Enhanced Lab08: Logistic Regression

Clarification Concepts for Logistic Regression

1. Logistic Regression Basics

What is Logistic Regression?

- Logistic regression is a classification algorithm used to predict categorical outcomes (e.g., spam or not spam)
- It estimates the probability that an instance belongs to a certain class using the sigmoid function

Why Not Use Linear Regression for Classification?

- Linear regression predicts continuous values, which can go beyond 0 and 1
- Logistic regression ensures outputs are bounded between 0 and 1, making it suitable for classification

2. The Sigmoid Function

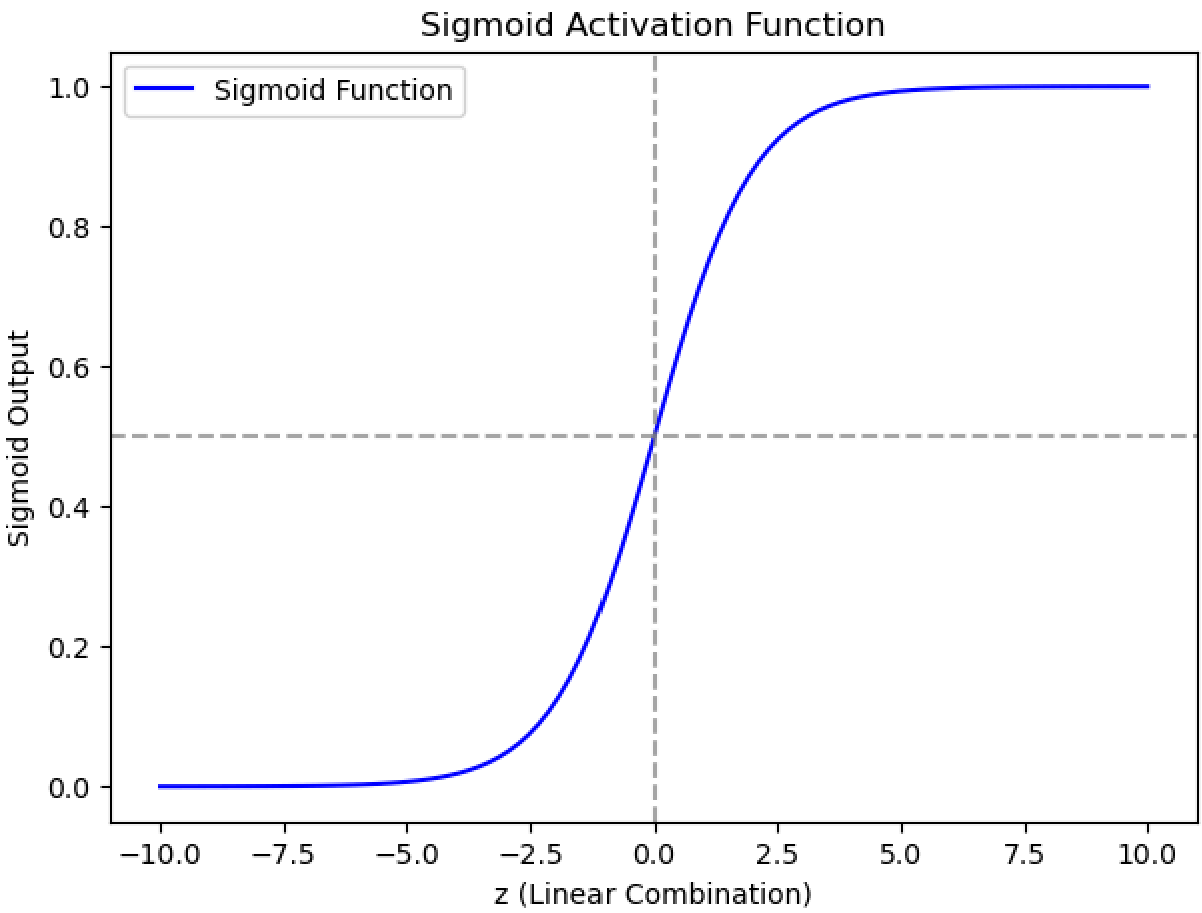
Formula:

σ(z) = 1/(1 + e^(-z))

where z = wX + b

```
import numpy as np
import matplotlib.pyplot as plt

# Define sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Generate values for z
z = np.linspace(-10, 10, 100)
sigma = sigmoid(z)

# Plot sigmoid function
plt.figure(figsize=(7, 5))
plt.plot(z, sigma, label="Sigmoid Function", color='b')
```

```
plt.axvline(0, color='gray', linestyle='--', alpha=0.7)
plt.axhline(0.5, color='gray', linestyle='--', alpha=0.7)
plt.xlabel("z (Linear Combination)")
plt.ylabel("Sigmoid Output")
plt.title("Sigmoid Activation Function")
plt.legend()
plt.show()
```



2. Advanced Concepts For Logistics Regression

2.1 Binary Classification with Regularization (L1/L2) with Real-World Datsets

2.1.1 Import Real-World Dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, roc_curve, auc
```

```
from sklearn.preprocessing import StandardScaler

np.random.seed(42)

# 1. Data Loading and Preprocessing
print("1. Data Loading and Initial Analysis")
print("-" * 50)

# Load the dataset
df = pd.read_csv('../Dataset/Lab_Enhancement_Lab8/emails.csv')
print(f"Dataset shape: {df.shape}")
print("\nClass distribution:")
print(df['spam'].value_counts(normalize=True))
```

```
1. Data Loading and Initial Analysis
--------------------------------------------------
Dataset shape: (5728, 2)

Class distribution:
spam
0    0.761173
1    0.238827
Name: proportion, dtype: float64
```

2.1.2 Feature Extraction

```
# 2. Feature Extraction
print("\n2. Feature Extraction")
print("-" * 50)

# Create TF-IDF Vectorizer
tfidf = TfidfVectorizer(max_features=1000, stop_words='english')
X = tfidf.fit_transform(df['text'])
y = df['spam'].astype(int)
```

```

2. Feature Extraction
--------------------------------------------------
```

2.1.3 Sampling Techniques for Imbalance Data

```
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 2.1 Handling Imbalanced Data
print("\nHandling Imbalanced Data")
print("-" * 50)

from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTETomek
# Display original class distribution
```

```
print("Original class distribution in training set:")
print(pd.Series(y_train).value_counts(normalize=True))

# 1. SMOTE (Oversampling)
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
print("\nClass distribution after SMOTE:")
print(pd.Series(y_train_smote).value_counts(normalize=True))

# 2. Random Undersampling
undersampler = RandomUnderSampler(random_state=42)
X_train_under, y_train_under = undersampler.fit_resample(X_train, y_train)
print("\nClass distribution after Undersampling:")
print(pd.Series(y_train_under).value_counts(normalize=True))

# 3. SMOTE + Tomek (Combined approach)
smote_tomek = SMOTETomek(random_state=42)
X_train_smote_tomek, y_train_smote_tomek = smote_tomek.fit_resample(X_train, y_train)
print("\nClass distribution after SMOTE + Tomek:")
print(pd.Series(y_train_smote_tomek).value_counts(normalize=True))
```

```
2.1 Handling Imbalanced Data
--------------------------------------------------
Original class distribution in training set:
spam
0    0.764732
1    0.235268
Name: proportion, dtype: float64

Class distribution after SMOTE:
spam
0    0.5
1    0.5
Name: proportion, dtype: float64

Class distribution after Undersampling:
spam
0    0.5
1    0.5
Name: proportion, dtype: float64

Class distribution after SMOTE + Tomek:
spam
0    0.5
1    0.5
Name: proportion, dtype: float64
```

2.1.4 Model Evaluation

```
# Function to compare models with different sampling techniques
def train_and_evaluate_with_sampling(X_train, y_train, X_test, y_test, sampling_method):
    # Models to test
    models = {
```

```
        'Basic': LogisticRegression(random_state=42, max_iter=1000),
        'L1': LogisticRegression(penalty='l1', solver='liblinear', C=1.0, random_state=42
        'L2': LogisticRegression(penalty='l2', C=1.0, random_state=42, max_iter=1000)
    }

    results = {}
    for model_name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        results[model_name] = {
            'accuracy': accuracy_score(y_test, y_pred),
            'precision': precision_score(y_test, y_pred),
            'recall': recall_score(y_test, y_pred),
            'f1': f1_score(y_test, y_pred)
        }

