The Superior University

Project Title

Medical Diagnosis System (Cancer prediction)

Project Details

1. Course: Artificial Intelligence
2. Instructor: Sir Rasikh Ali
3. Semester: 3rd
4. Section: BSAI-3A
5. Name: Muhammad Haseeb

Abstract

This project focuses on building a medical diagnosis system to predict cancer using machine learning. Early detection of cancer saves lives, and this system aims to help doctors with quick and accurate predictions. We used a dataset with medical information, cleaned it, and applied machine learning techniques such as SVM and Random Forest to predict whether a patient has cancer. With this in mind, the accuracy of the system is improved through scaling the data, balancing classes, and fine-tuning the models. The results indicate that it has an ability to make accurate predictions, thereby making it helpful to healthcare professionals. The report explains how the system was developed, tested, and evaluated, giving suggestions for how it may be done better in the future.

Report Structure

1. Title Page

Title: Medical Diagnosis System for Cancer Prediction

Name: Muhammad Haseeb

Table of Content:

Introduction

Dataset

Methodology

Implementation

Result

Discussion

Conclusion

Video link

2. Introduction

What is this project about?

- This project uses machine learning to predict cancer from medical data.
- Early detection is important for saving lives, and this tool can assist doctors.

Why did we do this?

- Diagnosing cancer accurately is hard and time-consuming. This system can provide fast and reliable predictions.

Goals:

- Process medical data to make it suitable for machine learning.
- Train machine learning models to predict cancer.
- Test the system for accuracy and performance.

3. Dataset

What kind of data was used?

- Database containing medical information including the size of the tumor, age, etc. and other details related to patients.
  Target variable was whether the patient is cancer-positive or not.

What did we do with the data?

- Data cleaned so that there were no errors present and missing values filled in.
- Transcribed the text-based data into numerical data. For example, male = 1 and female = 0
- Scaling numerical features so that all of them are at the same scale.

---

4. Methodology

Data Preparation:

- Load the dataset (CSV file).
- Fill missing values using column means for data completeness.
- Split data into features (X) and target (y).

Data Preprocessing:

- Scale features using **Standard Scaler** for uniformity.
- Split data into training (80%) and testing (20%) sets.

Model Training and Evaluation:

- Train **Random Forest, logistic Regression Classifier** on the training data.
- Evaluate model performance using accuracy on the test set.

Validation & Error Handling:

- Validate the model with test data.
- Handle errors gracefully, ensuring a smooth user experience.

Prediction and saving model:

- Predict the model for checking the cancer is benign and malignant.
- Using pickle for save the model.

5. Implementation:

Importing libraries and classifier:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score , precision_score , recall_score, classification_report

import warnings
warnings.filterwarnings('ignore')
```

Data loading:

```
df = pd.read_csv('dataset/data.csv')
df
```

✓ 0.1s Python

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con points_m
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10
...	...	...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00

569 rows × 33 columns

Data Preprocessing:

```
df.describe().T
```

Python

```
df=df.drop('Unnamed: 32',axis=1)
```

Python

```
df.duplicated().sum()
```

Python

```
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})
```

Python

```
df.corr()['diagnosis'].sort_values(ascending=False)
```

Python

```
cancerous = df[df['diagnosis'] == 1]
non_cancerous = df[df['diagnosis'] == 0]
```

Python

Model Training and Evaluation:

Logistic Regression

```
log_reg = LogisticRegression()
log_reg.fit(X_train,y_train)
```

```
y_predtest = log_reg.predict(X_test)
y_predtrain = log_reg.predict(X_train)
```

```
print("\nAccuracy Score:")
print(f"Train Accuracy: {accuracy_score(y_train, y_predtrain)}")
print(f"Test Accuracy: {accuracy_score(y_test, y_predtest)}")
```

```
print("\nPrecision Score:")
print(f"Train Precision: {precision_score(y_train, y_predtrain)}")
print(f"Test Precision: {precision_score(y_test, y_predtest)}")
```

```
print("\nRecall Score:")
print(f"Train Recall: {recall_score(y_train, y_predtrain)}")
print(f"Test Recall: {recall_score(y_test, y_predtest)}")
```

```
from imblearn.over_sampling import SMOTE

# Features and target
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

# Apply SMOTE
smote = SMOTE(random_state=42)
X_balanced, y_balanced = smote.fit_resample(X, y)

# Verify the new class distribution
from collections import Counter
print("Class distribution after SMOTE:")
print(Counter(y_balanced))
```

```
print(X.shape)
```

```
print(X.columns)
```

```
X_train, X_test, y_train, y_test= train_test_split(X_balanced,y_balanced,test_size=0.2,random_state=42)
```

RandomForestModel

