```
In [9]:  #
         import pandas as pd
         import numpy as np
```

```
In [10]: # 1.
         names = ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Siz
                  'Uniformity of Cell Shape', 'Marginal Adhesion', 'Single Epithel
                  'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses',

         data_path = "data/breast-cancer-wisconsin.data"
         data = pd.read_csv(data_path, names=names)

         # 2
         # 2.1
         data = data.replace(to_replace="?", value=np.nan)
         data = data.dropna()

         # 2.2
         data.iloc[:, 1:] = data.iloc[:, 1:].apply(pd.to_numeric, errors='coerce')
         data = data.dropna()

         x = data.iloc[:, 1:10].values.astype(np.float64)
         y = data["Class"].values.astype(int)
         y = np.where(y == 4, 1, 0)
```

```
In [11]: # 2.3
         def train_test_split_manual(X, y, test_size=0.25, random_state=2025):
             """                        """
             if random_state is not None:
                 np.random.seed(random_state)

             n_samples = X.shape[0]
             n_test = int(n_samples * test_size)

             #
             indices = np.random.permutation(n_samples)
             test_indices = indices[:n_test]
             train_indices = indices[n_test:]

             return X[train_indices], X[test_indices], y[train_indices], y[test_in
```

```
In [12]: # 2.4
         class StandardScaler:
             # TODO
             #
             #          : z = (x - mean) / std

             def __init__(self):
                 self.mean_ = None
                 self.std_ = None

             def fit(self, X):
                 """             """
```

```
        self.mean_ = np.mean(X, axis=0)
        self.std_ = np.std(X, axis=0)


    def transform(self, X):
        """                         """
        return (X - self.mean_) / self.std_

    def fit_transform(self, X):
        """           """
        self.fit(X)
        return self.transform(X)
```

$h_\theta(x) = \frac{1}{1+e^{-\theta x}}$

- x
- θ
- $h_\theta(x)$

$$\text{class} = \begin{cases} 1, & \text{if } h_\theta(x) \geq 0.5 \\ 0, & \text{if } h_\theta(x) < 0.5 \end{cases}$$

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

$$\theta := \theta - \alpha\frac{\partial J(\theta)}{\partial \theta}$$

α

In [39]:
```
# 3.
class LogisticRegression:
    """
       :
        learning_rate:
        n_iterations:
    """
    def __init__(self, learning_rate=0.01, n_iterations=1000, regularizat
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.weights = None
        self.bias = None
        self.regularization = regularization
        self.lambda_reg = lambda_reg

    def sigmoid(self, z: np.ndarray) -> np.ndarray:
        # TODO:     sigmoid
```

```python
        # sigmoid(z) = 1 / (1 + e^(-z))
        return 1 / (1 + np.exp(-z))

    def fit(self, X: np.ndarray, y: np.ndarray) -> None:
        # TODO:
        #                   BatchGradientDescent  BGD

        #     :
        # 1.
        # 2.     n_iterations
        # 3.         :
        #     step1.
        #     step2.
        #     step3.
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iterations):
            linear_pred = np.dot(X, self.weights) + self.bias
            predictions = self.sigmoid(linear_pred)

            d_weights = np.dot(X.T, (predictions - y)) / n_samples
            d_bias = np.sum(predictions - y) / n_samples

            if self.regularization == 'L1':
                d_weights += self.lambda_reg * np.sign(self.weights)
            elif self.regularization == 'L2':
                d_weights += 2 * self.lambda_reg * self.weights

            self.weights -= self.learning_rate * d_weights
            self.bias -= self.learning_rate * d_bias

    def predict(self, X: np.ndarray, threshold: float = 0.5) -> np.ndarra
        # TODO:
        #     :
        # 1.
        # 2.     sigmoid
        # 3.
        linear_pred = np.dot(X, self.weights) + self.bias
        predictions = self.sigmoid(linear_pred)
        return np.where(predictions >= threshold, 1, 0)

    def predict_proba(self, X: np.ndarray) -> np.ndarray:
        linear_pred = np.dot(X, self.weights) + self.bias
        return self.sigmoid(linear_pred)
```

