# Pima dataset classification

March 25, 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('pima.csv')
```

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

[5]:
```
   @inputs Preg  Plas  Pres  Skin  Insu  Mass   Pedi  Age   @outputs Class
0            14   175    62    30     0  33.6  0.212   38  tested_positive
1             4   146    78     0     0  38.5  0.520   67  tested_positive
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   @inputs Preg    768 non-null    int64
 1   Plas            768 non-null    int64
 2   Pres            768 non-null    int64
 3   Skin            768 non-null    int64
 4   Insu            768 non-null    int64
 5   Mass            768 non-null    float64
 6   Pedi            768 non-null    float64
 7   Age             768 non-null    int64
 8   @outputs Class  768 non-null    object
dtypes: float64(2), int64(6), object(1)
memory usage: 54.1+ KB
```

[7]:
```python
data = data.rename(columns={"@inputs Preg": "Preg", "@outputs Class":"Class"})
```

[8]:
```python
data1 = data
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Preg    768 non-null    int64
 1   Plas    768 non-null    int64
 2   Pres    768 non-null    int64
 3   Skin    768 non-null    int64
 4   Insu    768 non-null    int64
 5   Mass    768 non-null    float64
 6   Pedi    768 non-null    float64
 7   Age     768 non-null    int64
 8   Class   768 non-null    object
dtypes: float64(2), int64(6), object(1)
memory usage: 54.1+ KB
```

[10]:
```python
data.tail(10)
```

[10]:

| | Preg | Plas | Pres | Skin | Insu | Mass | Pedi | Age | Class |
|---|---|---|---|---|---|---|---|---|---|
| 758 | 1 | 86 | 66 | 52 | 65 | 41.3 | 0.917 | 29 | tested_negative |
| 759 | 1 | 109 | 60 | 8 | 182 | 25.4 | 0.947 | 21 | tested_negative |
| 760 | 2 | 100 | 68 | 25 | 71 | 38.5 | 0.324 | 26 | tested_negative |
| 761 | 6 | 114 | 88 | 0 | 0 | 27.8 | 0.247 | 66 | tested_negative |
| 762 | 6 | 92 | 92 | 0 | 0 | 19.9 | 0.188 | 28 | tested_negative |
| 763 | 5 | 117 | 92 | 0 | 0 | 34.1 | 0.337 | 38 | tested_negative |
| 764 | 4 | 83 | 86 | 19 | 0 | 29.3 | 0.317 | 34 | tested_negative |
| 765 | 7 | 119 | 0 | 0 | 0 | 25.2 | 0.209 | 37 | tested_negative |
| 766 | 1 | 95 | 66 | 13 | 38 | 19.6 | 0.334 | 25 | tested_negative |
| 767 | 1 | 181 | 64 | 30 | 180 | 34.1 | 0.328 | 38 | tested_positive |

[11]:
```python
# 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

[12]:
```python
# Train and evaluate a Logistic Regression model
lr_model = LogisticRegression(random_state=42)
```

[13]:
```python
lr_model.fit(X_train, y_train)
lr_preds = lr_model.predict(X_test)
```

[14]:
```
lr_acc = accuracy_score(y_test, lr_preds)
lr_prec = precision_score(y_test, lr_preds, pos_label='tested_positive')
lr_rec = recall_score(y_test, lr_preds, pos_label='tested_positive')
lr_f1 = f1_score(y_test, lr_preds, pos_label='tested_positive')
```

[15]:
```
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: 0.7922077922077922
Logistic Regression Precision: 0.7555555555555555
Logistic Regression Recall: 0.6181818181818182
Logistic Regression F1 Score: 0.6799999999999999
```

[16]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import numpy as np
import itertools
```

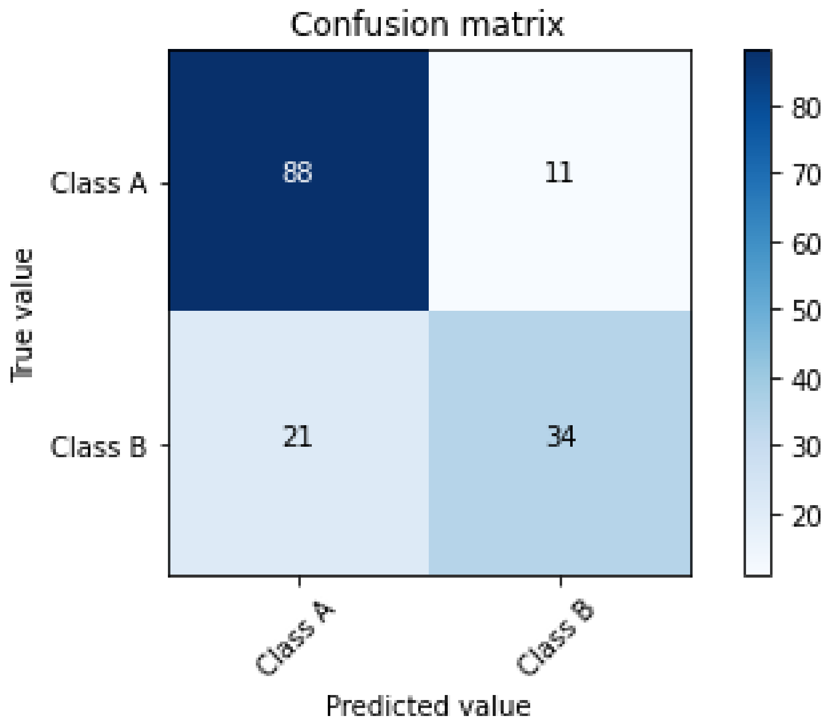
[17]:
```
# Define class labels
classes = ['Class A', 'Class B']
```

[18]:
```
# Compute confusion matrix
cm = confusion_matrix(y_test, lr_preds)
```

[19]:
```
# 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