    # Create comparison DataFrame
    results_df = pd.DataFrame(results).T
    print(f"\nResults with {sampling_method}:")
    print(results_df)
    return results_df
```

2.1.5 Comparison between sampling methods

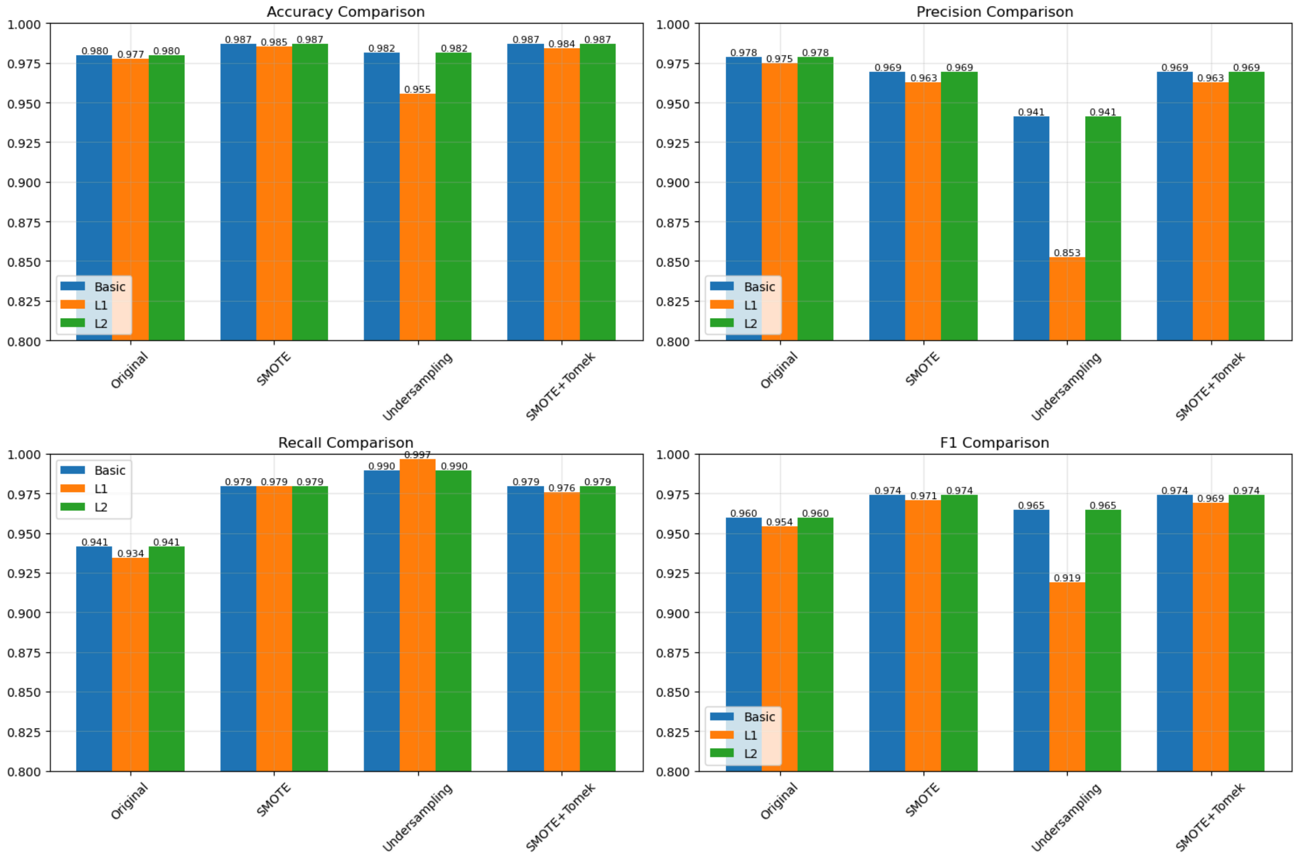
```
# Compare performance with different sampling methods
print("\nComparing model performance with different sampling methods:")
results_original = train_and_evaluate_with_sampling(X_train, y_train, X_test, y_test, "Or
results_smote = train_and_evaluate_with_sampling(X_train_smote, y_train_smote, X_test, y_
results_under = train_and_evaluate_with_sampling(X_train_under, y_train_under, X_test, y_
results_smote_tomek = train_and_evaluate_with_sampling(X_train_smote_tomek, y_train_smote
```

```
Comparing model performance with different sampling methods:

Results with Original Data:
       accuracy  precision    recall        f1
Basic  0.979930   0.978495  0.941379  0.959578
L1     0.977312   0.974820  0.934483  0.954225
L2     0.979930   0.978495  0.941379  0.959578

Results with SMOTE:
       accuracy  precision   recall        f1
Basic  0.986911   0.969283  0.97931  0.974271
L1     0.985166   0.962712  0.97931  0.970940
L2     0.986911   0.969283  0.97931  0.974271

Results with Undersampling:
       accuracy  precision    recall        f1
Basic  0.981675   0.940984  0.989655  0.964706
L1     0.955497   0.852507  0.996552  0.918919
L2     0.981675   0.940984  0.989655  0.964706

Results with SMOTE+Tomek:
       accuracy  precision    recall        f1
```

```
      Basic  0.986911   0.969283  0.979310  0.974271
      L1     0.984293   0.962585  0.975862  0.969178
      L2     0.986911   0.969283  0.979310  0.974271
```

2.1.6 Visualization of Comparison of Different Sampling Methods

```
# Visualize the comparison
metrics = ['accuracy', 'precision', 'recall', 'f1']
sampling_methods = ['Original', 'SMOTE', 'Undersampling', 'SMOTE+Tomek']
model_types = ['Basic', 'L1', 'L2']

plt.figure(figsize=(15, 10))
for i, metric in enumerate(metrics):
    plt.subplot(2, 2, i+1)

    x = np.arange(len(sampling_methods))
    width = 0.25

    for j, model_type in enumerate(model_types):
        metric_data = [
            results_original.loc[model_type, metric],
            results_smote.loc[model_type, metric],
            results_under.loc[model_type, metric],
            results_smote_tomek.loc[model_type, metric]
        ]

        bars = plt.bar(x + (j-1)*width, metric_data, width, label=model_type)

        # Add value labels on top of bars
        for bar in bars:
            height = bar.get_height()
            plt.text(bar.get_x() + bar.get_width()/2., height,
                    f'{height:.3f}',
                    ha='center', va='bottom', fontsize=8)

    plt.title(f'{metric.capitalize()} Comparison')
    plt.xticks(x, sampling_methods, rotation=45)
    plt.ylim(0.8, 1.0)  # Adjust y-axis range for better visualization
    plt.legend()
    plt.grid(True, alpha=0.3)

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




```
# Select the best sampling method based on F1 score
sampling_methods = {
    'Original': (X_train, y_train),
```

```
    'SMOTE': (X_train_smote, y_train_smote),
    'Undersampling': (X_train_under, y_train_under),
    'SMOTE+Tomek': (X_train_smote_tomek, y_train_smote_tomek)
}
```

2.1.7 Best Sampling Method

```
# Find best sampling method
best_f1_scores = {
    method: results_df['f1'].mean()
    for method, results_df in [
        ('Original', results_original),
        ('SMOTE', results_smote),
        ('Undersampling', results_under),
        ('SMOTE+Tomek', results_smote_tomek)
    ]
}

best_method = max(best_f1_scores.items(), key=lambda x: x[1])[0]
print(f"\nBest sampling method based on F1 score: {best_method}")
```

```
Best sampling method based on F1 score: SMOTE
```

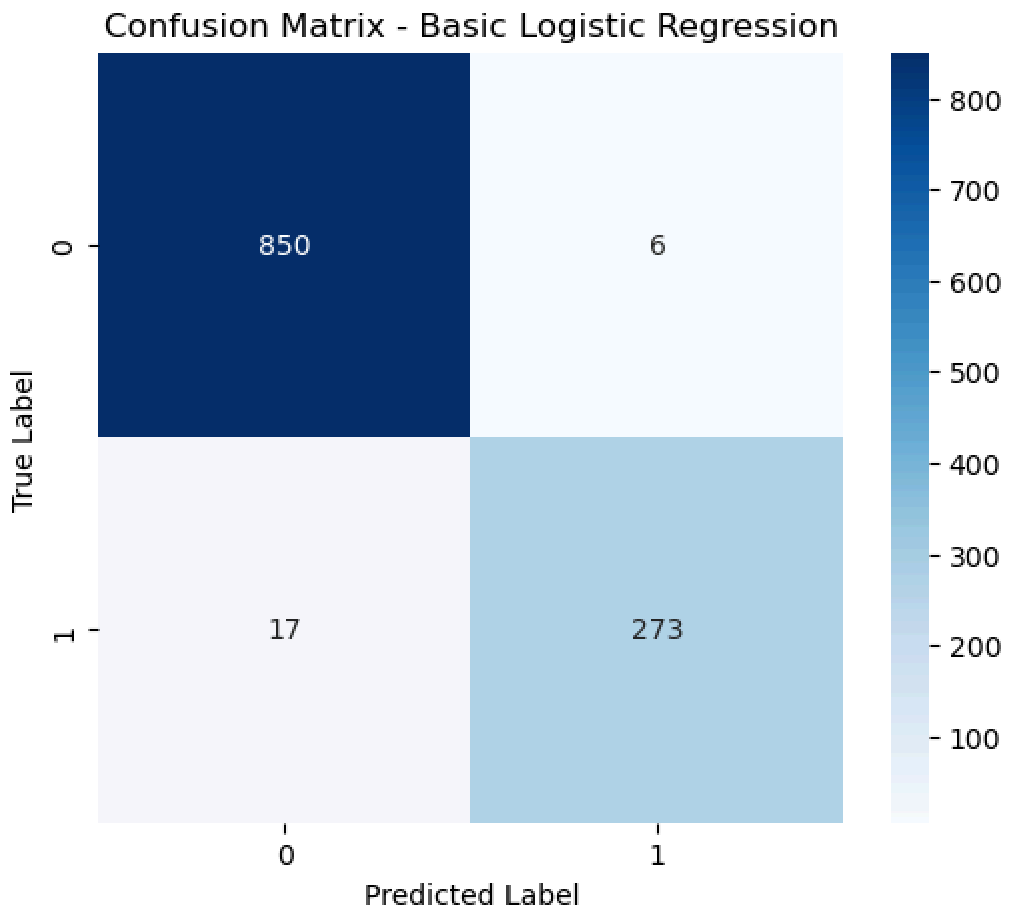
```
# Use the best sampling method for further modeling
X_train_balanced, y_train_balanced = sampling_methods[best_method]

# Function to evaluate and print metrics
def evaluate_model(model, X_test, y_test, model_name):
    y_pred = model.predict(X_test)

    print(f"\nMetrics for {model_name}:")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(f"Precision: {precision_score(y_test, y_pred):.4f}")
    print(f"Recall: {recall_score(y_test, y_pred):.4f}")
    print(f"F1 Score: {f1_score(y_test, y_pred):.4f}")

    # Plot confusion matrix
    plt.figure(figsize=(6, 5))
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix - {model_name}')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

    # Plot ROC curve
    y_pred_proba = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(6, 5))
```

