Quantum Optimization with Classical Machine Learning

This document explains in-depth how classical and quantum optimization techniques are applied in machine learning, specifically in the context of breast cancer classification using Support Vector Machines (SVM). The code presented combines both classical machine learning with quantum computing techniques using Qiskit.

1. Data Preparation

Loading the Breast Cancer Dataset

```
from sklearn import datasets
from sklearn.model_selection import train_test_split

# Load the breast cancer dataset
cancer = datasets.load_breast_cancer()
X, y = cancer.data, cancer.target
```

Explanation:

- **Breast Cancer Dataset**: The code uses the popular **Breast Cancer** dataset from `scikit-learn`. This dataset contains features related to breast cancer tumor characteristics.
- **Features (X)**: The characteristics of the cancer (e.g., size, shape, etc.).
- **Labels (y)**: Binary classification (1 for malignant, 0 for benign).

Splitting the Data

```
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

- **Training/Test Split**: The dataset is split into training (80%) and testing (20%) sets using `train_test_split`. The **random_state** ensures reproducibility.

2. Classical Optimization: SVM with Grid Search

SVM Model and Hyperparameter Tuning

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Define parameter grid for C values and kernel types
param_grid = {'C': [0.1, 1, 10, 100], 'kernel': ['linear']}
grid_search = GridSearchCV(SVC(), param_grid, refit=True, verbose=1)
grid_search.fit(X_train, y_train)

# Evaluate the best model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

Explanation:

- **Support Vector Machine (SVM)**: This is a popular supervised learning algorithm used for classification tasks.
- **GridSearchCV**: This technique is used to optimize hyperparameters of the SVM model. Here, it searches for the best value of the regularization parameter **C** and uses a **linear kernel**.
- **C Parameter**: This controls the trade-off between achieving a low training error and a low testing error.
- **Best Model**: After training, the best model is selected based on the highest accuracy on the validation set.

Evaluating Accuracy

```
print(f"Najlepsze parametry (klasyczna optymalizacja):
{grid_search.best_params_}")
print(f"Dokładność klasycznego SVM: {accuracy_score(y_test, y_pred) *
100:.2f}%")
```

- **Accuracy**: The accuracy of the classical SVM is evaluated on the test set, which gives insight into how well the model generalizes to unseen data.

3. Quantum Optimization: Grover's Algorithm

Custom Oracle for Quantum Circuit

```
from qiskit import QuantumCircuit

def custom_oracle(num_qubits):
    qc = QuantumCircuit(num_qubits)
    qc.cz(0, 1)  # Phase flip for state |11>
    return qc
```

Explanation:

- **Grover's Algorithm**: This is a quantum search algorithm used to find a specific element from an unstructured list. It can also be used for optimization.
- **Oracle**: The oracle is a quantum subroutine that marks the correct solution (or optimal value) by applying a phase flip (inverting the amplitude of the correct answer).
- **Custom Oracle**: In this case, the oracle flips the phase when the qubits are in the state |11>, marking it as the solution.

Grover's Circuit

```
qc = QuantumCircuit(num_qubits)
qc.h([0, 1])  # Initialize qubits in superposition
qc.compose(oracle_circuit, inplace=True)
qc.h([0, 1])
qc.measure_all()
```

- **Superposition**: All qubits are initialized in superposition using Hadamard gates, allowing the quantum computer to explore all possible states simultaneously.
- **Quantum Circuit Composition**: The oracle circuit is composed (added) to the main Grover's circuit.
- **Measurement**: The quantum circuit is measured to extract the optimal state after running the quantum algorithm.

Simulation of Quantum Circuit

```
from qiskit_aer import AerSimulator
from qiskit import transpile

backend = AerSimulator()
qc = transpile(qc, backend)
job = backend.run(qc, shots=1024)
result = job.result()
counts = result.get_counts()
```

- **AerSimulator**: This is a quantum simulator from Qiskit that simulates how quantum circuits would behave on real quantum hardware.
- **Transpilation**: The quantum circuit is optimized for the specific backend (in this case, the simulator).
- **Result**: The results of the quantum simulation are analyzed to determine which state (among |00|, |01|, |10|, and |11|) occurs the most frequently.

Mapping Quantum States to C Values

```
state_to_C = {
    '00': 0.1,
    '01': 1,
    '10': 10,
    '11': 100
}

optimal_state = max(counts, key=counts.get)
best_C_from_grover = state_to_C[optimal_state]
```

- **State Mapping**: The measured quantum states are mapped to possible values of **C** (regularization parameter). The state with the highest count is chosen as the optimal value.
- **Grover Optimization**: The best value of **C** is obtained through the quantum optimization process using Grover's algorithm.

4. Applying Quantum-Optimized C to SVM

```
svm_quantum = SVC(kernel='linear', C=best_C_from_grover)
svm_quantum.fit(X_train, y_train)
```

- **Quantum-Optimized SVM**: The value of **C** found using Grover's algorithm is used to train a new SVM model. This showcases a hybrid approach where quantum algorithms assist in hyperparameter optimization for classical machine learning models.

Cross-Validation

```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(svm_quantum, X, y, cv=5)
```

- **Cross-Validation**: The quantum-optimized SVM model is evaluated using 5-fold cross-validation, which splits the data into 5 parts and trains the model on different subsets to ensure generalization.

5. Visualization of Results

Plotting the Data

```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

plt.figure(figsize=(14, 6))

# Subplot 1: Scatter plot of features
plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(['red', 'green',
'blue']), edgecolor='k')
plt.title('Breast Cancer Data - Feature Distribution')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

# Subplot 2: Accuracy Comparison
plt.subplot(1, 2, 2)
plt.bar(['Classical SVM', 'Quantum-Optimized SVM'],
        [accuracy_score(y_test, y_pred) * 100, np.max(cv_scores) * 100],
        color=['orange', 'purple'])
plt.title('Accuracy Comparison')
plt.ylabel('Accuracy (%)')
plt.ylim(90, 100)

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

- **Visualization**: Two visualizations are created:
    1. A scatter plot showing the distribution of the cancer data in two feature dimensions.
    2. A bar plot comparing the accuracy of the classical SVM model and the quantum-optimized SVM.

Conclusion

This code demonstrates the use of **quantum computing** to optimize machine learning models. Specifically, **Grover's Algorithm** helps to find the best hyperparameters for the **SVM** model by exploring possible values in a quantum-enhanced manner. The classical **GridSearchCV** is compared to quantum optimization, showing the potential for hybrid quantum-classical algorithms in real-world machine learning tasks.