Australian dataset classification

March 23, 2023

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

[2]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
 ↪recall_score
```

[3]:
```
import warnings
warnings.filterwarnings("ignore")
```

[4]:
```
data = pd.read_csv('australian.csv')
```

[5]:
```
data.head(2)
```

[5]:
```
   @inputs A1    A2    A3  A4  A5  A6    A7  A8  A9  A10  A11  A12  A13   A14  \
0           1  2208  1146   2   4   4  1585   0   0    0    1    2  100  1213
1           0  2267     7   2   8   4   165   0   0    0    0    2  160     1

   @output Class
0              0
1              0
```

[6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   @inputs A1      690 non-null    int64
 1   A2              690 non-null    int64
 2   A3              690 non-null    int64
 3   A4              690 non-null    int64
 4   A5              690 non-null    int64
 5   A6              690 non-null    int64
 6   A7              690 non-null    int64
 7   A8             690 non-null    int64
 8   A9             690 non-null    int64
 9   A10            690 non-null    int64
 10  A11            690 non-null    int64
 11  A12            690 non-null    int64
 12  A13            690 non-null    int64
 13  A14            690 non-null    int64
 14  @output Class  690 non-null    int64
dtypes: int64(15)
memory usage: 81.0 KB
```

[7]:
```
data = data.rename(columns={"@inputs A1": "A1", "@output Class":"Class"})
```

[8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 15 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A1      690 non-null    int64
 1   A2      690 non-null    int64
 2   A3      690 non-null    int64
 3   A4      690 non-null    int64
 4   A5      690 non-null    int64
 5   A6      690 non-null    int64
 6   A7      690 non-null    int64
 7   A8      690 non-null    int64
 8   A9      690 non-null    int64
 9   A10     690 non-null    int64
 10  A11     690 non-null    int64
 11  A12     690 non-null    int64
 12  A13     690 non-null    int64
 13  A14     690 non-null    int64
 14  Class   690 non-null    int64
dtypes: int64(15)
memory usage: 81.0 KB
```

[9]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data.iloc[:, :-1], data.
 ↪iloc[:, -1], test_size=0.2, random_state=42)
```

0.0.1 Logistic Regression

[10]:
```
# Train and evaluate a Logistic Regression model
lr_model = LogisticRegression(random_state=42)
```

[11]:

```
lr_model.fit(X_train, y_train)
lr_preds = lr_model.predict(X_test)
```

[12]:

```
lr_acc = accuracy_score(y_test, lr_preds)
lr_prec = precision_score(y_test, lr_preds)
lr_rec = recall_score(y_test, lr_preds)
lr_f1 = f1_score(y_test, lr_preds)
```

[13]:

```
print("Logistic Regression Accuracy:", lr_acc)
print("Logistic Regression Precision:", lr_prec)
print("Logistic Regression Recall:", lr_rec)
print("Logistic Regression F1 Score:", lr_f1)
```

```
Logistic Regression Accuracy: 0.7608695652173914
Logistic Regression Precision: 0.6666666666666666
Logistic Regression Recall: 0.7058823529411765
Logistic Regression F1 Score: 0.6857142857142857
```

[14]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import numpy as np
import itertools
```

[15]:

```
# Define class labels
classes = ['Class A', 'Class B', 'Class C']
```

[16]:

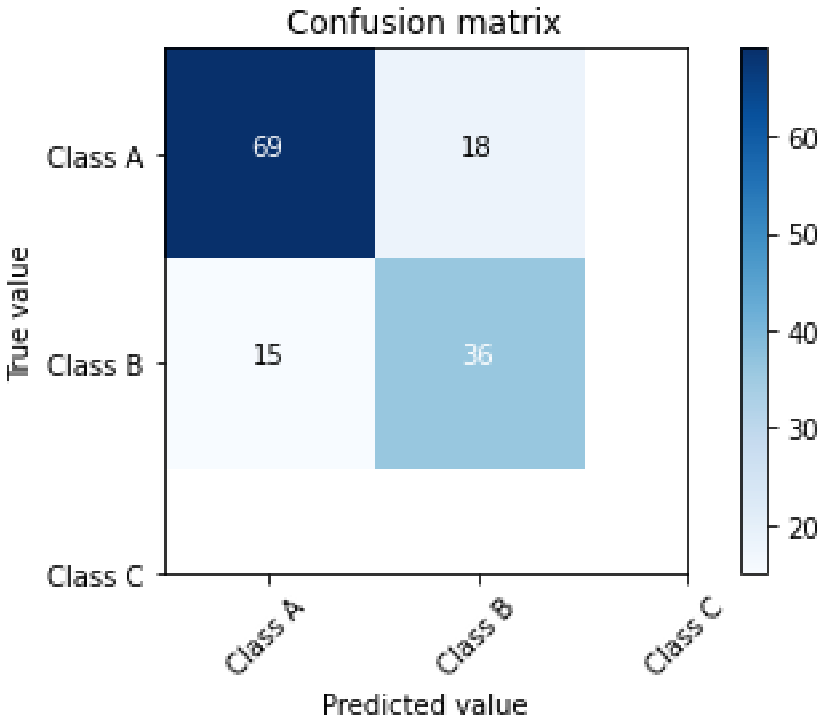
```
# Compute confusion matrix
cm = confusion_matrix(y_test, lr_preds)
```

[17]:

```
# Plot confusion matrix
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
plt.xlabel('Predicted value')
plt.ylabel('True value')

# Add text to each cell
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], 'd'),
             horizontalalignment="center",
             color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.show()
```



0.0.2 Decision Tree

[18]:
```
# Train and evaluate a Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
```

[18]:
```
DecisionTreeClassifier(random_state=42)
```

[19]:
```
dt_preds = dt_model.predict(X_test)
dt_acc = accuracy_score(y_test, dt_preds)
dt_prec = precision_score(y_test, dt_preds)
dt_rec = recall_score(y_test, dt_preds)
dt_f1 = f1_score(y_test, dt_preds)
```

[20]:
```
print("Results of Decision Tree:")
print("Decision Tree Accuracy:", dt_acc)
print("Decision Tree Precision:", dt_prec)
print("Decision Tree Recall:", dt_rec)
print("Decision Tree F1 Score:", dt_f1)
```

```
Results of Decision Tree:
Decision Tree Accuracy: 0.8333333333333334
Decision Tree Precision: 0.7692307692307693
Decision Tree Recall: 0.7843137254901961
Decision Tree F1 Score: 0.7766990291262137
```

[21]:

```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
```

[22]:

```python
# Calculate predicted probabilities for positive class
dt_probs = dt_model.predict_proba(X_test)[:, 1]
```

[23]:

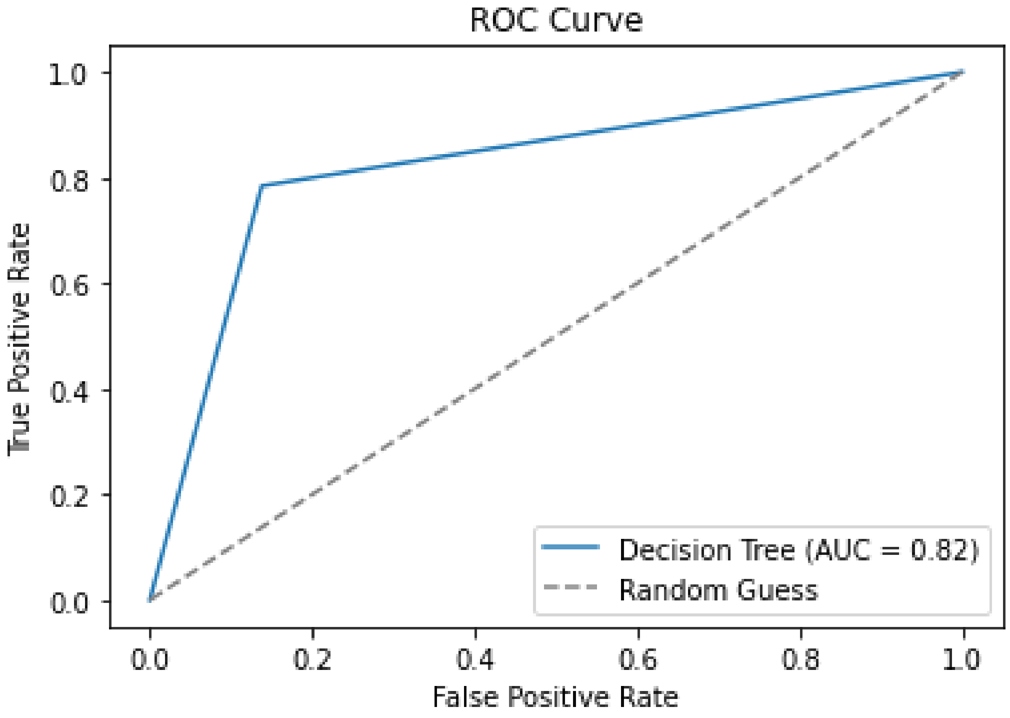
```python
# Calculate FPR, TPR, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, dt_probs)
```

[24]:

```python
# Calculate AUC
auc_dt = auc(fpr, tpr)
```

[25]:

```python
# Plot ROC curve
plt.plot(fpr, tpr, label='Decision Tree (AUC = {:.2f})'.format(auc_dt))
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



0.0.3 Gradient Boosting

[26]:
```
gb_model = GradientBoostingClassifier(random_state=42)
gb_model.fit(X_train, y_train)
```

[26]:
```
GradientBoostingClassifier(random_state=42)
```

[27]:
```
gb_preds = gb_model.predict(X_test)
```

[28]:
```
gb_acc = accuracy_score(y_test, gb_preds)
gb_prec = precision_score(y_test, gb_preds)
gb_rec = recall_score(y_test, gb_preds)
gb_f1 = f1_score(y_test, gb_preds)
```

[29]:
```
print("Results of Gradient Boosting")
print("Gradient Boosting Accuracy:", gb_acc)
print("Gradient Boosting Precision:", gb_prec)
print("Gradient Boosting Recall:", gb_rec)
print("Gradient Boosting F1 Score:", gb_f1)
```

```
Results of Gradient Boosting
Gradient Boosting Accuracy: 0.8695652173913043
Gradient Boosting Precision: 0.8367346938775511
Gradient Boosting Recall: 0.803921568627451
Gradient Boosting F1 Score: 0.8200000000000001
```

0.0.4 Artificial Neural Network

[30]:
```
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
```

[31]:
```
# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[32]:
```
# Define the ANN model
model = Sequential()
model.add(Dense(units=16, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(units=8, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
```

[33]:
```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
  ↪metrics=['accuracy'])
```

[34]:

```
# Train the model
model.fit(X_train, y_train, batch_size=32, epochs=50)
```

```
Epoch 1/50
18/18 [==============================] - 1s 2ms/step - loss: 0.7125 - accuracy:
0.5344
Epoch 2/50
18/18 [==============================] - 0s 2ms/step - loss: 0.6512 - accuracy:
0.6757
Epoch 3/50
18/18 [==============================] - 0s 2ms/step - loss: 0.6018 - accuracy:
0.7174
Epoch 4/50
18/18 [==============================] - 0s 2ms/step - loss: 0.5588 - accuracy:
0.7518
Epoch 5/50
18/18 [==============================] - 0s 2ms/step - loss: 0.5200 - accuracy:
0.7826
Epoch 6/50
18/18 [==============================] - 0s 2ms/step - loss: 0.4844 - accuracy:
0.8043
Epoch 7/50
18/18 [==============================] - 0s 2ms/step - loss: 0.4551 - accuracy:
0.8207
Epoch 8/50
18/18 [==============================] - 0s 2ms/step - loss: 0.4280 - accuracy:
0.8406
Epoch 9/50
18/18 [==============================] - 0s 1ms/step - loss: 0.4053 - accuracy:
0.8496
Epoch 10/50
18/18 [==============================] - 0s 1ms/step - loss: 0.3858 - accuracy:
0.8605
Epoch 11/50
18/18 [==============================] - 0s 1ms/step - loss: 0.3706 - accuracy:
0.8641
Epoch 12/50
18/18 [==============================] - 0s 2ms/step - loss: 0.3580 - accuracy:
0.8678
Epoch 13/50
18/18 [==============================] - 0s 2ms/step - loss: 0.3470 - accuracy:
0.8750
Epoch 14/50
18/18 [==============================] - 0s 2ms/step - loss: 0.3378 - accuracy:
0.8822
Epoch 15/50
18/18 [==============================] - 0s 2ms/step - loss: 0.3306 - accuracy:
0.8822
```