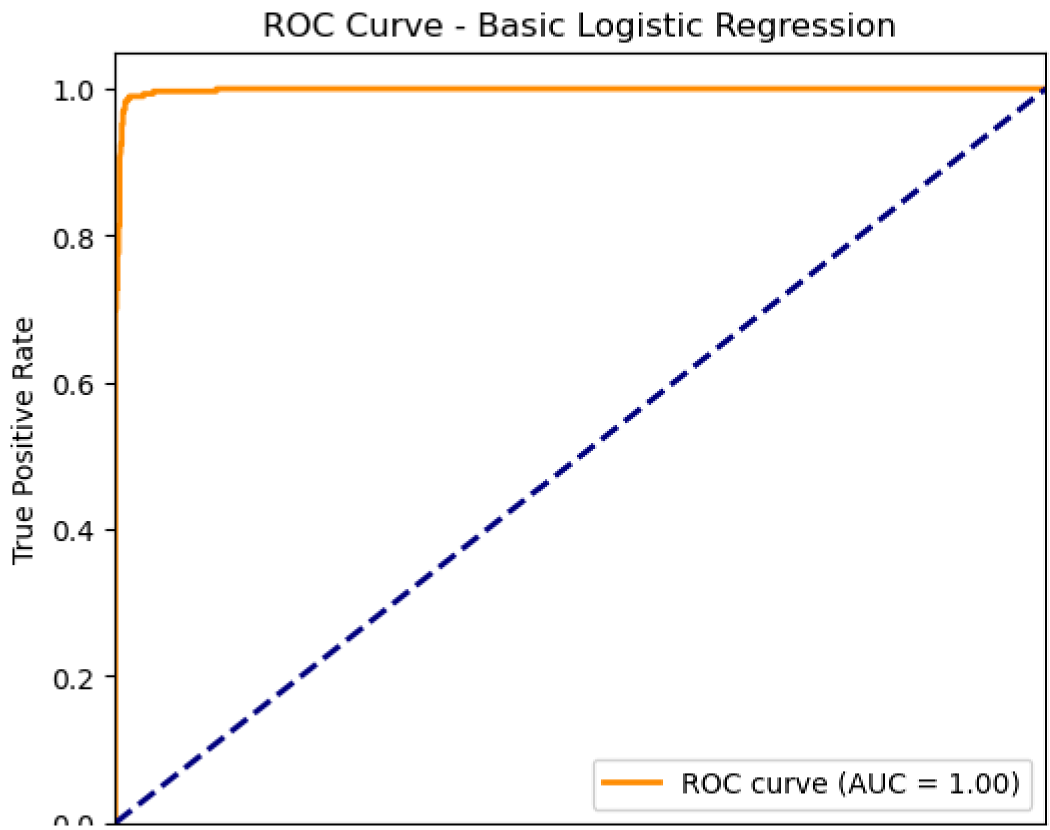
```
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})'
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - {model_name}')
    plt.legend(loc="lower right")
    plt.show()

    return model.coef_
```

2.1.10 Comparison Between Basic, L1 & L2 Regularization

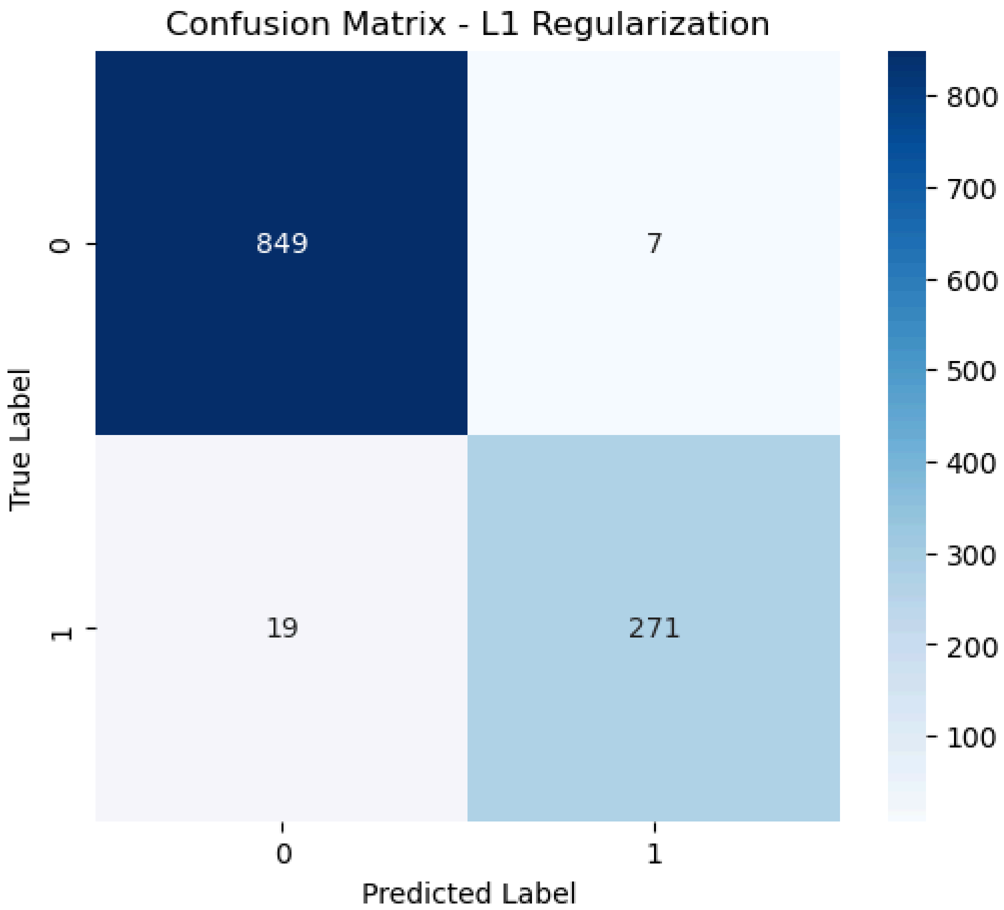
```
# 3. Basic Logistic Regression (Baseline)
print("\n3. Basic Logistic Regression (No Regularization)")
print("-" * 50)

baseline_model = LogisticRegression(random_state=42, max_iter=1000)
baseline_model.fit(X_train, y_train)
baseline_coef = evaluate_model(baseline_model, X_test, y_test, "Basic Logistic Regression

# 4. L1 Regularization (Lasso)
print("\n4. Logistic Regression with L1 Regularization")
print("-" * 50)

l1_model = LogisticRegression(penalty='l1', solver='liblinear', C=1.0, random_state=42)
l1_model.fit(X_train, y_train)
l1_coef = evaluate_model(l1_model, X_test, y_test, "L1 Regularization")

# 5. L2 Regularization (Ridge)
print("\n5. Logistic Regression with L2 Regularization")
print("-" * 50)

l2_model = LogisticRegression(penalty='l2', C=1.0, random_state=42, max_iter=1000)
l2_model.fit(X_train, y_train)
l2_coef = evaluate_model(l2_model, X_test, y_test, "L2 Regularization")
```

