Neural Network Model Analysis

1. Introduction

This document outlines the implementation and evaluation of a neural network model used to classify instances of a dataset, specifically targeting the breast cancer dataset from sklearn. The objective is to predict whether a tumor is malignant or benign based on several features measured in the dataset.The data was randomly split into training (80%) and testing (20%) sets using sklearn's train_test_split function.

Model Configuration

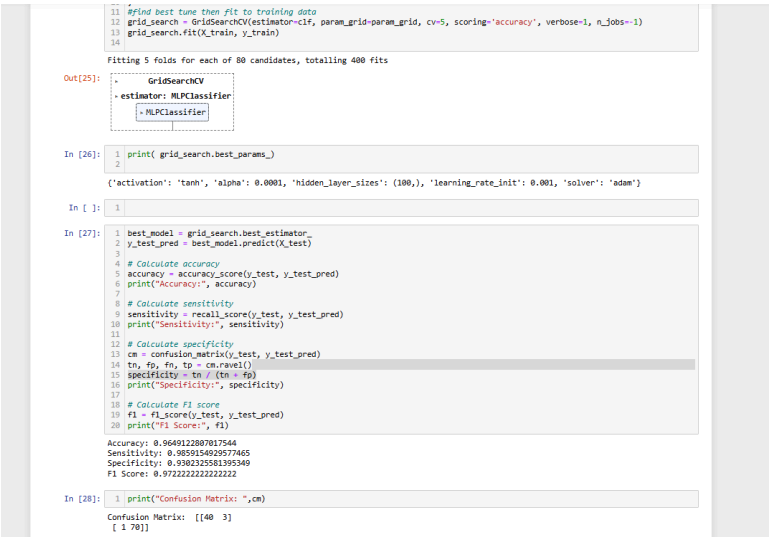
A Multi-layer Perceptron (MLP) classifier was employed. The choice of an MLP was motivated by its capability to learn non-linear relationships and my previous experience with Neural Networks. 5 fold Cross validation was used to prevent overfitting. Using gridsearch I found the best tune to be:

Best parameters: {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'learning_rate_init': 0.001, 'solver': 'adam'}

6. Results

The results highlighted the model’s capability to effectively classify the instances with considerable accuracy. Sensitivity and specificity metrics indicated the model's balanced performance in identifying both classes. The f1 score shows that the model was well balanced and did not prioritize specificity or sensitivity. Overall the accuracy was quite high as well.

1. Accuracy: 0.9649122807017544
2. Sensitivity: 0.9859154929577465
3. Specificity: 0.9302325581395349
4. F1 Score: 0.9722222222222222



```
#Find best tune then fit to training data
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=5, scoring='accuracy', verbose=1, n_jobs=-1)
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
```

Out[25]: GridSearchCV — estimator: MLPClassifier — MLPClassifier

In [26]:
```
print( grid_search.best_params_)
```

```
{'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'learning_rate_init': 0.001, 'solver': 'adam'}
```

In [ ]:
```

```

In [27]:
```
best_model = grid_search.best_estimator_
y_test_pred = best_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_test_pred)
print("Accuracy:", accuracy)

# Calculate sensitivity
sensitivity = recall_score(y_test, y_test_pred)
print("Sensitivity:", sensitivity)

# Calculate specificity
cm = confusion_matrix(y_test, y_test_pred)
tn, fp, fn, tp = cm.ravel()
specificity = tn / (tn + fp)
print("Specificity:", specificity)

# Calculate F1 score
f1 = f1_score(y_test, y_test_pred)
print("F1 Score:", f1)
```

```
Accuracy: 0.9649122807017544
Sensitivity: 0.9859154929577465
Specificity: 0.9302325581395349
F1 Score: 0.9722222222222222
```

In [28]:
```
print("Confusion Matrix: ",cm)
```

```
Confusion Matrix:  [[40  3]
 [ 1 70]]
```