In [14]:
```python
def get_metrics(y_true, y_pred):
    """

    """
    def recall_score(y_true, y_pred):
        """

              = TP / (TP + FN)
        """
        TP = np.sum((y_true == 1) & (y_pred == 1))
        FN = np.sum((y_true == 1) & (y_pred == 0))
        return TP / (TP + FN) if TP + FN > 0 else 0
    def precision_score(y_true, y_pred):
        """      = TP / (TP + FP)"""
        TP = np.sum((y_true == 1) & (y_pred == 1))
        FP = np.sum((y_true == 0) & (y_pred == 1))
        return TP / (TP + FP) if TP + FP > 0 else 0

    def accuracy_score(y_true, y_pred):
        """      = (TP + TN) /    """
        return np.mean(y_true == y_pred)


    def confusion_matrix(y_true, y_pred):
        """       """
        TN = np.sum((y_true == 0) & (y_pred == 0))
        FP = np.sum((y_true == 0) & (y_pred == 1))
        FN = np.sum((y_true == 1) & (y_pred == 0))
        TP = np.sum((y_true == 1) & (y_pred == 1))

        return np.array([[TN, FP], [FN, TP]])

    recall = recall_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    accuracy = accuracy_score(y_true, y_pred)
    cm = confusion_matrix(y_true, y_pred)
    return recall, precision, accuracy, cm
```

L1   L2

In [31]:
```
def compare_regularization_effects():
    """                 """
    print("=" * 80)
    print("               ")
    print("=" * 80)

    #
    X_train, X_test, y_train, y_test = train_test_split_manual(x, y, test

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    #
    models = {
        '       ': LogisticRegression(learning_rate=0.01, n_iterations=10
        'L1     ': LogisticRegression(learning_rate=0.01, n_iterations=10
                                                  regularization='l1', lamb
        'L2     ': LogisticRegression(learning_rate=0.01, n_iterations=10
                                                  regularization='l2', lamb
    }

    results = {}

    for name, model in models.items():
        print(f"\n    {name}    ...")
        model.fit(X_train, y_train)

        #
        y_pred = model.predict(X_test)
        recall, precision, accuracy, cm = get_metrics(y_test, y_pred)

        results[name] = {
            'recall': recall,
            'precision': precision,
            'accuracy': accuracy,
            'weights_norm': np.linalg.norm(model.weights),
            'weights_sparsity': np.sum(np.abs(model.weights) < 1e-6) / le
        }

        print(f"{name}     :")
        print(f"       : {recall:.4f}")
        print(f"       : {precision:.4f}")
        print(f"       : {accuracy:.4f}")
        print(f"         : {np.linalg.norm(model.weights):.4f}")
        print(f"           : {np.sum(np.abs(model.weights) < 1e-6) / len(r

    return results
```

In [36]:
```
def analyze_threshold_effects():
    """                           """
    print("\n" + "=" * 80)
    print("             ")
    print("=" * 80)

    #
    X_train, X_test, y_train, y_test = train_test_split_manual(x, y, test

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    model = LogisticRegression(learning_rate=0.01, n_iterations=1000)
    model.fit(X_train, y_train)

    #
    y_proba = model.predict_proba(X_test)

    #
    thresholds = np.arange(0.1, 1.0, 0.1)
    threshold_results = []

    print(f"{'   ':<8} {'     ':<10} {'     ':<10} {'     ':<10} {'F1
    print("-" * 50)

    for threshold in thresholds:
        y_pred = (y_proba >= threshold).astype(int)
        recall, precision, accuracy, cm = get_metrics(y_test, y_pred)
        f1_score = 2 * (precision * recall) / (precision + recall) if (pr

        threshold_results.append({
            'threshold': threshold,
            'recall': recall,
            'precision': precision,
            'accuracy': accuracy,
            'f1_score': f1_score
        })
```

```python
        print(f"{threshold:<8.1f} {recall:<10.4f} {precision:<10.4f} {acc

    #
    best_f1_idx = np.argmax([r['f1_score'] for r in threshold_results])
    best_threshold = threshold_results[best_f1_idx]

    print(f"\n       : {best_threshold['threshold']:.1f}")
    print(f"   F1   : {best_threshold['f1_score']:.4f}")

    return threshold_results, best_threshold
```