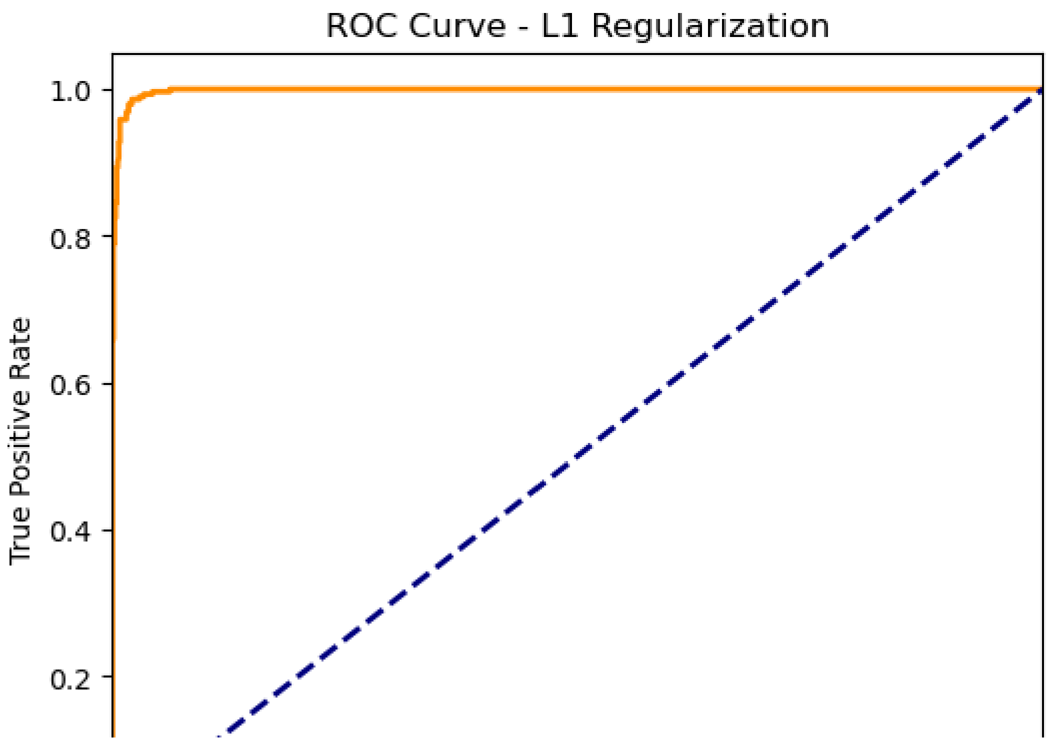
```
3. Basic Logistic Regression (No Regularization)
--------------------------------------------------

Metrics for Basic Logistic Regression:
Accuracy: 0.9799
Precision: 0.9785
Recall: 0.9414
F1 Score: 0.9596
```

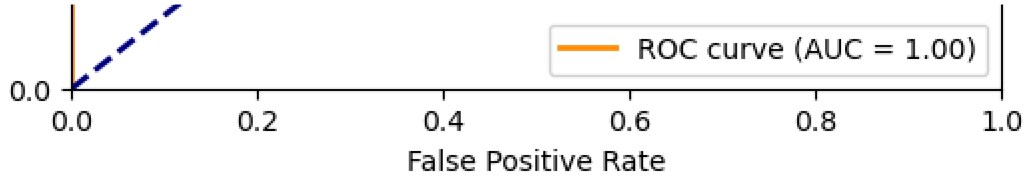




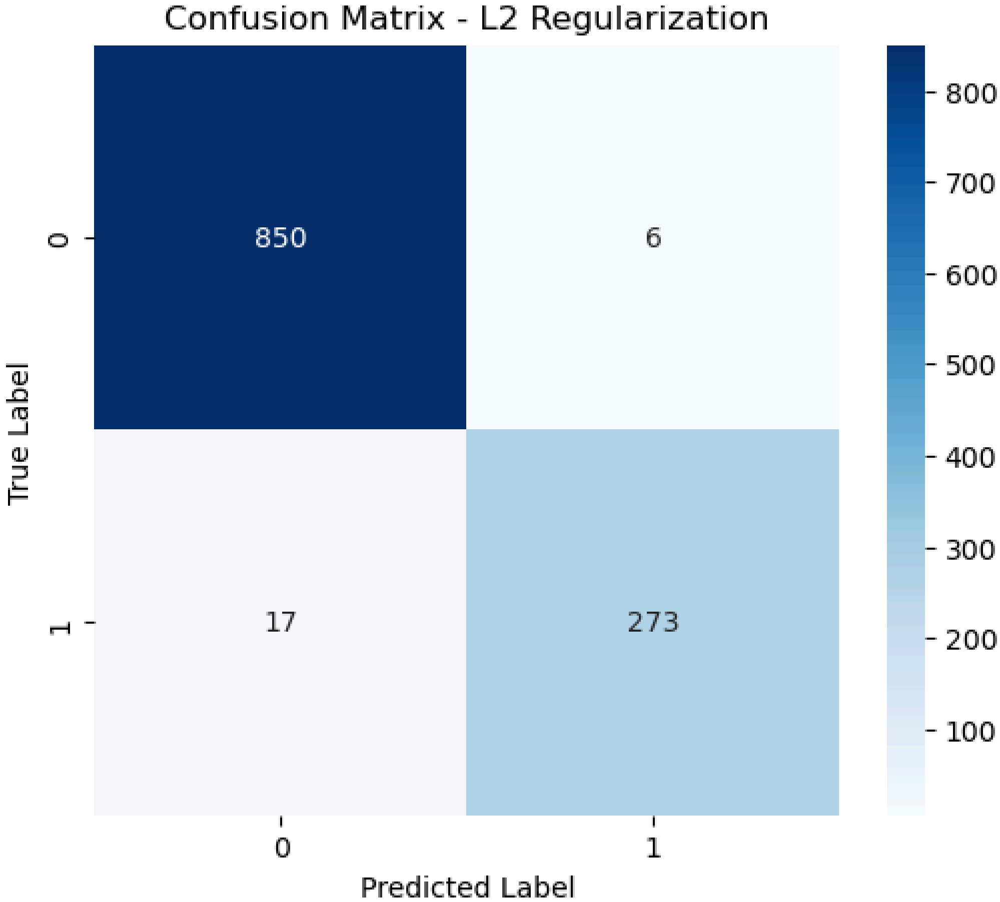

```
4. Logistic Regression with L1 Regularization
--------------------------------------------------

Metrics for L1 Regularization:
Accuracy: 0.9773
Precision: 0.9748
Recall: 0.9345
F1 Score: 0.9542
```

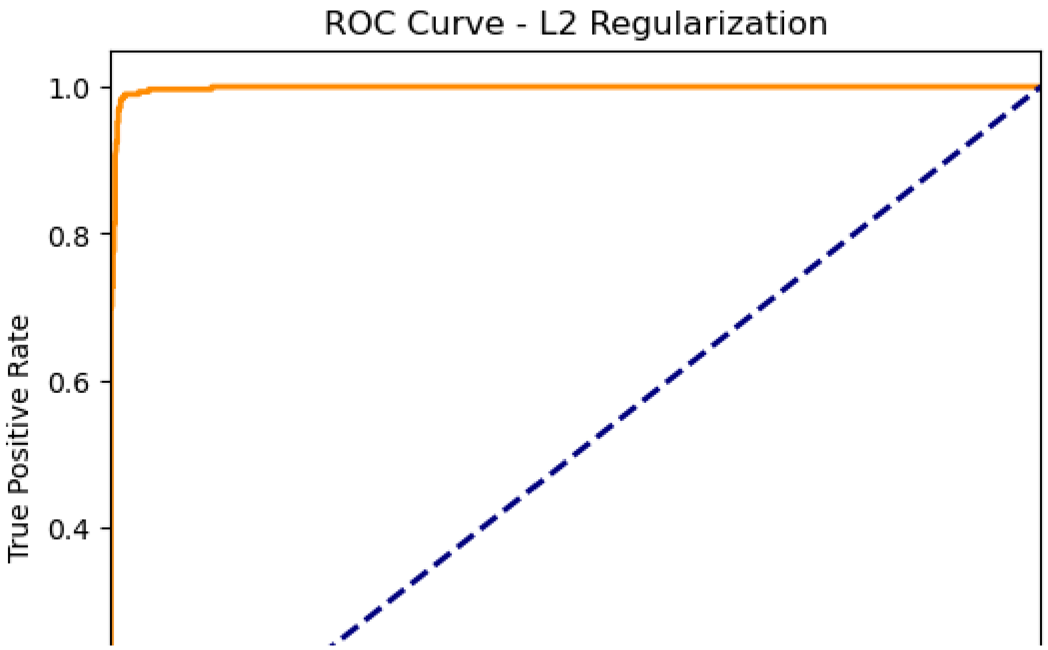







```
5. Logistic Regression with L2 Regularization
--------------------------------------------------

Metrics for L2 Regularization:
Accuracy: 0.9799
Precision: 0.9785
Recall: 0.9414
F1 Score: 0.9596
```





