**🩺 Project:HEALTHCARE DATASET**

**✅ Project Content**

This repository contains the code, documentation, and analysis of a healthcare dataset that spans patient demographics, diagnoses, treatments, financial data, and hospital details. The project aims to extract meaningful insights, support decision-making, and suggest areas for further research in healthcare analytics.

**Project Code:**

import pandas as pd, numpy as np, pickle, gradio as gr

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.linear\_model import LogisticRegression

with open("logistic\_model.pkl", "rb") as f:

model, feature\_names, scaler, label\_encoders = pickle.load(f)

def predict\_healthcare(age, gender, blood\_type, date\_of\_admission, doctor, hospital,

insurance\_provider, billing\_amount, room\_number, admission\_type,

discharge\_date, medication, test\_results):

try:

input\_dict = {

"Age": age,

"Gender": label\_encoders["Gender"].transform([gender])[0],

"Blood Type": label\_encoders["Blood Type"].transform([blood\_type])[0],

"Date of Admission": pd.to\_datetime(date\_of\_admission).toordinal(),

"Doctor": label\_encoders["Doctor"].transform([doctor])[0],

"Hospital": label\_encoders["Hospital"].transform([hospital])[0],

"Insurance Provider": label\_encoders["Insurance Provider"].transform([insurance\_provider])[0],

"Billing Amount": billing\_amount,

"Room Number": room\_number,

"Admission Type": label\_encoders["Admission Type"].transform([admission\_type])[0],

"Discharge Date": pd.to\_datetime(discharge\_date).toordinal(),

"Medication": label\_encoders["Medication"].transform([medication])[0],

"Test Results": label\_encoders["Test Results"].transform([test\_results])[0]

}

input\_array = np.array([input\_dict[feat] for feat in feature\_names]).reshape(1, -1)

pred = model.predict(scaler.transform(input\_array))[0]

return label\_encoders["Medical Condition"].inverse\_transform([pred])[0]

except Exception as e:

return f"Error: {e}"

choices = lambda key: label\_encoders[key].classes\_.tolist()

interface = gr.Interface(

fn=predict\_healthcare,

inputs=[

gr.Number(label="Age"),

gr.Dropdown(choices=choices("Gender"), label="Gender"),

gr.Dropdown(choices=choices("Blood Type"), label="Blood Type"),

gr.Textbox(label="Date of Admission (YYYY-MM-DD)"),

gr.Dropdown(choices=choices("Doctor"), label="Doctor"),

gr.Dropdown(choices=choices("Hospital"), label="Hospital"),

gr.Dropdown(choices=choices("Insurance Provider"), label="Insurance Provider"),

gr.Number(label="Billing Amount"),

gr.Number(label="Room Number"),

gr.Dropdown(choices=choices("Admission Type"), label="Admission Type"),

gr.Textbox(label="Discharge Date (YYYY-MM-DD)"),

gr.Dropdown(choices=choices("Medication"), label="Medication"),

gr.Dropdown(choices=choices("Test Results"), label="Test Results"),

],

**Key Technologies :**

| **Technology** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Description** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Python 3.9+** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Core programming language |
| **Pandas** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Data manipulation and analysis |
| **NumPy** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Numerical operations |
| **Matplotlib** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Data visualization |
| **Seaborn** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Advanced statistical visualizations |
| **Scikit-learn** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Machine learning and predictive modeling |
| **Jupyter** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Notebook environment for interactive coding |

\

**Description :**

This project is designed to explore and analyze a synthetic healthcare dataset with 55,500 records. Each record represents a patient and contains information such as:

* Demographics (Age, Gender, Blood Type)
* Diagnosis (Medical Condition)
* Hospital Admission Details
* Billing and Insurance
* Treatments and Test Results

**Key Goals:**

1. Identify trends in medical conditions and their correlation with age or gender.
2. Analyze hospital admission types and associated costs.
3. Detect anomalies or outliers in billing.
4. Visualize relationships between diagnosis, treatment, and outcomes.

