Task 2

Disease Prediction

Data Description

133 columns have been provided with 132 of them being symptoms experienced by patients and last column in prognosis for the same patient.

Examples to symptoms are itching ,skin_rash , nodal_skin_eruptions, continuous_sneezing shivering, chills, joint_pain, stomach_pain, acidity,ulcers_on_tongue, scurring,skin_peeling , silver_like_dusting, small_dents_in_nails ,inflammatory_nails ,blister, red_sore_around_nose, yellow_crust_ooze

1.Preprocessing step

1.1 Remove unnamed column as it contain NAn values or 0

1.2 Check for null values , we find that data is cleaned and there is no null values , 0 means it doesn't have the symptom or 1 means it has this symptom.

1.3 Display the unique values in the 'prognosis' column of your DataFrame and prints the total number of unique values and check if the data is balanced or not , and It is balanced as every disease occurrences are equal (120 people per disease)

1.4 Handle Categorical Values

The pd.get_dummies() function in pandas is used to convert categorical variable(s) into dummy/indicator variables. When applied to the target variable Y, it essentially converts each unique category in Y into a separate binary column.

2. Split the data into 70% training and 30 % testing & allow shuffling

3. Standard Scaling to features

By applying this standardization process, the features in both the training and test datasets are transformed to have zero mean and unit variance, which is a common preprocessing step in machine learning workflows.

4. Building the model

4.1 Decision trees can be a good model for disease prediction problems for several reasons:

1. Interpretability:

- Decision trees are inherently interpretable. The rules in a decision tree are easy to understand, making it possible to interpret and communicate the model's decisions to both experts and non-experts.

2. Feature Importance:

- Decision trees provide a clear measure of feature importance. You can easily identify which features are most relevant for making predictions. This is valuable for understanding the factors contributing to a disease.

3. No Need for Feature Scaling:

- Decision trees are not sensitive to the scale of input features. Unlike some other models like SVM, where feature scaling is crucial, decision trees can work with features on different scales.

4. Efficiency:

- Decision trees are computationally efficient during both training and prediction, especially for smaller to medium-sized datasets.

The model gives a very good accuracy at testing which is `0.99`

4.2 Random Forest

Random Forest often performs well in disease prediction for several reasons:

1. **Handling Non-Linearity:**
    - Credit scoring problems are often non-linear, meaning the relationship between features (financial variables) and creditworthiness may not be a simple straight line. Random Forest can capture complex non-linear relationships between features and the target variable.
2. **Ensemble Learning:**
    - Random Forest is an ensemble learning method that builds multiple decision trees and combines their predictions. This helps to reduce overfitting and improve generalization to new, unseen data.
3. **Feature Importance:**
    - Random Forest provides a measure of feature importance. In credit scoring, understanding which financial variables contribute most to the prediction of creditworthiness is crucial for interpretability. Random Forest can highlight important features.
4. **Tuning Flexibility:**
    - Random Forest has hyperparameters that can be tuned for optimal performance. Parameters like the number of trees, tree depth, and the number of features considered at each split provide flexibility to optimize the model.
5. **Reducing Overfitting:**
    - Random Forest introduces randomness during the training process, both in terms of the features considered at each split and the data used to build each tree. This randomness helps prevent overfitting and improves the model's ability to generalize.

The model gives a very good accuracy at testing which is `0.978`.

4.3 Multilayer Perceptron

A Multilayer Perceptron (MLP), which is a type of artificial neural network, can be suitable for disease prediction problems, especially when dealing with complex and non-linear relationships in the data. After getting dummy variables for the Y labels of diseases (one-hot encoding), an MLP can be trained to predict the probability or class of a particular disease based on input features.

classifierMLP = MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000, random_state=42)

classifierMLP.fit(x_train, y_train)

The model gives accuracy 100 % which means it may suffer from overfitting problem which can be fixed by adding more data and more tuning to hyperparameters.

