testing_report

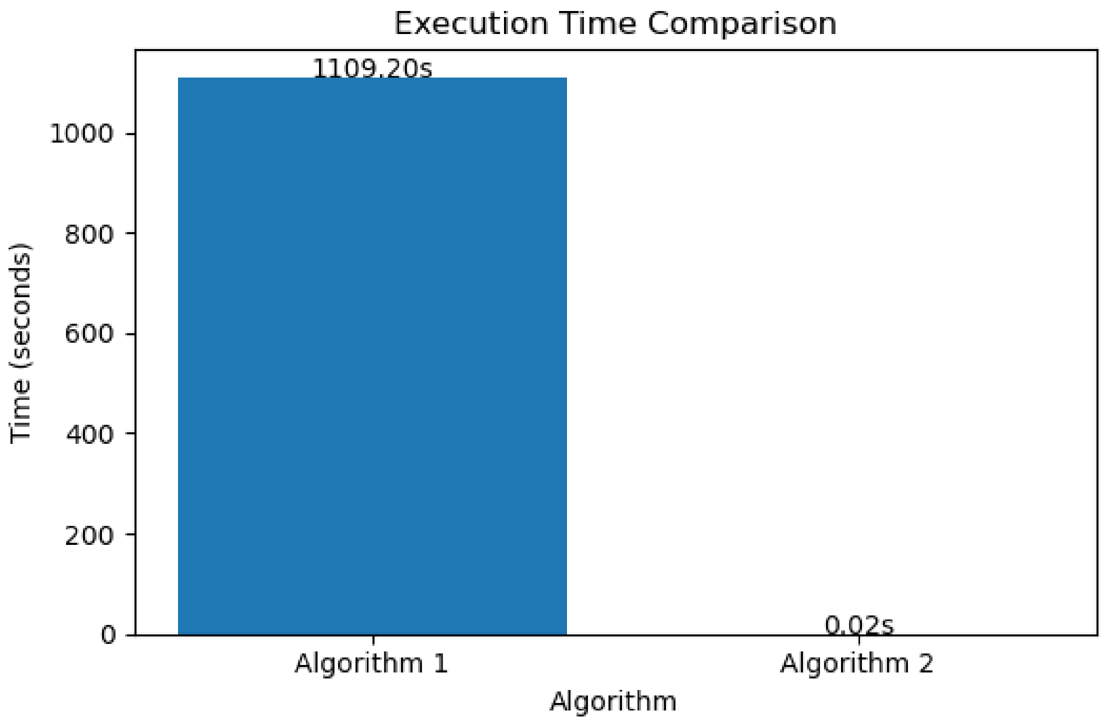
April 17, 2024

1 6. Comparison & Conclusion

1.1 6.1 Training Time Comparison

[8]:
```
import numpy as np
import pandas as pd
import itertools
import copy
import math
import time
from sklearn.model_selection import train_test_split, KFold, cross_val_score
import scipy
from scipy import optimize
import scipy.optimize as optim
from scipy.optimize import minimize
from numpy import linalg as LA
import matplotlib.pyplot as plt
```