## Output:

### 1. Age Distribution

### 2. Most Common Medical Conditions

### 3. Billing Analysis by Admission Type

### 4. Gender vs Condition Heatmap

### 5. Insurance Provider Distribution

These visualizations help identify which age groups are most vulnerable to specific diseases, where healthcare costs are highest, and which treatments are most commonly used.

## Further Research :

This project opens the door to deeper investigations such as:

1. **Predictive Modeling:**
   * Predicting patient readmission risk
   * Forecasting billing costs based on diagnosis and treatment
2. **Natural Language Processing:**
   * Extracting insights from medical notes or doctor remarks (if available)
3. **Real-time Monitoring Systems:**
   * Integrating with hospital data for dynamic dashboards
4. **Bias and Fairness:**
   * Analyzing potential demographic biases in treatment or cost
5. **Insurance Optimization:**
   * Suggesting cost-effective treatment plans based on historical billing

**Cats vs Dogs Image Classification**

**Project Description**

This project implements a binary image classification model to distinguish between images of cats and dogs. The model uses transfer learning with the pre-trained MobileNetV2 architecture on the TensorFlow Cats vs Dogs dataset.

**Key Technologies**

* Python
* TensorFlow 2.x
* TensorFlow Datasets
* Transfer Learning (MobileNetV2)
* Matplotlib for visualization

**Project Content**

* Dataset loading and preprocessing
* Transfer learning with MobileNetV2 (pretrained on ImageNet)
* Model training and evaluation
* Image prediction function with visualization

**Code Snippet**

import tensorflow as tf

import tensorflow\_datasets as tfds

import matplotlib.pyplot as plt

from io import BytesIO

import PIL.Image

import requests

IMG\_SIZE = 160

# Load dataset

(train\_ds, val\_ds), ds\_info = tfds.load(

'cats\_vs\_dogs',

split=['train[:80%]', 'train[80%:]'],

with\_info=True,

as\_supervised=True

)

# Preprocessing function

def format\_image(image, label):

image = tf.image.resize(image, (IMG\_SIZE, IMG\_SIZE))

image = image / 255.0

return image, label

train\_ds = train\_ds.map(format\_image).batch(32).prefetch(buffer\_size=tf.data.AUTOTUNE)

val\_ds = val\_ds.map(format\_image).batch(32).prefetch(buffer\_size=tf.data.AUTOTUNE)

# Load pretrained MobileNetV2 model without classifier head

base\_model = tf.keras.applications.MobileNetV2(input\_shape=(IMG\_SIZE, IMG\_SIZE, 3),

include\_top=False,

weights='imagenet')

base\_model.trainable = False # Freeze base model

# Build the model

model = tf.keras.Sequential([

base\_model,

tf.keras.layers.GlobalAveragePooling2D(),

tf.keras.layers.Dense(1, activation='sigmoid') # Binary classification output

])

# Compile the model

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

# Train the model

history = model.fit(train\_ds, validation\_data=val\_ds, epochs=3)

# Prediction function

def predict\_image(url):

img = PIL.Image.open(BytesIO(requests.get(url).content)).resize((IMG\_SIZE, IMG\_SIZE))

img\_array = tf.keras.preprocessing.image.img\_to\_array(img)

img\_array = tf.expand\_dims(img\_array, 0) / 255.0

prediction = model.predict(img\_array)[0][0]

label = "Dog" if prediction > 0.5 else "Cat"

plt.imshow(img)

plt.title(f"Prediction: {label} ({prediction:.2f})")

plt.axis('off')

plt.show()

# Example usage

predict\_image("https://upload.wikimedia.org/wikipedia/commons/3/3a/Cat03.jpg")

**Output**

* The model will train on 80% of the Cats vs Dogs dataset and validate on the remaining 20%.
* After training, the predict\_image function downloads an image from a URL, preprocesses it, and displays the image along with the predicted label (Cat or Dog) and the model’s confidence score.
* Example output:   
    
