Ionosphere dataset classification

March 24, 2023

[2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

[3]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
 ↪recall_score
```

[4]:
```
import warnings
warnings.filterwarnings("ignore")
```

[5]:
```
data = pd.read_csv('ionosphere.csv')
```

[6]:
```
data.head(5)
```

[6]:
```
   @inputs Pulse1  Pulse3  Pulse4  Pulse5  Pulse6  Pulse7  Pulse8  Pulse9  \
0               1   0.995  -0.059   0.852   0.023   0.834  -0.377   1.000
1               1   1.000  -0.188   0.930  -0.362  -0.109  -0.936   1.000
2               1   1.000  -0.452   1.000   1.000   0.712  -1.000   0.000
3               1   1.000  -0.024   0.941   0.065   0.921  -0.233   0.772
4               1   0.023  -0.006  -0.099  -0.119  -0.008  -0.118   0.147

   Pulse10  Pulse11  …  Pulse26  Pulse27  Pulse28  Pulse29  Pulse30  \
0    0.038    0.852  …   -0.512    0.411   -0.462    0.213   -0.341
1   -0.045    0.509  …   -0.266   -0.205   -0.184   -0.190   -0.116
2    0.000    0.000  …    0.907    0.516    1.000    1.000   -0.201
3   -0.164    0.528  …   -0.652    0.133   -0.532    0.024   -0.622
4    0.066    0.038  …   -0.015   -0.032    0.092   -0.079    0.007

   Pulse31  Pulse32  Pulse33  Pulse34  @outputs Class
0    0.423   -0.545    0.186   -0.453               g
1   -0.166   -0.063   -0.137   -0.024               b
2    0.257    1.000   -0.324    1.000               b
3   -0.057   -0.596   -0.046   -0.657               g
4    0.000    0.000    0.000    0.120               b
```

```
[5 rows x 34 columns]
```

[7]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 351 entries, 0 to 350
Data columns (total 34 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   @inputs Pulse1   351 non-null    int64
 1   Pulse3           351 non-null    float64
 2   Pulse4           351 non-null    float64
 3   Pulse5           351 non-null    float64
 4   Pulse6           351 non-null    float64
 5   Pulse7           351 non-null    float64
 6   Pulse8           351 non-null    float64
 7   Pulse9           351 non-null    float64
 8   Pulse10          351 non-null    float64
 9   Pulse11          351 non-null    float64
 10  Pulse12          351 non-null    float64
 11  Pulse13          351 non-null    float64
 12  Pulse14          351 non-null    float64
 13  Pulse15          351 non-null    float64
 14  Pulse16          351 non-null    float64
 15  Pulse17          351 non-null    float64
 16  Pulse18          351 non-null    float64
 17  Pulse19          351 non-null    float64
 18  Pulse20          351 non-null    float64
 19  Pulse21          351 non-null    float64
 20  Pulse22          351 non-null    float64
 21  Pulse23          351 non-null    float64
 22  Pulse24          351 non-null    float64
 23  Pulse25          351 non-null    float64
 24  Pulse26          351 non-null    float64
 25  Pulse27          351 non-null    float64
 26  Pulse28          351 non-null    float64
 27  Pulse29          351 non-null    float64
 28  Pulse30          351 non-null    float64
 29  Pulse31          351 non-null    float64
 30  Pulse32          351 non-null    float64
 31  Pulse33          351 non-null    float64
 32  Pulse34          351 non-null    float64
 33  @outputs Class   351 non-null    object
dtypes: float64(32), int64(1), object(1)
memory usage: 93.4+ KB
```

[8]:
```python
data = data.rename(columns={"@inputs Pulse1": "Pulse1", "@outputs Class":
 ↪"Class"})
```