```
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

k_fold = KFold(n_splits= 5, shuffle=True, random_state=42)
```

```
model = RandomForestClassifier(n_estimators=100 , criterion = "entropy" , max_depth= 4 ,random_state= 0)
scores = cross_val_score(model, X, y, cv=k_fold, scoring='accuracy')
print("Cross-validation scores:", scores)
print("Mean accuracy:", scores.mean()*100)
print("Standard deviation:", scores.std())
model.fit(X_train, y_train)
```

+ Code + Markdown

```
y_predTest = model.predict(X_test)
y_predTrain = model.predict(X_train)
```

Model saving:

Saving the Model and Scaler

```
import pickle
# Train the RandomForestClassifier model
rf_model = RandomForestClassifier(n_estimators=100, criterion="entropy", max_depth=4, random_state=0)
rf_model.fit(X_train, y_train)
# Assuming rf_model and sc are your trained model and scaler
try:
    with open('model.pkl', 'wb') as model_file:
        pickle.dump(model, model_file)
    with open('features.pkl', 'wb') as features_file:
        pickle.dump(important_features, features_file)
    print("Model, scaler, and features saved successfully!")

except PermissionError as e:
    print(f"Permission error: {e}. Please check file permissions.")

except Exception as e:
    print(f"An error occurred: {e}")
```

```
# Output the accuracy of the model
print(f"Model accuracy on test set: {model.score(X_test, y_test)}")
```

Flask implementation:

```
app.py > predict
def predict():
            for i, feature in enumerate(all_features):
                feature_value = float(request.form[feature])  # Get the feature value from the form
                input_data[i] = feature_value

            # Scale the input data using the same scaler used in training
            scaled_input = scaler.transform([input_data])  # Ensure input is 2D

            # Make prediction using the trained model
            prediction = model.predict(scaled_input)
            prediction_prob = model.predict_proba(scaled_input)[0][prediction[0]]

            # Interpret prediction
            diagnosis = "Malignant" if prediction[0] == 1 else "Benign"

            # Return the result to the user with prediction and probability
            return render_template('index.html',
                                   diagnosis=diagnosis,
                                   probability=f'{prediction_prob * 100:.2f}%',
                                   all_features=all_features)
        except ValueError:
            # Handle cases where input values are invalid
            return render_template('index.html',
                                   diagnosis="Error: Please input numeric values for all features.",
                                   all_features=all_features)
        except Exception as e:
            # Handle other potential errors
            return render_template('index.html',
                                   diagnosis=f"An unexpected error occurred: {e}",
                                   all_features=all_features)


if __name__ == "__main__":
    app.run(debug=True)
```

```
import pickle
import numpy as np

# Load the trained model and scaler
model = pickle.load(open('model.pkl', 'rb'))
scaler = pickle.load(open('scaler.pkl', 'rb'))

# Define the 22 features used in training
all_features = [
    'mean radius', 'mean texture', 'mean perimeter', 'mean area', 'mean smoothness',
    'mean compactness', 'mean concavity', 'mean concave points', 'mean symmetry', 'mean fractal dimension',
    'radius error', 'texture error', 'perimeter error', 'area error', 'smoothness error',
    'compactness error', 'concavity error', 'concave points error', 'symmetry error', 'fractal dimension error',
    'worst radius', 'worst texture'
]

app = Flask(__name__)

@app.route('/')
def home():
    # Render the form with all 22 features
    return render_template('index.html', all_features=all_features)

@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        try:
            # Initialize an array for all 22 features
            input_data = np.zeros(22)  # Assuming 22 features in total

            # Dynamically retrieve all 22 features from the form
            for i, feature in enumerate(all_features):
                feature_value = float(request.form[feature])  # Get the feature value from the form
```

6. Results

- The SVM model gave the best accuracy of around [X%] (replace with your result).
- Random Forest also performed well, but slightly lower than SVM.

The model performed best after we:

- Scaled the data.
- Balanced the dataset using SMOTE.
- Tuned the hyperparameters using GridSearchCV.

---

7. Discussion

What worked well?

- SVM was very accurate for this dataset.
- Data cleaning and balancing improved the model's performance.

What were the challenges?

- The dataset was imbalanced, meaning one class had fewer samples.
- Preprocessing took time but was necessary for better results.

How can we improve it?

- Use a larger dataset to train the model better.
- Try advanced techniques like deep learning for more accuracy.

---

8. Conclusion

- Built a machine learning system to predict cancer with good accuracy.
- Showed how machine learning can support healthcare professionals.
- Early detection of cancer saves lives, and this system can make diagnoses faster and easier.

---

9. Demo video link

https://drive.google.com/file/d/1QeFMFTY8ztRtsw0Agdlh5y6JA7F8sjuJ/view?usp=sharing

https://drive.google.com/file/d/1ffN2qFJwyhWvkgM8sL5fsLnhCpbZIWUN/view?usp=sharing

---