  *Prediction: Cat (0.95)*

**Further Research**

* **Fine-tuning:** Unfreeze part of the MobileNetV2 layers and retrain to improve accuracy.
* **Data augmentation:** Apply image transformations to increase dataset variety.
* **Other architectures:** Experiment with different pretrained models like ResNet, EfficientNet, or DenseNet.
* **Multi-class classification:** Extend the model to classify other animal species.
* **Deploy model:** Build a web or mobile app that uses the model for real-time predictions.

.

# 🎬 Project: IMDB Movie Review Sentiment Analysis Using LSTM

## 📁 Project Content

* Load the IMDB movie review dataset.
* Clean and preprocess text data (remove HTML, punctuation, stopwords).
* Convert text to sequences and pad them.
* Build and train an LSTM-based deep learning model for binary sentiment classification.
* Evaluate model performance on test data.
* Test the model on custom sample movie reviews.

## 📝 Complete Project Code

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import re

import nltk

nltk.download('stopwords')

nltk.download('punkt')

from nltk.corpus import stopwords

from bs4 import BeautifulSoup

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from keras.layers import Dropout

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Embedding, LSTM

import pandas.testing as tm

# Load dataset

movie\_reviews = pd.read\_csv("/content/IMDB Dataset.csv")

# Check shape and nulls

print(movie\_reviews.shape)

print(movie\_reviews.isnull().sum())

# Visualize sentiment distribution

sns.countplot(x='sentiment', data=movie\_reviews)

plt.show()

# Remove HTML tags

def strip\_html(text):

soup = BeautifulSoup(text, "html.parser")

return soup.get\_text()

movie\_reviews['review'] = movie\_reviews['review'].apply(strip\_html)

# Remove punctuations and single chars

def remove\_punctuations(text):

pattern = r'[^a-zA-Z0-9\s]'

text = re.sub(pattern, '', text)

text = re.sub(r"\s+[a-zA-Z]\s+", ' ', text) # single chars

text = re.sub(r'\s+', ' ', text) # multiple spaces

return text

movie\_reviews['review'] = movie\_reviews['review'].apply(remove\_punctuations)

# Customize stopwords list to keep negations

stopword\_list = stopwords.words('english')

updated\_stopword\_list = [w for w in stopword\_list if w != 'not' and not w.endswith("n't")]

# Remove stopwords (excluding negations)

from nltk.tokenize import word\_tokenize

stop\_words = set(updated\_stopword\_list)

def remove\_stopwords(text):

words = word\_tokenize(text)

filtered = [word for word in words if word.lower() not in stop\_words]

return ' '.join(filtered)

movie\_reviews['review'] = movie\_reviews['review'].apply(remove\_stopwords)

# Convert sentiment to binary labels

movie\_reviews['sentiment'] = movie\_reviews['sentiment'].apply(lambda x: 1 if x == "positive" else 0)

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(movie\_reviews['review'].values, movie\_reviews['sentiment'].values,

test\_size=0.20, random\_state=42)

# Tokenization and sequences

tokenizer = Tokenizer(num\_words=5000)

tokenizer.fit\_on\_texts(X\_train)

X\_train\_tok = tokenizer.texts\_to\_sequences(X\_train)

X\_test\_tok = tokenizer.texts\_to\_sequences(X\_test)

vocab\_size = len(tokenizer.word\_index) + 1

maxlen = 100

X\_train\_pad = pad\_sequences(X\_train\_tok, padding='post', maxlen=maxlen, truncating='post')

X\_test\_pad = pad\_sequences(X\_test\_tok, padding='post', maxlen=maxlen, truncating='post')

print('Number of unique words in the corpus:', vocab\_size)