[9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 351 entries, 0 to 350
Data columns (total 34 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Pulse1   351 non-null    int64
 1   Pulse3   351 non-null    float64
 2   Pulse4   351 non-null    float64
 3   Pulse5   351 non-null    float64
 4   Pulse6   351 non-null    float64
 5   Pulse7   351 non-null    float64
 6   Pulse8   351 non-null    float64
 7   Pulse9   351 non-null    float64
 8   Pulse10  351 non-null    float64
 9   Pulse11  351 non-null    float64
 10  Pulse12  351 non-null    float64
 11  Pulse13  351 non-null    float64
 12  Pulse14  351 non-null    float64
 13  Pulse15  351 non-null    float64
 14  Pulse16  351 non-null    float64
 15  Pulse17  351 non-null    float64
 16  Pulse18  351 non-null    float64
 17  Pulse19  351 non-null    float64
 18  Pulse20  351 non-null    float64
 19  Pulse21  351 non-null    float64
 20  Pulse22  351 non-null    float64
 21  Pulse23  351 non-null    float64
 22  Pulse24  351 non-null    float64
 23  Pulse25  351 non-null    float64
 24  Pulse26  351 non-null    float64
 25  Pulse27  351 non-null    float64
 26  Pulse28  351 non-null    float64
 27  Pulse29  351 non-null    float64
 28  Pulse30  351 non-null    float64
 29  Pulse31  351 non-null    float64
 30  Pulse32  351 non-null    float64
 31  Pulse33  351 non-null    float64
 32  Pulse34  351 non-null    float64
 33  Class    351 non-null    object
dtypes: float64(32), int64(1), object(1)
memory usage: 93.4+ KB
```

[10]:
```
# 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

[11]:
```
# Train and evaluate a Logistic Regression model
lr_model = LogisticRegression(random_state=42)
```

[12]:
```
lr_model.fit(X_train, y_train)
lr_preds = lr_model.predict(X_test)
```

[13]:
```
lr_acc = accuracy_score(y_test, lr_preds)
lr_prec = precision_score(y_test, lr_preds, pos_label=' b')
lr_rec = recall_score(y_test, lr_preds, pos_label=' b')
lr_f1 = f1_score(y_test, lr_preds, pos_label=' b')
```

[14]:
```
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: 0.8028169014084507
Logistic Regression Precision: 0.8421052631578947
Logistic Regression Recall: 0.5925925925925926
Logistic Regression F1 Score: 0.6956521739130435
```

[15]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import numpy as np
import itertools
```

[16]:
```
# Define class labels
classes = ['Class A', 'Class B']
```

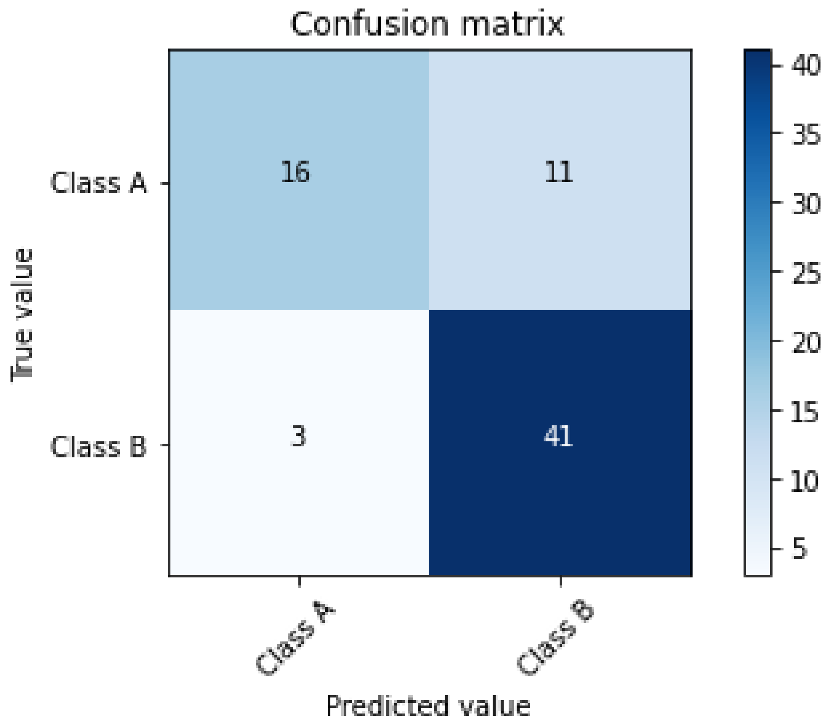
[17]:
```
# Compute confusion matrix
cm = confusion_matrix(y_test, lr_preds)
```

[18]:
```
# 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

[19]:
```
# Train and evaluate a Decision Tree model
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
```

[19]: DecisionTreeClassifier()