[20]:
```
# Train and evaluate a Decision Tree model
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
```

[20]: DecisionTreeClassifier()

[21]:
```
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='tested_positive')
dt_rec = recall_score(y_test, dt_preds, pos_label='tested_positive')
dt_f1 = f1_score(y_test, dt_preds, pos_label='tested_positive')
```

[22]:
```
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: 0.7207792207792207
Decision Tree Precision: 0.6071428571428571
Decision Tree Recall: 0.6181818181818182
```

Decision Tree F1 Score: 0.6126126126126126

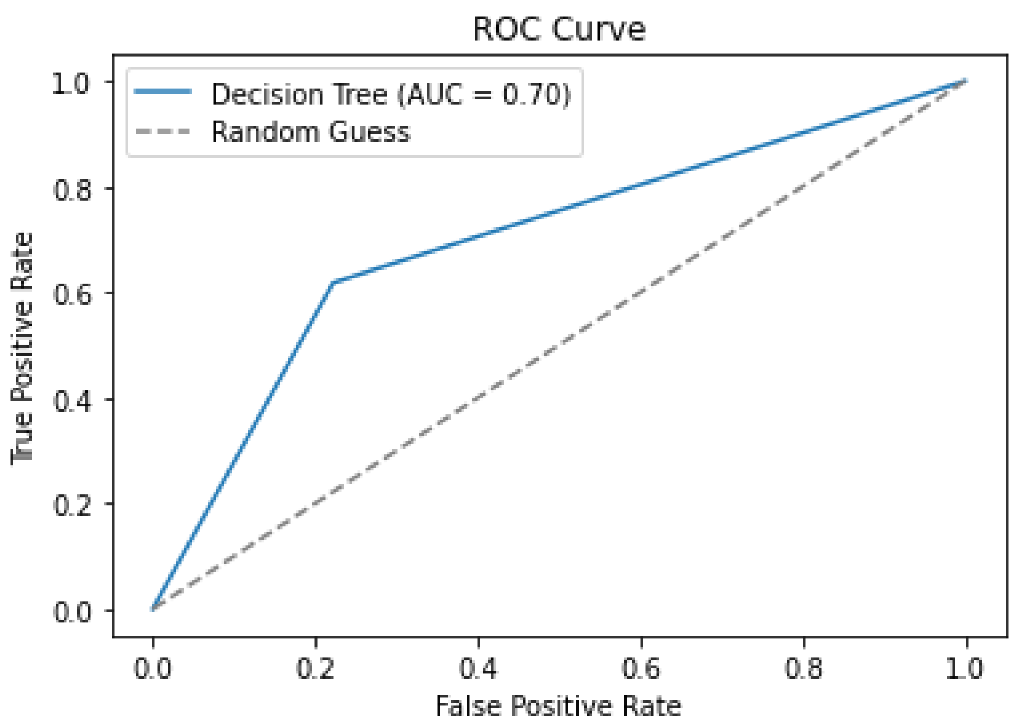
[23]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
```

[24]:
```
# Calculate predicted probabilities for positive class
dt_probs = dt_model.predict_proba(X_test)[:, 1]
```

[25]:
```
# Calculate FPR, TPR, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, dt_probs, pos_label='tested_positive')
```

[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()
gb_model.fit(X_train, y_train)
```

[28]:
```
GradientBoostingClassifier()
```

[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='tested_positive')
gb_rec = recall_score(y_test, gb_preds, pos_label='tested_positive')
gb_f1 = f1_score(y_test, gb_preds, pos_label='tested_positive')
```

[31]:
```
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: 0.8246753246753247
Gradient Boosting Precision: 0.7692307692307693
Gradient Boosting Recall: 0.7272727272727273
Gradient Boosting F1 Score: 0.7476635514018691
```