# Build LSTM model

model = Sequential()

model.add(Embedding(input\_dim=vocab\_size, output\_dim=100, input\_length=maxlen))

model.add(LSTM(64, return\_sequences=True))

model.add(Dropout(0.3))

model.add(LSTM(32))

model.add(Dropout(0.3))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())

# Train model

history = model.fit(X\_train\_pad, y\_train, batch\_size=128, epochs=3, verbose=1, validation\_split=0.2)

# Evaluate model

y\_test = np.array(y\_test)

score, acc = model.evaluate(X\_test\_pad, y\_test, batch\_size=128)

print(f'Test score: {score}')

print(f'Test accuracy: {acc}')

print(f'Accuracy: {acc:.2%}')

# Test on new samples

test\_samples = [

"This movie is fantastic! I really like it because it is so good!",

"Good movie!",

"Maybe I like this movie.",

"Not to my taste, will skip and watch another movie",

"If you like action, then this movie might be good for you.",

"Bad movie!",

"Not a good movie!",

"This movie really sucks! Can I get my money back please?"

]

test\_samples\_tokens = tokenizer.texts\_to\_sequences(test\_samples)

test\_samples\_tokens\_pad = pad\_sequences(test\_samples\_tokens, maxlen=maxlen)

pred = model.predict(test\_samples\_tokens\_pad)

# Display predictions

for text, p in zip(test\_samples, pred):

sentiment = "Positive" if p[0] > 0.5 else "Negative"

print(f"Review: {text}\nPrediction Score: {p[0]:.4f} --> Sentiment: {sentiment}\n")

## 🔑 Key Technologies

* **Python** — main programming language
* **Pandas, NumPy** — data manipulation
* **Matplotlib, Seaborn** — data visualization
* **NLTK** — natural language processing (tokenization, stopwords)
* **BeautifulSoup** — HTML parsing and cleaning
* **TensorFlow / Keras** — deep learning framework for model building
* **Scikit-learn** — train-test splitting and evaluation

## 📄 Description

The project focuses on sentiment classification of movie reviews from the IMDB dataset by building an LSTM neural network:

* First, data is loaded and explored to ensure balance and no missing values.
* Reviews are cleaned by stripping HTML tags and removing punctuations.
* Stopwords are removed except negations (not, n't) to preserve sentiment meaning.
* Text is tokenized and sequences are padded to a uniform length.
* An LSTM model is constructed with two LSTM layers and dropout for regularization.
* The model is trained to predict positive or negative sentiment.
* Finally, the model is tested on unseen reviews and some custom sample sentences.

## 🎯 Expected Output

* Dataset shape: (50000, 2)
* Sentiment distribution plot showing roughly equal positive and negative counts.
* Model summary printed with layer info.
* Training progress showing loss and accuracy per epoch.
* Test accuracy ~85% (varies based on random seed and training)
* Sample predictions with sentiment scores and labels, e.g.:

Review: This movie is fantastic! I really like it because it is so good!

Prediction Score: 0.9567 --> Sentiment: Positive

Review: Not a good movie!

Prediction Score: 0.1034 --> Sentiment: Negative

Review: This movie really sucks! Can I get my money back please?

Prediction Score: 0.0412 --> Sentiment: Negative

## 🔍 Further Research & Improvements

* **Hyperparameter tuning:** Explore different LSTM units, embedding sizes, batch sizes, learning rates.
* **Pretrained embeddings:** Use GloVe, Word2Vec or FastText embeddings instead of training from scratch.
* **Bidirectional LSTM:** Improve context understanding by reading sequences forward and backward.
* **More epochs:** Longer training might increase accuracy.
* **Advanced preprocessing:** Lemmatization, handling slang, emojis, and contractions.
* **Other architectures:** Test CNNs, Transformer models, or ensemble models.
* **Explainability:** Use LIME or SHAP to interpret model predictions.
* **Deployment:** Build a simple web app using Flask, FastAPI, or Gradio for live predictions.
* **Multilingual support:** Adapt pipeline to other languages or multilingual reviews.