Big data analytics

Theory Project

Muhammad Saad Ali

FA22-BDS-030

**Project Part 1**

1. **Overview**

* **Problem**

Develop a web-based application that predicts the likelihood of a person being COVID-19 positive based on symptoms and exposure history. The app collects user inputs through a frontend interface and uses a machine learning model served via a FastAPI backend to provide real-time predictions with confidence levels.

* **Dataset**
  + **Name**

Covid-19 Prediction Dataset

* + **Link**

<https://www.kaggle.com/datasets/hemanthhari/symptoms-and-covid-presence>

1. **Data Processing**

columns\_to\_drop = [

    'Asthma', 'Chronic Lung Disease', 'Headache', 'Heart Disease', 'Diabetes',

    'Hyper Tension', 'Abroad travel', 'Attended Large Gathering',

    'Family working in Public Exposed Places', 'Sanitization from Market'

]

df = df.drop(columns=columns\_to\_drop)

df.head()

le = LabelEncoder()

for col in df.columns:

    df[col] = le.fit\_transform(df[col].astype(str))

X = df.drop("COVID-19", axis=1)

y = df["COVID-19"]

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

1. **Model Development**

def eval\_metrics(y\_true, y\_pred):

return {

"accuracy": accuracy\_score(y\_true, y\_pred),

"precision": precision\_score(y\_true, y\_pred),

"recall": recall\_score(y\_true, y\_pred),

"f1": f1\_score(y\_true, y\_pred)

}

os.makedirs("models", exist\_ok=True)

models = {

"LogisticRegression": LogisticRegression(max\_iter=1000, class\_weight='balanced'),

"RandomForest": RandomForestClassifier(class\_weight='balanced'),

"XGBoost": XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', scale\_pos\_weight=3.0)

}

best\_model = None

best\_score = 0

mlflow.set\_experiment("COVID\_Prediction")

for name, model in models.items():

with mlflow.start\_run(run\_name=name):

mlflow.set\_tag("Dataset", "Covid Dataset")

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

metrics = eval\_metrics(y\_test, y\_pred)

mlflow.log\_params(model.get\_params())

mlflow.log\_metrics(metrics)

input\_example = X\_test[:5]

signature = infer\_signature(X\_train, model.predict(X\_train))

model\_path = f"models/{name}.pkl"

joblib.dump(model, model\_path)

mlflow.sklearn.log\_model(model, name + "\_model", signature=signature, input\_example=input\_example)

print(f"\n{name} - F1 Score: {metrics['f1']:.4f}")

if metrics["f1"] > best\_score:

best\_score = metrics["f1"]

best\_model = model

1. **Model Saving and Versioning**
   1. **Save best model**

A screen shot of a computer

AI-generated content may be incorrect.

* 1. **Log training and evaluation results**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

1. **Cloud Storage Setup**

**Project Part 2**

1. **Frontend**
   1. **Streamlit Code**

import streamlit as st

import requests

with open("style.css") as f:

st.markdown(f"<style>{f.read()}</style>", unsafe\_allow\_html=True)

st.markdown('<div class="full-header">COVID-19 Prediction App</div>', unsafe\_allow\_html=True)

def yes\_no\_input(label):

return 1 if st.selectbox(label, ["No", "Yes"]) == "Yes" else 0

breathing\_problem = yes\_no\_input("Breathing Problem")

fever = yes\_no\_input("Fever")

dry\_cough = yes\_no\_input("Dry Cough")

sore\_throat = yes\_no\_input("Sore Throat")

running\_nose = yes\_no\_input("Running Nose")

fatigue = yes\_no\_input("Fatigue")

gastrointestinal = yes\_no\_input("Gastrointestinal Issue")

contact = yes\_no\_input("Contact with COVID Patient")

visited = yes\_no\_input("Visited Public Exposed Places")

masks = yes\_no\_input("Wearing Masks")

if st.button("Predict"):

input\_data = {

"breathing\_problem": breathing\_problem,

"fever": fever,

"dry\_cough": dry\_cough,

"sore\_throat": sore\_throat,

"running\_nose": running\_nose,

"fatigue": fatigue,

"gastrointestinal": gastrointestinal,

"contact\_with\_covid\_patient": contact,

"visited\_public\_exposed\_places": visited,

"wearing\_masks": masks

}

res = requests.post("http://localhost:8000/predict", json=input\_data)

if res.status\_code == 200:

result = res.json()

if "prediction" in result:

popup\_class = "result-danger" if result["prediction"] == "COVID Positive" else "result-success"

st.markdown(

f'<div class="result-popup {popup\_class}">'

f'Prediction: {result["prediction"]}<br>Confidence: {result["confidence"]}%</div>',

unsafe\_allow\_html=True

)

else:

st.error(f"Error: {result.get('error', 'Unknown error')}")

else:

st.error("Failed to get prediction from server.")

* 1. **CSS**

.full-header {

width: 100%;

background-color: lightgrey;

padding: 20px 0;

text-align: center;

font-size: 2em;

font-weight: bold;

margin-bottom: 20px;

}

.stSelectbox, .stNumberInput {

background-color: #f0f0f0;

padding: 10px;

border-radius: 5px;

margin-bottom: 15px;

}

.stButton>button {

background-color: white;

color: black;

float: right;

padding: 0.5em 2em;

transition: background-color 0.3s, color 0.3s;

}

.stButton>button:hover {

background-color: #007BFF;

color: white;

}

.result-popup {

position: fixed;

bottom: 100px;

left: 50%;

transform: translateX(-50%);

padding: 20px 40px;

border-radius: 8px;

font-size: 1.2em;

color: white;

animation: fadeInUp 0.5s ease-out forwards;

}

.result-success {

background-color: #28a745;

}

.result-danger {

background-color: #dc3545;

}

@keyframes fadeInUp {

from {

opacity: 0;

transform: translateX(-50%) translateY(20px);

}

to {

opacity: 1;

transform: translateX(-50%) translateY(0);

}

}

1. **Backend**

from fastapi import FastAPI

from api.schema import PredictRequest

from api.model\_loader import model

import numpy as np

app = FastAPI()

@app.get("/health")

def health\_check():

return {"status": "running"}

@app.post("/predict")

def predict(data: PredictRequest):

try:

features = [

data.breathing\_problem,

data.fever,

data.dry\_cough,

data.sore\_throat,

data.running\_nose,

data.fatigue,

data.gastrointestinal,

data.contact\_with\_covid\_patient,

data.visited\_public\_exposed\_places,

data.wearing\_masks

]

prediction = model.predict([features])[0]

probability = model.predict\_proba([features])[0][1]

return {

"prediction": "COVID Positive" if prediction == 1 else "COVID Negative",

"confidence": round(float(probability) \* 100, 2) # Cast to float here

}

except Exception as e:

return {"error": str(e)}

1. **Loading the Model**

import joblib

import os

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

model\_path = os.path.join(BASE\_DIR, 'models', 'best\_model.pkl')

model = joblib.load(model\_path)

1. **CI/CD Pipeline**

A computer screen shot of a black screen

AI-generated content may be incorrect.

1. **Deployment**
2. **Testing**

|  |  |
| --- | --- |
|  |  |