In [41]:
```python
import matplotlib.pyplot as plt

def plot_threshold_analysis(threshold_results):
    """              """
    thresholds = [r['threshold'] for r in threshold_results]
    recalls = [r['recall'] for r in threshold_results]
    precisions = [r['precision'] for r in threshold_results]
    f1_scores = [r['f1_score'] for r in threshold_results]

    plt.figure(figsize=(12, 8))

    plt.subplot(2, 2, 1)
    plt.plot(thresholds, recalls, 'b-o', label='Recall')
    plt.xlabel('Threshold')
    plt.ylabel('Recall')
    plt.title('Recall vs Threshold')
    plt.grid(True)

    plt.subplot(2, 2, 2)
    plt.plot(thresholds, precisions, 'r-o', label='Precision')
    plt.xlabel('Threshold')
    plt.ylabel('Precision')
    plt.title('Precision vs Threshold')
    plt.grid(True)

    plt.subplot(2, 2, 3)
    plt.plot(thresholds, f1_scores, 'g-o', label='F1-Score')
    plt.xlabel('Threshold')
    plt.ylabel('F1-Score')
    plt.title('F1-Score vs Threshold')
    plt.grid(True)

    plt.subplot(2, 2, 4)
    plt.plot(thresholds, recalls, 'b-o', label='Recall')
    plt.plot(thresholds, precisions, 'r-o', label='Precision')
    plt.plot(thresholds, f1_scores, 'g-o', label='F1-Score')
    plt.xlabel('Threshold')
    plt.ylabel('Metric Value')
    plt.title('All Metrics vs Threshold')
    plt.legend()
    plt.grid(True)

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

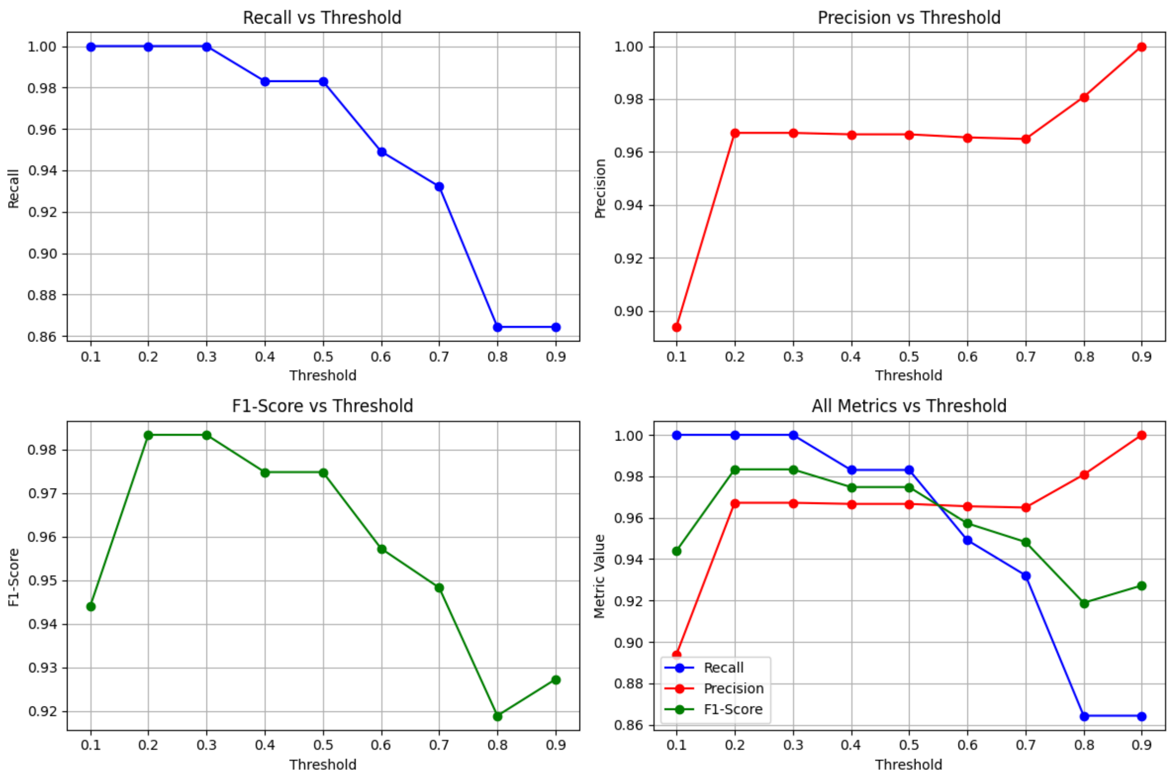
```
def plot_regularization_comparison(results):
    """             """
    models = list(results.keys())
    recalls = [results[m]['recall'] for m in models]
    precisions = [results[m]['precision'] for m in models]
    weights_norms = [results[m]['weights_norm'] for m in models]

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

    #
    x = np.arange(len(models))
    width = 0.35

    ax1.bar(x - width/2, recalls, width, label='Recall', alpha=0.8)
    ax1.bar(x + width/2, precisions, width, label='Precision', alpha=0.8)
    ax1.set_xlabel('Model')
    ax1.set_ylabel('Metric Value')
    ax1.set_title('Regularization Methods Performance Comparison')
    ax1.set_xticks(x)
    ax1.set_xticklabels(models, rotation=45)
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    #
    ax2.bar(models, weights_norms, alpha=0.8, color='orange')
    ax2.set_xlabel('Model')
    ax2.set_ylabel('Weight Norm')
    ax2.set_title('Weight Norm Comparison')
    ax2.tick_params(axis='x', rotation=45)
    ax2.grid(True, alpha=0.3)