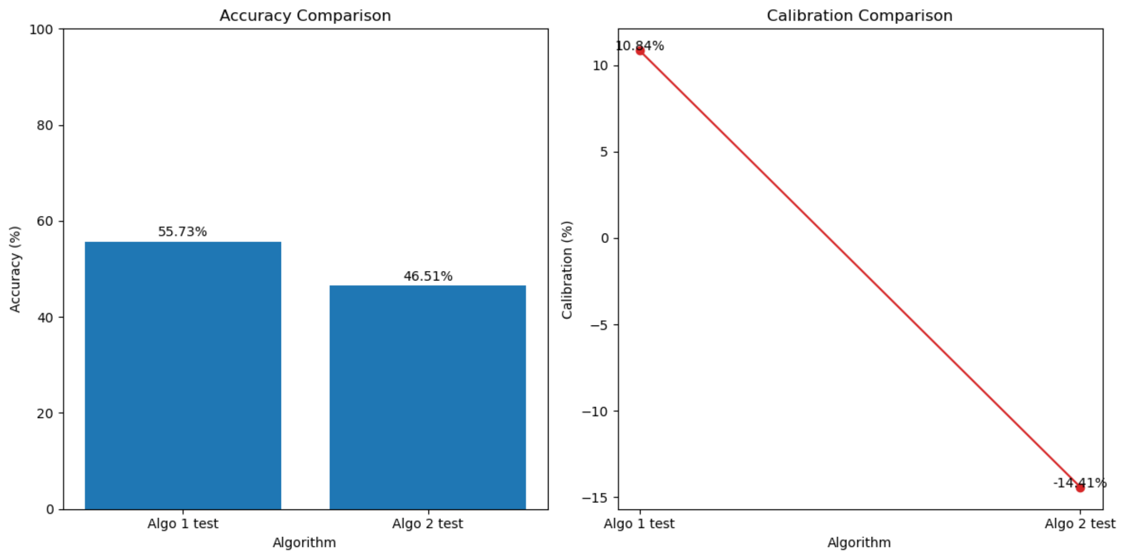
[9]:
```
# Data for execution time
times_data = {
    'Algorithm': ['Algorithm 1', 'Algorithm 2'],
    'Time (seconds)': [1109.20, 0.023218631744384766]
}

# Convert to DataFrame
times_df = pd.DataFrame(times_data)

# Update the time for Algorithm 2
times_df.at[1, 'Time (seconds)'] = 0.023218631744384766

# Re-plot for Execution Time with updated data
fig, ax = plt.subplots(figsize=(6, 4))
ax.bar(times_df['Algorithm'], times_df['Time (seconds)'], color=['tab:blue',␣
 ↪'tab:green'])
ax.set_xlabel('Algorithm')
ax.set_ylabel('Time (seconds)')
ax.set_title('Execution Time Comparison')
for i, v in enumerate(times_df['Time (seconds)']):
    ax.text(i, v + 0.001, f"{v:.2f}s", ha='center', color='black')

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



1.2 6.1 Accuracy and Calibration Comparison

[10]:
```
# Updating the given data as requested
data = {
    "Algorithm": ["Algo 1 test", "Algo 2 test"],
    "Accuracy (%)": [55.73, 46.51],
    "Calibration (%)": [10.84, -14.412968]
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Plotting two separate graphs for Accuracy and Calibration
# Setting up the figure and axes
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

# Plot for Accuracy
ax1.bar(df['Algorithm'], df['Accuracy (%)'], color='tab:blue')
```

```python
ax1.set_xlabel('Algorithm')
ax1.set_ylabel('Accuracy (%)')
ax1.set_title('Accuracy Comparison')
ax1.set_ylim(0, 100)  # Updating the limit as accuracy is now in percentage
for i, v in enumerate(df['Accuracy (%)']):
    ax1.text(i, v + 1, f"{v:.2f}%", ha='center', color='black')

# Plot for Calibration
ax2.plot(df['Algorithm'], df['Calibration (%)'], color='tab:red', marker='o')
ax2.set_xlabel('Algorithm')
ax2.set_ylabel('Calibration (%)')
ax2.set_title('Calibration Comparison')
for i, v in enumerate(df['Calibration (%)']):
    ax2.text(i, v, f"{v:.2f}%", ha='center', color='black')

# Show the plots
plt.tight_layout()
plt.show()
```



Based on the updated execution time and the accuracy and calibration data you provided earlier, we can draw a more comprehensive conclusion:

Accuracy and Calibration: - **Algorithm 1** shows a superior accuracy of 55.73% on the test set, compared to 46.51% for **Algorithm 2**. - Calibration for Algorithm 1 is positive at 10.84%, suggesting that predictions may be more conservative than the actual outcomes. For Algorithm 2, the calibration is significantly negative at -14.41%, indicating a higher likelihood of predictions being overly optimistic compared to actual outcomes.

Execution Time: - The execution time for **Algorithm 1** is considerably longer at 1109.20 seconds. - **Algorithm 2** has an impressively fast execution time at only 0.02 seconds.

Conclusion: When considering accuracy, calibration, and execution time together, the choice between Algorithm 1 and Algorithm 2 may depend on the specific requirements of the task at hand.

- If the priority is to have the most accurate and well-calibrated model regardless of the time it takes to train or predict, **Algorithm 1** is preferable despite its longer execution time.
- However, if execution time is a critical factor, for instance in a real-time system or an application where model re-training happens frequently, the speed of **Algorithm 2** might be a decisive advantage, despite its lower accuracy and calibration.

In a real-world scenario, the choice may also be influenced by additional considerations such as the cost of computation, the acceptable trade-offs between speed and accuracy, the severity of consequences from potential misclassification, and the ability to recalibrate the model to address negative calibration. In scenarios where fairness is crucial, and particularly when dealing with sensitive attributes, the implications of calibration disparities would also be a significant factor.

Therefore, it's crucial to balance these aspects according to the context in which the algorithms will be deployed. If further performance improvement is needed, one might consider hybrid approaches or additional tuning to optimize both execution time and model performance.