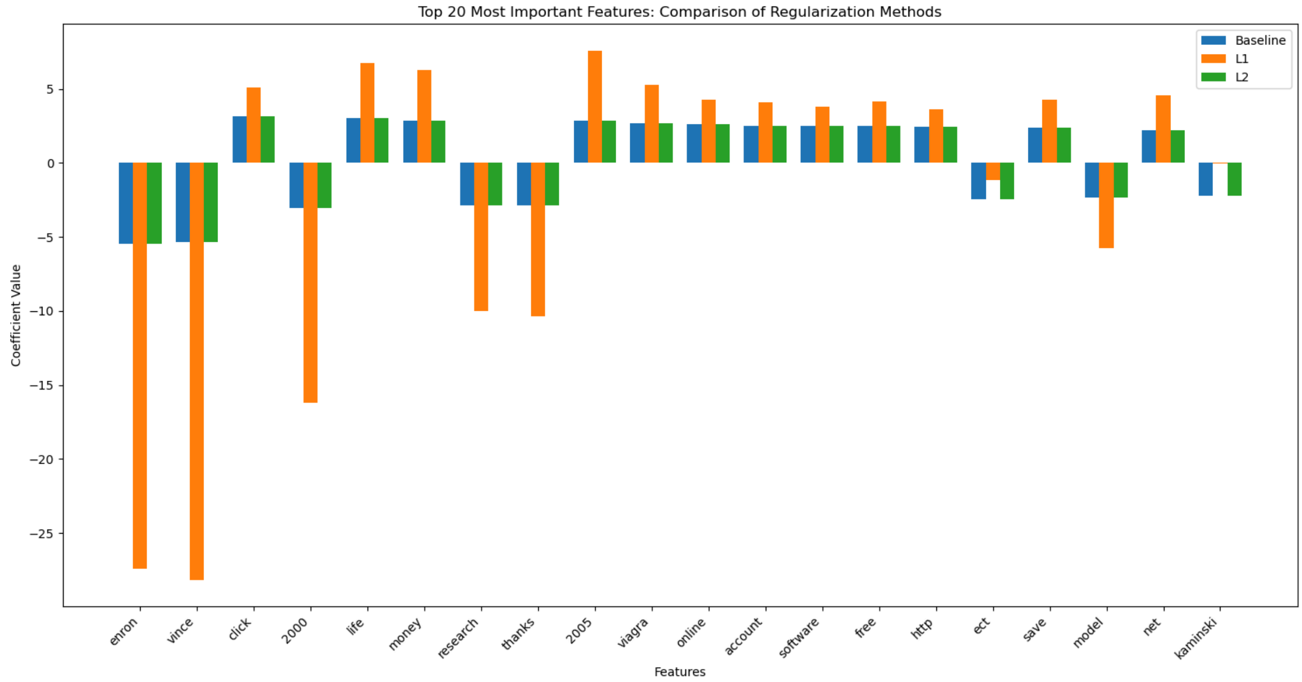

0.2
0.0
ROC curve (AUC = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

```
#6. Comparing Feature Coefficients
print("\n6. Comparing Feature Coefficients")
print("-" * 50)

# Get feature names
feature_names = tfidf.get_feature_names_out()

# Create a DataFrame with coefficients
coef_df = pd.DataFrame({
    'Feature': feature_names,
    'Baseline': baseline_coef[0],
    'L1': l1_coef[0],
    'L2': l2_coef[0]
})

# Sort by absolute value of baseline coefficients
coef_df['Abs_Baseline'] = abs(coef_df['Baseline'])
coef_df = coef_df.sort_values('Abs_Baseline', ascending=False)
```

```
6. Comparing Feature Coefficients
--------------------------------------------------
```

2.1.9 Top 20 most important features

```
# Plot top 20 most important features
plt.figure(figsize=(15, 8))
top_features = coef_df.head(20)
x = np.arange(len(top_features))
width = 0.25

plt.bar(x - width, top_features['Baseline'], width, label='Baseline')
plt.bar(x, top_features['L1'], width, label='L1')
plt.bar(x + width, top_features['L2'], width, label='L2')

plt.xlabel('Features')
plt.ylabel('Coefficient Value')
plt.title('Top 20 Most Important Features: Comparison of Regularization Methods')
plt.xticks(x, top_features['Feature'], rotation=45, ha='right')
plt.legend()
plt.tight_layout()
plt.show()
```



2.1.10 Non-zero Coefficients

```
# Print number of non-zero coefficients
print("\nNumber of non-zero coefficients:")
print(f"Baseline: {np.sum(baseline_coef[0] != 0)}")
print(f"L1 Regularization: {np.sum(l1_coef[0] != 0)}")
print(f"L2 Regularization: {np.sum(l2_coef[0] != 0)}")
```

```
Number of non-zero coefficients:
Baseline: 1000
L1 Regularization: 133
```

```
L2 Regularization: 1000
```

2.1.11 Optimal Regularization Strength

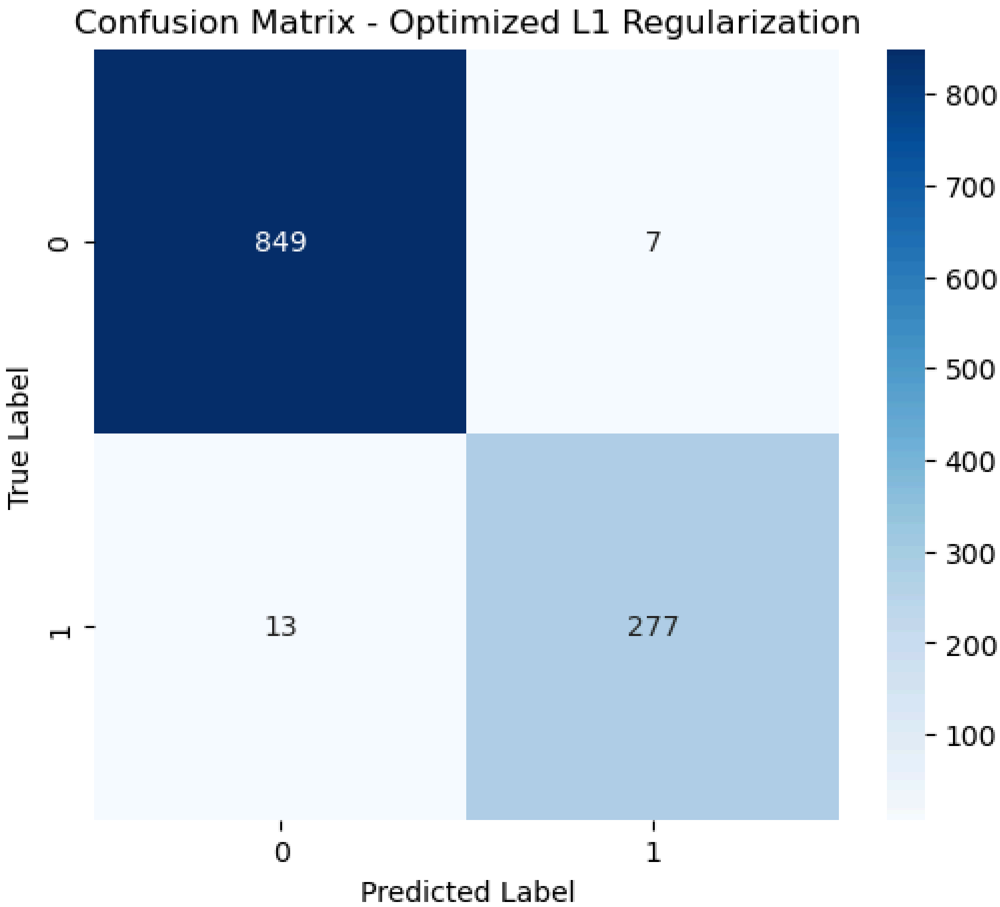
```
# 7. Cross-validation to find optimal regularization strength
print("\n7. Finding Optimal Regularization Strength")
print("-" * 50)

from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

# L1 Regularization
l1_grid = GridSearchCV(
    LogisticRegression(penalty='l1', solver='liblinear', random_state=42),
    param_grid,
    cv=5,
    scoring='f1'
)
l1_grid.fit(X_train, y_train)

# L2 Regularization
l2_grid = GridSearchCV(
    LogisticRegression(penalty='l2', random_state=42, max_iter=1000),
    param_grid,
    cv=5,
    scoring='f1'
)
l2_grid.fit(X_train, y_train)

print("\nBest parameters:")
print(f"L1 Regularization: C={l1_grid.best_params_['C']}")
print(f"L2 Regularization: C={l2_grid.best_params_['C']}")

print("\nBest cross-validation scores:")
print(f"L1 Regularization: {l1_grid.best_score_:.4f}")
print(f"L2 Regularization: {l2_grid.best_score_:.4f}")
```

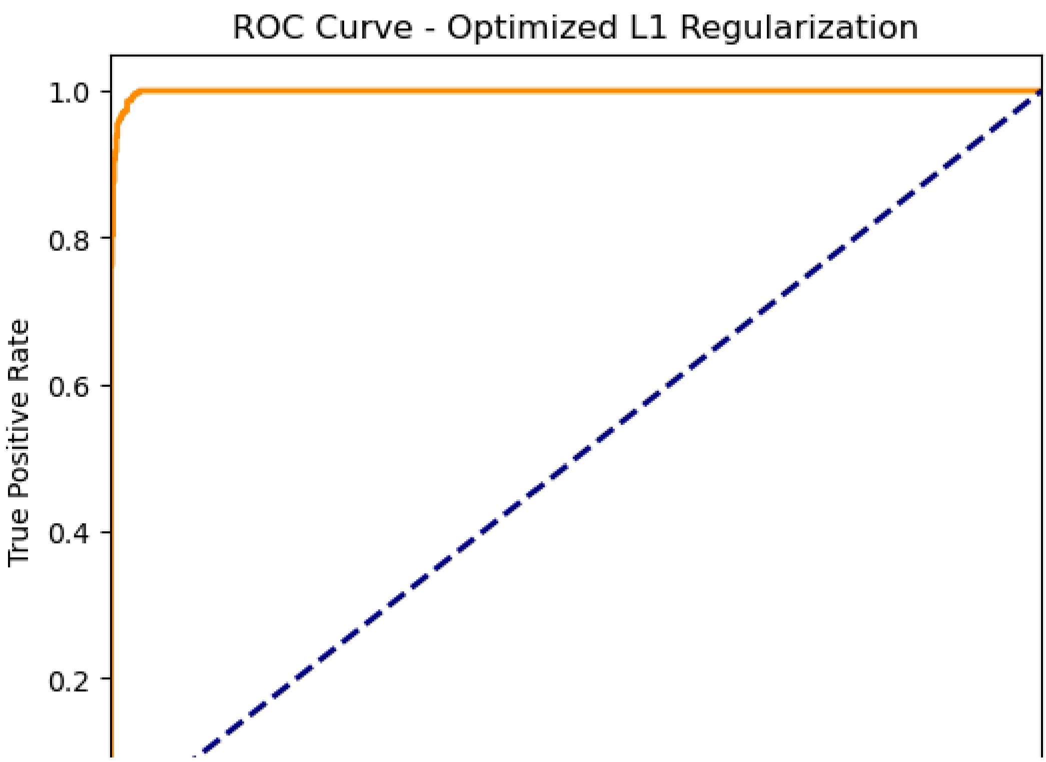
```
7. Finding Optimal Regularization Strength
--------------------------------------------------

Best parameters:
L1 Regularization: C=10
L2 Regularization: C=10

Best cross-validation scores:
L1 Regularization: 0.9552
L2 Regularization: 0.9670
```