[20]:
```
dt_preds = dt_model.predict(X_test)
dt_acc = accuracy_score(y_test, dt_preds)
dt_prec = precision_score(y_test, dt_preds, pos_label=' g')
dt_rec = recall_score(y_test, dt_preds, pos_label=' g')
dt_f1 = f1_score(y_test, dt_preds, pos_label=' g')
```

[21]:
```
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: 0.8591549295774648
Decision Tree Precision: 0.8863636363636364
Decision Tree Recall: 0.8863636363636364
Decision Tree F1 Score: 0.8863636363636365
```

[22]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import LabelEncoder
```

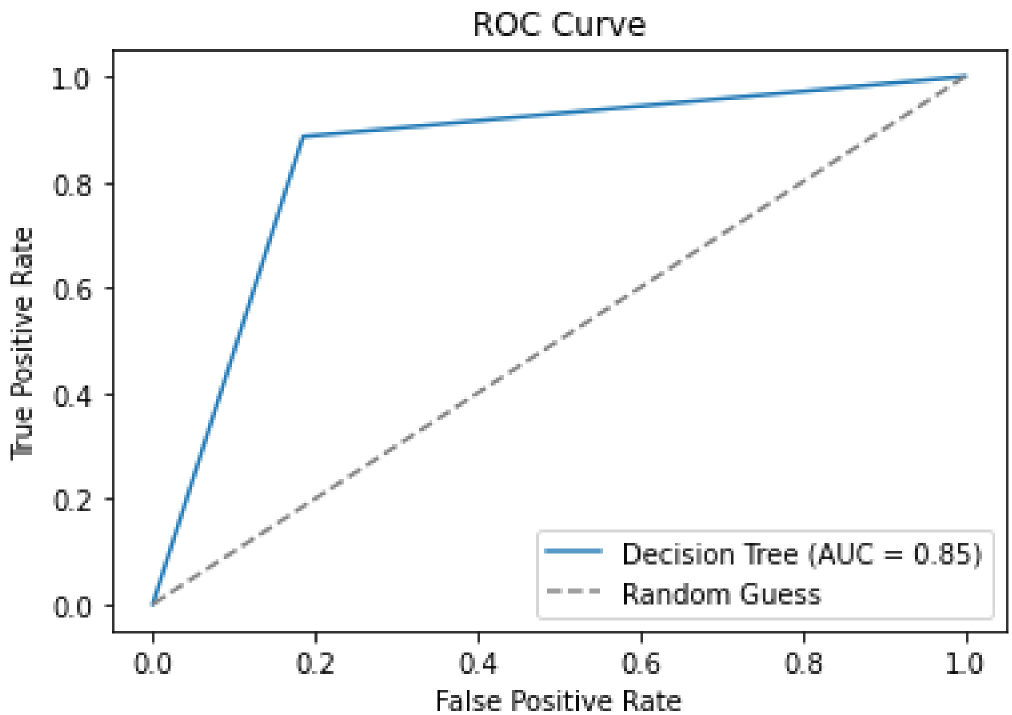
[23]:
```
# Calculate predicted probabilities for positive class
dt_probs = dt_model.predict_proba(X_test)[:, 1]
```

[24]:
```
y_true_binary = (y_test == ' g').astype(int)
```

[25]:
```
# calculate FPR, TPR, and thresholds
fpr, tpr, thresholds = roc_curve(y_true_binary, dt_probs, pos_label=1)
```

[26]:
```
# Calculate AUC
auc_dt = auc(fpr, tpr)
```

[27]:
```
# 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

[28]:
```
gb_model = GradientBoostingClassifier(random_state=42)
gb_model.fit(X_train, y_train)
```

[28]:
```
GradientBoostingClassifier(random_state=42)
```

[29]:
```
gb_preds = gb_model.predict(X_test)
```

[30]:
```
gb_acc = accuracy_score(y_test, gb_preds)
gb_prec = precision_score(y_test, gb_preds, pos_label=' g')
gb_rec = recall_score(y_test, gb_preds, pos_label=' g')
gb_f1 = f1_score(y_test, gb_preds, pos_label=' g')
```

[36]:
```
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: 0.9014084507042254
Gradient Boosting Precision: 0.8936170212765957
```

```
Gradient Boosting Recall: 0.9545454545454546
Gradient Boosting F1 Score: 0.9230769230769231
```