Lab Manual: NLP Text Classification with Quantum ML

---

1. Theoretical Background

- **Natural Language Processing (NLP)**: A subfield of artificial intelligence (AI) that focuses on the interaction between computers and humans through natural language. Text classification is a common NLP task that involves assigning predefined categories to text documents.
- **Classical Machine Learning (ML)**: Traditional algorithms (like Logistic Regression, SVM, etc.) used for classification tasks based on statistical methods and heuristics.
- **Quantum Machine Learning (QML)**: An emerging field that combines quantum computing with machine learning. QML algorithms can leverage quantum properties like superposition and entanglement to process data in ways that classical computers cannot.
- **Hybrid Models**: Combining classical and quantum models to utilize the strengths of both methodologies for improved performance.

---

2. Data Preparation

1. **Dataset**: For this example, we will use the `20 Newsgroups` dataset from `scikit-learn`, which contains around 20,000 newsgroup documents, organized into 20 different newsgroups.
2. **Preprocessing**: The text data will be preprocessed to convert it into a format suitable for classification, including tokenization, vectorization, and removing stop words.

---

3. Pipeline

1. Load the dataset.
2. Preprocess the text data.
3. Split the data into training and test sets.
4. Train a classical ML model (e.g., Logistic Regression).
5. Train a quantum ML model.
6. Combine predictions from both models.
7. Evaluate and compare the results.

---

4. Pseudocode

```
Load Dataset
Preprocess Text Data
Split Data into Training and Testing Sets
```

```
Train Classical Model
Make Predictions using Classical Model
Train Quantum Model
Make Predictions using Quantum Model
Combine Predictions from both Models
Evaluate and Compare Model Accuracies
```

---

5. Implementation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from qiskit import QuantumCircuit
from qiskit_aer import AerSimulator

# Step 1: Load the dataset
newsgroups = fetch_20newsgroups(subset='all', categories=['comp.graphics',
'sci.space'])
X = newsgroups.data
y = newsgroups.target

# Step 2: Preprocess the text data
vectorizer = TfidfVectorizer(stop_words='english')
X_vectorized = vectorizer.fit_transform(X)

# Step 3: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_vectorized, y,
test_size=0.25, random_state=42)

# Step 4: Train a classical model (Logistic Regression)
classical_model = LogisticRegression(max_iter=1000)
classical_model.fit(X_train, y_train)

# Step 5: Predictions and evaluation for classical model
y_pred_classical = classical_model.predict(X_test)
classical_accuracy = accuracy_score(y_test, y_pred_classical)
print("Classical Model Accuracy:", classical_accuracy)

# Step 6: Create a quantum circuit for QML
def create_quantum_circuit(num_qubits):
    circuit = QuantumCircuit(num_qubits, num_qubits)
    for i in range(num_qubits):
        circuit.h(i)  # Apply Hadamard to each qubit
    circuit.measure(range(num_qubits), range(num_qubits))  # Measure each
qubit
    return circuit

# Step 7: Run the quantum circuit and evaluate
def run_circuit(qc):
    simulator = AerSimulator()
```

```python
    result = simulator.run(qc).result()  # Directly run the circuit
    counts = result.get_counts()
    return counts

# Simulate predictions based on the quantum circuit
def quantum_predictions(X_test):
    predictions = []
    for i in range(len(X_test)):
        num_qubits = 2  # Adjust based on your classification needs
        qc = create_quantum_circuit(num_qubits)
        counts = run_circuit(qc)
        result = max(counts, key=counts.get)  # Get the most frequent
measurement result
        label = int(result, 2) % 2  # Simplified prediction
        predictions.append(label)
    return predictions

# Step 8: Get quantum predictions
y_pred_quantum = quantum_predictions(X_test)

# Step 9: Evaluate quantum model accuracy
quantum_accuracy = accuracy_score(y_test, y_pred_quantum)
print("Quantum Model Accuracy:", quantum_accuracy)

# Step 10: Combine classical and quantum predictions
def combine_predictions(classical_preds, quantum_preds):
    combined_preds = []
    for classical, quantum in zip(classical_preds, quantum_preds):
        # Simple combination: prioritize classical predictions
        combined_preds.append(classical)  # Can implement weighted voting
    return np.array(combined_preds)

# Step 11: Get combined predictions
y_pred_combined = combine_predictions(y_pred_classical, y_pred_quantum)

# Step 12: Evaluate combined model accuracy
combined_accuracy = accuracy_score(y_test, y_pred_combined)
print("Combined Model Accuracy:", combined_accuracy)

# Step 13: Print classification reports for all models
print("\nClassification Report for Classical Model:")
print(classification_report(y_test, y_pred_classical))

print("\nClassification Report for Quantum Model:")
print(classification_report(y_test, y_pred_quantum))

print("\nClassification Report for Combined Model:")
print(classification_report(y_test, y_pred_combined))

# Step 14: Plotting results
def plot_results(classical_accuracy, quantum_accuracy, combined_accuracy):
    models = ['Classical ML', 'Quantum ML', 'Combined Model']
    accuracies = [classical_accuracy, quantum_accuracy, combined_accuracy]

    plt.bar(models, accuracies, color=['blue', 'orange', 'green'])
    plt.title('Model Accuracies Comparison')
    plt.ylabel('Accuracy')
```

```
    plt.ylim(0, 1)  # Set y-axis limits to [0,1]
    plt.show()

# Plot the results
plot_results(classical_accuracy, quantum_accuracy, combined_accuracy)
```

---

6. Comparisons

- **Classical Model**: Generally more robust with higher accuracy, well-tested algorithms (like Logistic Regression).
- **Quantum Model**: Still in development; may not perform as well as classical models currently. Limited by the depth of the quantum circuit and the number of qubits.
- **Combined Model**: Aims to leverage the strengths of both models. Depending on the approach to combining predictions, it can yield improved accuracy.

---

7. Pros and Cons

Approach	Pros	Cons
Classical ML	- Well-established algorithms	- May not harness complex data patterns
	- High accuracy on many datasets	
Quantum ML	- Potential for faster processing	- Limited by current hardware
	- Can handle high-dimensional data uniquely	- Requires more research for effective algorithms
Combined Model	- Leverages strengths of both models	- Complexity in implementation
	- Can outperform individual models	- May require tuning for optimal performance

---

This lab manual provides a structured approach to implementing NLP text classification using both classical and quantum methods, along with a hybrid strategy for combining their outputs.