4.4 Deep Neural Network

Like MLPs, deep neural networks can be prone to overfitting, leading to high accuracy on the training set but poor generalization to new data. Regularization techniques and cross-validation are essential to mitigate this.

```
# Assuming y_train contains your class labels as strings
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
num_classes = len(np.unique(y_train_encoded))

# Assuming y_train_encoded contains your class labels as integers
from keras.utils import to_categorical

# Convert integer labels to one-hot encoding
y_train_one_hot = to_categorical(y_train_encoded, num_classes=num_classes)

# Build and compile the model
model = Sequential()
model.add(Dense(32, activation='relu', input_shape=(x_train.shape[1],)))
model.add(Dense(16, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
early_stopping = EarlyStopping(patience=2, monitor='val_accuracy')

# Train the model
```

```
model.fit(x_train, y_train_one_hot, batch_size=40, epochs=10,
validation_split=0.25, callbacks=[early_stopping])
```

Epoch 1/10

74/74 [==============================] - 1s 4ms/step - loss: 3.5490 - accuracy: 0.1565 - val_loss: 3.3074 - val_accuracy: 0.2358

Epoch 2/10

74/74 [==============================] - 0s 2ms/step - loss: 2.8432 - accuracy: 0.3709 - val_loss: 2.2829 - val_accuracy: 0.6230

Epoch 3/10

74/74 [==============================] - 0s 2ms/step - loss: 1.6308 - accuracy: 0.8069 - val_loss: 1.0406 - val_accuracy: 0.9167

Epoch 4/10

74/74 [==============================] - 0s 2ms/step - loss: 0.6434 - accuracy: 0.9621 - val_loss: 0.3486 - val_accuracy: 0.9980

Epoch 5/10

74/74 [==============================] - 0s 2ms/step - loss: 0.2201 - accuracy: 1.0000 - val_loss: 0.1328 - val_accuracy: 1.0000

Epoch 6/10

74/74 [==============================] - 0s 2ms/step - loss: 0.0945 - accuracy: 1.0000 - val_loss: 0.0682 - val_accuracy: 1.0000

Epoch 7/10

74/74 [==============================] - 0s 2ms/step - loss: 0.0520 - accuracy: 1.0000 - val_loss: 0.0414 - val_accuracy: 1.0000

Explanation

Steps

Label Encoding:

The LabelEncoder is used to convert class labels (y_train) from strings to integers. This step is crucial for neural network training since the network requires numerical inputs.

One-Hot Encoding:

The integer-encoded labels (y_train_encoded) are then converted to one-hot encoding using to_categorical. One-hot encoding is necessary for categorical (multi-class) classification problems.

one-hot encoding is a standard practice in multi-class classification tasks to ensure compatibility with the requirements of machine learning models, loss functions, and output layer activations. It helps avoid potential misinterpretations of ordinal relationships and ensures that the model can effectively learn and make predictions in a categorical context.

Neural Network Model:

The neural network model is constructed using the Sequential API from Keras. It consists of three layers: an input layer with 32 neurons and ReLU activation, a hidden layer with 16 neurons and ReLU activation, and an output layer with a softmax activation function (suitable for multi-class classification).

Compilation:

The model is compiled with the Adam optimizer, categorical crossentropy loss (appropriate for multi-class classification), and accuracy as the evaluation metric.

Early Stopping:

The EarlyStopping callback is set up to monitor the validation accuracy during training. If the accuracy does not improve for a certain number of epochs (defined by the patience parameter), training will be stopped early to prevent overfitting.

Training:

The model is trained using the fit method, where x_train is the input data, y_train_one_hot is the one-hot encoded labels, batch_size is the number of samples per gradient update, and epochs is the number of training epoch.

4.5 Linear Classifiers

Support Vector Machines (SVM) and Logistic Regression are two classical machine learning algorithms that are often used for binary and multiclass classification problems, including disease prediction.

Problems of linear classifiers :

Feature Overfitting:

In medical datasets, there might be a large number of features, and some of these features could be irrelevant or noisy. If the models try to fit to every feature, including noise, they may not generalize well to new, unseen data.

- Limited Data:

Medical datasets are often limited in size due to the challenges of data collection and privacy concerns. Limited data can lead to overfitting, especially if the model is complex. The model might memorize the training data instead of learning the underlying patterns.

- Lack of Diversity in Data:

Disease prediction datasets may lack diversity, and if the model is too complex, it might memorize specific patterns present in the training data without generalizing well to different cases