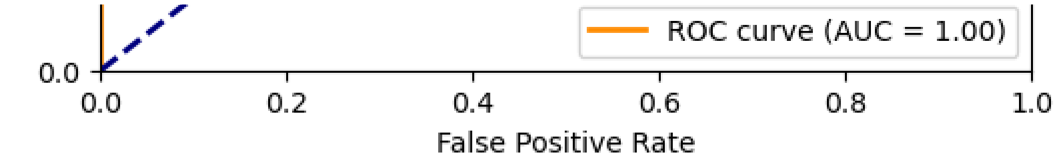
2.1.12 Final Model Comparison

```
# 8. Final model comparison with optimal parameters
print("\n8. Final Model Comparison with Optimal Parameters")
print("-" * 50)

# Train final models with best parameters
final_l1_model = LogisticRegression(penalty='l1', solver='liblinear', C=l1_grid.best_para
final_l2_model = LogisticRegression(penalty='l2', C=l2_grid.best_params_['C'], random_sta

final_l1_model.fit(X_train, y_train)
final_l2_model.fit(X_train, y_train)

print("\nFinal L1 Model Performance:")
evaluate_model(final_l1_model, X_test, y_test, "Optimized L1 Regularization")

print("\nFinal L2 Model Performance:")
evaluate_model(final_l2_model, X_test, y_test, "Optimized L2 Regularization")
```

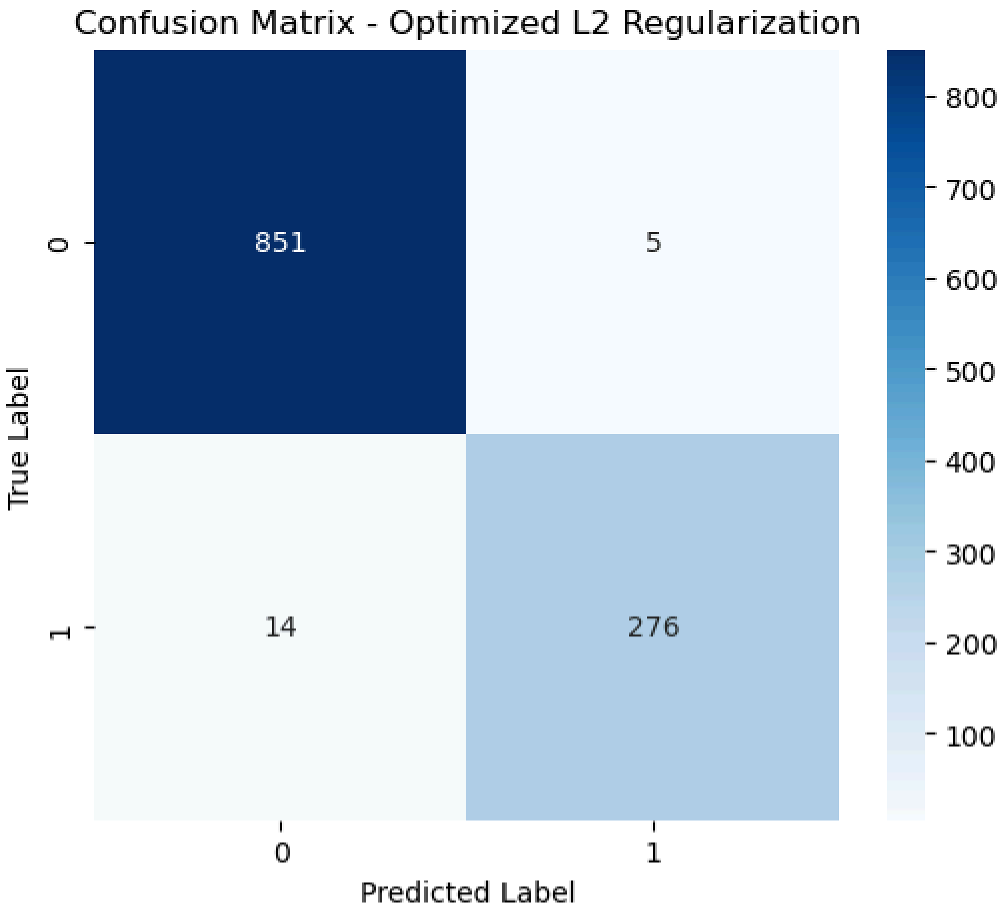
```
8. Final Model Comparison with Optimal Parameters
--------------------------------------------------

Final L1 Model Performance:

Metrics for Optimized L1 Regularization:
Accuracy: 0.9825
Precision: 0.9754
Recall: 0.9552
F1 Score: 0.9652
```