```
Epoch 16/50
18/18 [==============================] - 0s 2ms/step - loss: 0.3237 - accuracy:
0.8859
Epoch 17/50
18/18 [==============================] - 0s 2ms/step - loss: 0.3182 - accuracy:
0.8895
Epoch 18/50
18/18 [==============================] - 0s 2ms/step - loss: 0.3133 - accuracy:
0.8931
Epoch 19/50
18/18 [==============================] - 0s 1ms/step - loss: 0.3088 - accuracy:
0.8913
Epoch 20/50
18/18 [==============================] - 0s 1ms/step - loss: 0.3052 - accuracy:
0.8913
Epoch 21/50
18/18 [==============================] - 0s 1ms/step - loss: 0.3014 - accuracy:
0.8877
Epoch 22/50
18/18 [==============================] - 0s 1ms/step - loss: 0.2985 - accuracy:
0.8913
Epoch 23/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2959 - accuracy:
0.8913
Epoch 24/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2927 - accuracy:
0.8967
Epoch 25/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2897 - accuracy:
0.8967
Epoch 26/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2872 - accuracy:
0.8967
Epoch 27/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2849 - accuracy:
0.8967
Epoch 28/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2827 - accuracy:
0.8967
Epoch 29/50
18/18 [==============================] - 0s 1ms/step - loss: 0.2801 - accuracy:
0.9004
Epoch 30/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2785 - accuracy:
0.9004
Epoch 31/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2768 - accuracy:
0.9022
```

```
Epoch 32/50
18/18 [==============================] - 0s 1ms/step - loss: 0.2753 - accuracy:
0.9022
Epoch 33/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2725 - accuracy:
0.9040
Epoch 34/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2715 - accuracy:
0.9022
Epoch 35/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2697 - accuracy:
0.9022
Epoch 36/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2680 - accuracy:
0.9040
Epoch 37/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2662 - accuracy:
0.9040
Epoch 38/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2651 - accuracy:
0.9040
Epoch 39/50
18/18 [==============================] - 0s 1ms/step - loss: 0.2634 - accuracy:
0.9058
Epoch 40/50
18/18 [==============================] - 0s 1ms/step - loss: 0.2620 - accuracy:
0.9040
Epoch 41/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2612 - accuracy:
0.9040
Epoch 42/50
18/18 [==============================] - 0s 1ms/step - loss: 0.2604 - accuracy:
0.9058
Epoch 43/50
18/18 [==============================] - 0s 1ms/step - loss: 0.2586 - accuracy:
0.9076
Epoch 44/50
18/18 [==============================] - 0s 1ms/step - loss: 0.2583 - accuracy:
0.9076
Epoch 45/50
18/18 [==============================] - 0s 1ms/step - loss: 0.2567 - accuracy:
0.9094
Epoch 46/50
18/18 [==============================] - 0s 1ms/step - loss: 0.2548 - accuracy:
0.9094
Epoch 47/50
18/18 [==============================] - 0s 1ms/step - loss: 0.2535 - accuracy:
0.9094
```

```
Epoch 48/50
18/18 [==============================] - 0s 1ms/step - loss: 0.2522 - accuracy:
0.9076
Epoch 49/50
18/18 [==============================] - 0s 2ms/step - loss: 0.2515 - accuracy:
0.9149
Epoch 50/50
18/18 [==============================] - 0s 1ms/step - loss: 0.2505 - accuracy:
0.9094
```

[34]:
```
<keras.callbacks.History at 0x1afa656fa00>
```

[35]:
```python
# Evaluate the model
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)
```

```
5/5 [==============================] - 0s 1ms/step
```

[36]:
```python
accuracy = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

[37]:
```python
print("Results of Artificial Neural Network:")
print("ANN Accuracy:", accuracy)
print("ANN Precision:", prec)
print("ANN Recall:", rec)
print("ANN F1 Score:", f1)
```

```
Results of Artificial Neural Network:
ANN Accuracy: 0.8840579710144928
ANN Precision: 0.8888888888888888
ANN Recall: 0.7843137254901961
ANN F1 Score: 0.8333333333333334
```