['yellow\_fingers'],  data['anxiety'], data['peer\_pressure'], data['chronic\_disease'],  data['fatigue'], data['allergy'], data['wheezing'],  data['alcohol\_consumption'], data['coughing'],  data['shortness\_of\_breath'], data['swallowing\_difficulty'],  data['chest\_pain']  ]])  # If "lung\_cancer" is missing, add a default value (e.g., 0)  if 'lung\_cancer' in data:  input\_data = np.append(input\_data, [[data['lung\_cancer']]], axis=1)  else:  input\_data = np.append(input\_data, [[0]], axis=1) # Default to 0  # Make prediction  prediction = model.predict(input\_data)  return {"lung\_cancer\_prediction": int(prediction[0])}  except Exception as e:  return {"error": str(e)} |

* Anaconda Prompt I WILL NAVIGATE TO MY FOLDER WHERE THE TRAINED DATA AND THE app.py is saved and run this code:

|  |
| --- |
| uvicorn app:app --reload |

* **Access the API Docs** by opening this URL in your browser:  
   <http://127.0.0.1:8000/docs>

### ****Potential Limitations and Future Improvements of the Lung Cancer Prediction Model****

## ****Limitations of the Current Model****

**Limited Dataset Quality**

* If the dataset is **small or imbalanced**, the model may struggle with generalization.
* If it contains **biased** data, predictions may be skewed.

**Feature Selection Issues**

* The model assumes that the selected features (e.g., smoking, anxiety) are the best predictors.
* There might be **missing critical factors**, such as genetic predisposition or pollution exposure.

**Overfitting Risks**

* If the model is too complex (e.g., deep trees in RandomForest), it may **memorize** training data instead of generalizing.
* This leads to high accuracy on training but poor performance on real-world data.

**Model Deployment Challenges**

* If the model isn’t **optimized**, predictions may take longer, affecting real-time applications.
* Hosting on limited resources (e.g., local FastAPI server) may not handle large traffic loads.

## ****Future Improvements****

**Enhance Dataset & Feature Engineering**

* Collect **more real-world data** from diverse sources.
* Add **more relevant features** like genetics, air pollution, family history, etc.

**Try Advanced Models**

* Experiment with **Neural Networks, XGBoost, or SVM** for better accuracy.
* Use **feature importance techniques** to refine input selection.

**Use Explainable AI (XAI)**

* Apply **SHAP (SHapley Additive Explanations)** to understand which factors drive predictions.
* Visualize decision-making to improve trust in predictions.