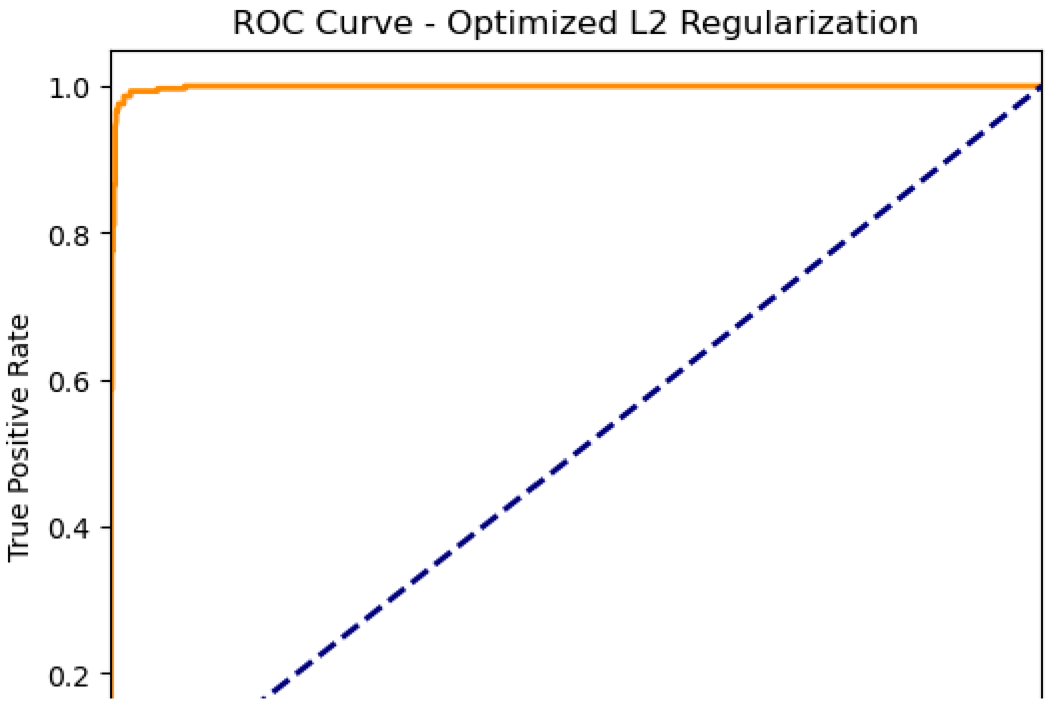


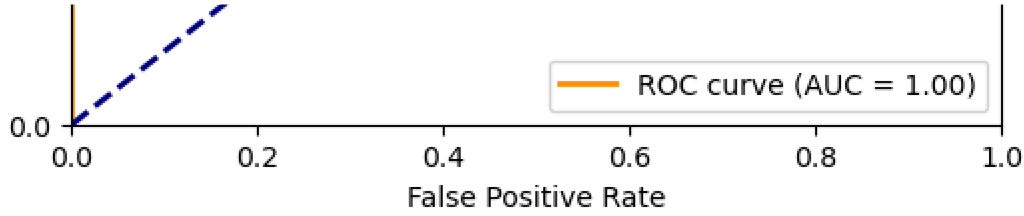





```
Final L2 Model Performance:

Metrics for Optimized L2 Regularization:
Accuracy: 0.9834
Precision: 0.9822
Recall: 0.9517
F1 Score: 0.9667
```








```
array([[-4.33711257e-01,  1.10911303e+00, -1.02553210e+00,
         7.02076560e-01, -1.08319863e+00, -9.52637833e-01,
        -6.13046200e-01, -9.37132962e-01,  3.24197016e-01,
         4.41916051e-01, -2.84572322e-01,  7.03882065e-01,
         2.69977911e+00, -1.46479915e+00, -4.89109655e-01,
        -8.00116689e-01, -1.79352004e-01,  8.27523312e-01,
        -7.77938980e-01, -6.84555125e-02, -8.81614227e-01,
        -2.50728372e-01, -1.62192831e+00,  4.82543429e-01,
        -6.18158673e+00, -3.94821455e+00,  1.84531575e+00,
         5.86263876e+00,  1.17054495e+00, -1.16230117e+00,
         5.66424333e-01, -1.19741270e+00,  2.76631521e+00,
        -5.63959100e-01, -3.48711233e-01,  1.98616413e-01,
         6.75867400e-02, -9.26914698e-01, -1.25246423e+00,
        -1.89922096e+00, -4.29408790e-01, -4.80664677e-01,
        -2.16377680e-02,  4.67403831e-02, -5.36871779e-01,
         5.49421734e-01,  4.63784987e-01,  2.31180109e-01,
        -9.05842034e-02, -8.31220096e-01, -2.97299707e-01,
        -1.73737158e-01,  4.31342050e-01, -2.00104872e+00,
         9.31767719e-02, -3.98961488e-01, -2.77512358e-01,
         8.27232940e-01,  5.91368028e-01,  1.77028252e+00,
         1.39338865e+00,  8.97815609e-01,  4.35488148e-02,
         2.97720419e-02, -1.00687641e+00, -2.71695381e-01,
         5.30932735e-01,  1.22426186e-01,  5.44177303e-01,
        -1.25954362e-01,  2.89944663e-01,  8.04118268e-01,
        -5.93018546e-01,  2.29292347e+00, -4.89094874e-01,
        -4.65577839e+00,  1.06769180e+00, -5.61772480e-01,
        -1.44989590e+00,  2.34797305e+00,  2.72281766e+00,
         1.20025249e+00, -1.08290145e-01, -6.29880739e-01,
        -4.13510798e+00,  5.28806030e+00, -9.85921523e-01,
         1.63876854e+00, -1.16396423e+00, -1.40471329e-01,
        -8.56307212e-02, -1.12799574e+00, -9.19329450e-01,
        -1.08049056e-01, -8.77700871e-01,  2.04622714e+00,
         3.09269950e+00,  9.80318480e-01, -9.99107558e-01,
        -9.93885628e-01, -1.39411858e-01, -9.38298516e-01,
        -6.69986923e-01,  9.98150987e-01, -1.57716973e+00,
        -8.90913971e-01, -4.61716968e-01, -8.59497378e-01,
        -4.96115769e-01, -2.41840643e+00, -8.78068635e-01,
        -1.59968188e+00, -1.51668455e+00, -1.01126292e+00,
        -1.36875310e+00, -2.26290475e+00, -5.77960422e-02,
        -5.44127636e-01, -1.30394107e+00, -7.93769213e-01,
        -3.76639437e-01, -1.60830468e+00,  2.85017597e+00,
        -1.82687631e+00,  4.16841441e+00, -1.35698120e+00,
        -1.10520938e+00, -2.57135171e+00, -4.88714099e-01,
        -1.10059129e+00, -5.18280889e-01, -1.07224105e+00,
         6.48207726e-01,  6.82233566e-01, -3.71078930e-01,
        -1.28640004e+00, -4.76010053e-01, -3.32662340e+00,
        -5.27851394e-01, -8.87901755e-01, -6.86450169e-01,
         1.46756817e+00, -1.29861840e+00,  2.39739658e-01,
         1.65148982e+00, -1.27727446e+00,  2.51691067e+00,
        -4.49848249e-01, -9.39850354e-01, -6.13379376e-01,
        -3.20121477e-01,  9.49176600e-01, -2.96117834e-01,
        -4.55369797e-01,  1.75235070e-01,  1.34288375e+00,
        -3.57000880e-01,  5.56932677e-01, -8.83137093e-01,
        -1.23742812e-01, -1.36872179e+00, -1.54125709e+00,
```

```
-6.29811112e-01,  5.93333703e-01, -4.39140905e-01,
 1.90401718e-01,  1.52287681e+00, -2.41848183e-01,
 9.60398877e-01,  2.22615465e+00, -2.61569895e+00,
-7.22703247e-01, -6.06206383e-01, -6.02055382e-01,
-1.16343153e+00, -1.18549648e+00, -3.30048710e-02,
 3.93636566e-01, -1.39005585e+00,  7.39348736e-01,
-1.92951361e+00, -7.61780124e-01,  2.17529566e-01,
-2.94475091e+00,  1.61466400e+00, -7.23145816e-01,
-1.01680927e+00,  2.15739587e+00, -5.34654316e-01,
-1.56129837e+00, -6.03953450e-01, -6.80697910e-01,
 1.20382488e+00,  2.06895449e-01, -9.71080461e-01,
-6.57832992e-01,  1.58426120e+00, -2.11936322e-01,
 5.60587878e-01,  4.46707837e+00, -2.77363625e-02,
-5.19708410e-01, -7.00492876e-01, -1.12907530e+00,
 1.33183271e+00,  1.33951486e+00,  3.80631934e-01,
-2.86204354e+00, -3.56543505e-01,  1.31679548e-01,
-1.23353827e+00,  1.01352799e+00, -4.38891660e-01,
 7.57416379e-01,  2.77311051e+00,  5.49725060e-01,
-1.12141095e+00,  1.05380990e+00, -3.14877312e+00,
-1.85434589e-01, -4.94414694e-01, -2.62430850e+00,
-8.30722400e-01,  6.79926916e-01, -5.65018025e-01,
-8.54496723e-01, -3.02899803e-02,  1.60189812e+00,
 2.79826635e-01, -2.74357501e-01, -1.40276575e+00,
-1.85660059e+00, -1.13823763e+00, -1.93652568e+00,
 1.30914582e+00, -7.42805771e-01, -2.68841433e-01,
 8.18069371e-01, -6.93628808e-01, -1.04072594e+00,
-5.71597645e-01,  1.52873577e+00,  1.37436542e+00,
-9.98657954e-01,  3.31304557e-01, -9.55865298e-01,
-1.55133915e+00, -1.23259395e+00, -6.37268215e-01,
 1.67104229e+00, -4.04843742e-01, -4.45891149e-01,
-1.03351248e+00, -2.16744964e+00, -6.63361530e-02,
-1.80580646e+00,  2.35146157e-01, -7.66539922e-01,
 1.10048312e-01,  5.27305894e-02, -5.86329966e-01,
-9.47193278e-01, -1.20936179e+00, -6.12850340e-01,
-9.33572764e-01,  3.12450470e+00, -1.06646598e+00,
-1.77395278e+00, -8.93421197e-02, -1.80069781e+00,
 1.60417031e+00, -2.32695187e+00,  2.20574477e-01,
-9.17270845e-01,  1.95786039e-01, -6.61392736e-01,
-1.01314794e+00, -1.24626807e+00, -1.04383296e+00,
-1.59063430e+00, -4.28362652e-01,  1.76096536e-01,
-1.37208058e+00, -1.51595114e+00, -1.37175711e+00,
-5.73115467e-01, -6.54059409e-01,  5.91453687e-01,
-3.10851510e+00, -2.55358800e+00,  7.87148940e-01,
 1.13873660e+00,  1.90340249e+00,  2.99936732e+00,
-7.10183736e-02, -4.59391911e-01,  4.86465583e-01,
-1.44166309e+00, -5.41242893e-01, -1.47449398e+00,
-5.56402308e-01,  1.82967288e+00, -1.70593023e+00,
-1.69400072e+00, -9.54966578e-01, -8.73613189e-01,
-3.49679157e+00, -2.18808147e+00, -1.62040341e+00,
 1.36885101e+00, -1.62406209e-01,  3.85689214e-01,
-7.92707747e-01, -2.07883731e+00, -2.11134388e+00,
 1.55416769e+00, -1.78303098e+00, -1.35107999e+00,
-1.65142526e+00, -9.54654737e-01, -4.78517256e+00,
 3.97224348e-01, -1.05227207e+01, -1.91773007e-01,
 1.02080619e+00,  4.60432979e-01, -3.17449176e-01,
-4.54906498e-01,  7.69923999e-01,  1.02726990e+00,
-4.81173920e-01, -7.70142900e-01, -1.59691702e+00,
-1.89889652e-01,  5.80457460e-02, -1.30402762e+00,
-1.02984579e+00, -1.05869525e+00,  5.25262273e-02,
 4.01864147e-03,  1.66621699e-01,  3.75277675e-01,
```

```
        1.08533155e-01,  1.97675941e-01, -5.17767978e-01,
       -3.10709299e-01,  2.19853675e+00, -1.05171810e+00,
        2.51432524e+00, -1.56085092e+00, -1.04732858e+00,
       -9.59764310e-01, -9.97994081e-01, -9.71843243e-01,
       -1.40697877e+00, -6.83023342e-01, -2.47660873e+00,
        3.67252579e-01, -2.08779034e+00, -4.51080177e-01,
       -1.03186458e+00, -1.62879694e+00,  5.36274074e-02,
        5.06428541e-02, -9.90604160e-01, -5.64418670e-01,
        4.51321523e-01, -1.20495803e+00, -5.71583461e-02,
       -4.18443552e-01, -2.41559042e+00, -1.20676423e+00,
        4.50679201e+00, -1.74979348e+00,  4.88402640e-01,
        2.57124447e-01,  2.83642200e+00,  2.12430311e+00,
       -2.02807377e+00, -8.52292723e-01,  1.04690622e+00,
       -4.94938164e-01, -5.02805912e-02, -4.04746499e-01,
        2.13157610e-01, -8.33314364e-01, -7.08072382e-01,
        7.03112435e-01, -1.61811565e-01, -2.16595347e+00,
       -1.36826960e-01,  2.28043310e+00,  3.96890785e-02,
        9.12904888e-01, -1.41064451e+00, -1.11026972e+00,
        9.14371287e-01, -7.45522568e-01, -2.18201764e+00,
       -3.45734338e-01,  1.35656316e+00, -1.11100438e+00,
       -6.94339019e-01, -5.92544830e-01, -9.94336411e-02,
        1.21081989e+00,  6.32797943e-01,  2.21940642e-01,
       -2.95201527e-01, -3.78625775e-01,  1.09242154e+00,
        1.64656746e+00, -1.07337195e+00, -1.73934170e+00,
        2.05683589e+00, -8.08723563e-01, -1.35710774e+00,
        1.80049680e+00, -2.73018762e+00, -1.05277020e-01,
       -1.55678799e+00, -1.16345140e+00,  1.46782344e+00,
        1.13910193e+00, -3.14531609e+00, -1.17621530e+00,
       -1.73072090e+00,  3.70924543e+00, -2.50843813e+00,
        4.31618014e-01, -7.74569128e-01,  1.16744968e+00,
        3.33892019e-01,  1.61685643e-01, -3.30987110e-01,
        5.43095811e-01, -8.58216124e-02,  2.23457741e-01,
        2.08215336e+00, -4.43616744e-01, -2.15841599e-01,
       -2.79745240e-01, -6.39894227e-01,  2.43635743e-02,
        1.16810672e+00, -2.33930574e-01, -4.81044814e-02,
        1.94600899e+00, -5.87906089e-01,  1.59578507e+00,
       -8.70881814e-02, -4.69067420e-01, -1.98327351e+00,
        1.13908075e-02,  2.31431913e+00, -8.26626497e-01,
        3.32121926e-01,  6.37791827e-02, -1.24478412e+00,
        1.46182384e+00, -2.16054478e+00, -1.43848303e+00,
       -2.91158663e-01, -4.45448746e-01, -8.45346252e-01,
       -2.17711959e+00, -1.81028784e+00, -5.82851088e-01,
       -1.96686429e-01, -1.54078823e+00, -1.04540807e+00,
       -1.84834189e+00, -5.15398690e-01, -1.19314061e+00,
        2.11995569e+00, -4.50060558e-02, -1.26628989e-01,
       -6.79665244e-01,  2.18772024e+00, -4.40559294e+00,
       -1.05852497e+00, -2.47583885e+00,  5.57513140e-01,
       -5.69317755e-01,  6.54827131e-01, -1.69256263e+00,
        1.12161991e+00, -2.10308547e+00, -1.86528637e-01,
       -2.45857255e-01,  2.78696596e-01,  7.47173728e-01,
        1.27005147e+00, -1.13954094e+00,  9.92157639e-01,
        1.86847425e+00,  2.65034506e+00, -1.87529585e+00,
        1.22595442e-01,  1.54709497e+00, -3.65774826e-01,
       -3.32617959e+00,  1.38586401e-01, -1.32783283e+00,
       -2.59509981e+00,  5.16684192e+00,  1.55043375e+00,
       -2.27753215e+00,  2.02662020e+00,  5.96708780e-01,
        1.42722597e+00,  5.14579144e-01,  1.05436914e+00,
        1.22557539e-01,  4.31088855e-01,  8.12268904e-01,
       -2.42799666e-01, -9.26367352e-01, -1.55610810e+00,
        2.24139188e+00, -6.74595741e-02, -2.35458210e+00,
        1.71119840e+00,  4.04793009e-01, -3.63623410e-01,
```

```
 1.81115698e+00, -4.36861560e-01,  6.12294005e-02,
 2.94803725e+00, -8.58216240e-02, -1.40694263e+00,
-6.69376951e-01, -2.66781949e-01,  3.48114307e+00,
 5.21356612e-02, -1.29916810e+00,  1.75187051e-01,
 1.52273134e+00,  1.27905756e+00,  5.63926659e-01,
 4.12027374e-01, -1.61295151e+00,  5.71184152e-01,
-1.44401776e+00, -9.39124962e-01, -8.54665959e-01,
-1.82824530e+00, -1.10891619e-02,  3.25172571e+00,
-1.97044758e+00, -1.30880315e+00, -5.99789470e-01,
 3.68288473e-01,  9.09486980e-01, -1.41886312e+00,
-3.97096116e-01, -7.25792946e-02,  3.39702353e-01,
 1.68079608e+00, -3.17486532e-01, -3.06470439e+00,
-8.73896151e-01,  5.76969721e-01, -2.94747578e-01,
-1.74849149e+00,  2.53680520e+00, -4.89590463e-01,
-3.49743811e-01, -2.14572555e+00,  2.61623894e+00,
 2.45085082e+00,  1.55649596e+00, -4.48157338e+00,
-1.53613019e+00, -6.98380209e-01, -1.05339352e+00,
-2.08356161e+00,  4.33122820e+00,  4.38712599e-01,
-2.49734272e-01, -6.64802740e-01, -1.91328062e+00,
-2.36514332e+00, -7.07883351e-01,  3.88996398e-01,
-5.24201198e-01, -4.85835911e-01, -2.36410801e+00,
 7.16980307e-01,  6.92257025e-01,  1.05175815e-01,
-9.09080304e-01, -6.07910697e-01, -1.78025596e+00,
-3.57475847e-01,  1.51716351e+00,  3.76742897e+00,
-3.45970155e+00, -6.34998467e-01,  1.05022409e-01,
-4.19810648e-01,  1.44162812e+00,  2.65217322e-03,
-3.77001774e-01,  1.35157162e-01, -7.04073700e-01,
-2.76256793e+00,  2.05706055e+00, -6.71225676e-01,
-1.29480335e+00, -1.81827117e+00, -9.25748615e-01,
-1.26869975e+00, -9.30864200e-01,  2.29863397e+00,
 1.32504357e+00, -1.87374285e+00, -1.72150896e+00,
 1.04903967e+00,  3.71517696e-01,  3.63615242e+00,
-8.39960092e-01,  4.43519247e-01, -1.08524260e+00,
-1.36039035e-01,  1.25028931e-01, -2.62841059e+00,
-2.12441050e+00,  1.12550291e+00,  1.70131960e-01,
-1.66580845e-01,  6.41258116e-01,  6.12251257e-01,
 1.25712036e+00, -7.71746875e-02, -2.90064986e-01,
-1.95489493e+00,  2.29763755e+00, -1.27152896e-01,
-1.83640071e+00,  2.31306703e-01, -1.11853584e+00,
-8.19543231e-01, -7.50578972e-02, -4.73512396e+00,
-3.28482155e-01, -3.37135719e-01, -1.28404185e+00,
-7.56482387e-01,  2.21212193e+00,  1.07495325e+00,
 1.46479102e+00, -3.40869892e+00,  1.64136750e+00,
-1.32744411e+00,  7.71373826e-01, -1.34304236e+00,
-1.37252391e-01,  9.51236534e-01,  5.11793415e-01,
-3.28815401e-01, -7.78288701e-01, -3.05225294e+00,
 1.45894164e+00, -1.41680340e+00,  8.63159543e-02,
 6.18665835e-01, -7.79925153e-01,  5.98418187e-01,
-1.87521424e+00, -3.44928418e-01, -5.43807776e-01,
-1.22596356e+00, -1.24128071e-01, -1.20131252e+00,
-5.80921210e-01, -8.38584383e-02, -2.56046348e-01,
 1.57630213e+00, -2.15564840e+00, -9.43361177e-01,
 9.01411869e-01, -1.69457842e-01, -2.29368950e+00,
-2.50628150e-01,  1.01397641e-01,  3.62063958e-01,
-4.79727904e-01,  1.15449713e+00,  5.21413317e-01,
-7.17919912e-01, -8.56839792e-01,  1.24023157e+00,
 2.77356230e+00, -1.01279993e+00, -1.90554584e-01,
-6.08574312e-01, -1.56622332e+00,  7.28627269e-01,
 5.03061538e-01,  4.16428460e-01,  3.18429042e+00,
-5.23478817e-01,  9.78071734e-01,  7.88780552e-01,
```