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

```
In [8]: if __name__=='__main__':
    print("=" * 60)
    print("             -            ")
    print("=" * 60)

    #
    X_train, X_test, y_train, y_test = train_test_split_manual(x, y, test

    print(f"\n        :")
    print(f"          : {X_train.shape[0]}")
    print(f"          : {X_test.shape[0]}")
    print(f"     : {X_train.shape[1]}")

    #
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    print("\n          ...")
    #
    model = LogisticRegression(learning_rate=0.01, n_iterations=1000)
    model.fit(X_train, y_train)

    #
    y_pred = model.predict(X_test)
```

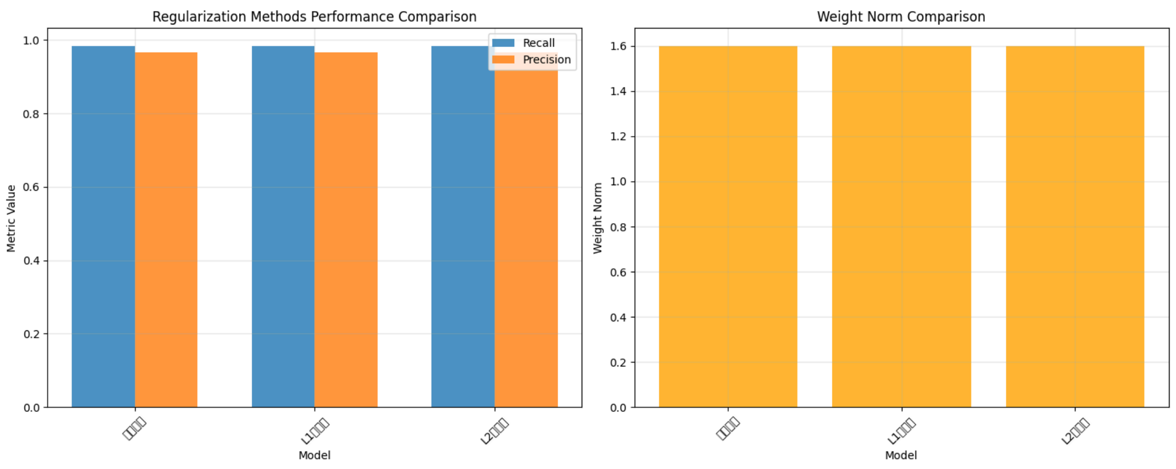
```python
    #
    print("\n" + "=" * 60)
    print("          ")
    print("=" * 60)

    recall, precision, accuracy, cm = get_metrics(y_test, y_pred)

    print(f"\n      (Recall):    {recall:.4f} ({recall*100:.2f}%)")
    print(f"      (Precision): {precision:.4f} ({precision*100:.2f}%)")
    print(f"      (Accuracy):  {accuracy:.4f} ({accuracy*100:.2f}%)")

    print(f"\n       :")
    print(f"                             ")
    print(f"               {cm[0,0]:4d}     {cm[0,1]:4d}")
    print(f"               {cm[1,0]:4d}     {cm[1,1]:4d}")
```

```
============================================================
             –
============================================================

         :
          : 513
          : 170
     : 9

          ...

============================================================

============================================================

      (Recall):    0.9831 (98.31%)
      (Precision): 0.9667 (96.67%)
      (Accuracy):  0.9824 (98.24%)

       :

                109        2
                  1       58
```

```python
In [42]: if __name__ == '__main__':
    #               ...

    #
    print("\n" + "=" * 80)
    print("                      ")
    print("=" * 80)

    #
    reg_results = compare_regularization_effects()

    #
    threshold_results, best_threshold = analyze_threshold_effects()

    #
    plot_threshold_analysis(threshold_results)
    plot_regularization_comparison(reg_results)

    #
    print("\n" + "=" * 80)
    print("       ")
    print("=" * 80)

    print("\n1.          :")
    print("   - L1                               ")
    print("   - L2                        ")
    print("   -                              ")

    print("\n2.        :")
    print(f"   -                       ")
    print(f"   -                       ")
    print(f"   -          {best_threshold['threshold']:.1f}
==========================================================================
======

==========================================================================
======
==========================================================================
======

==========================================================================
======

                ...
            :
       : 0.9831
       : 0.9667
       : 0.9824
         : 1.5988
           : 0.0000

    L1          ...
L1          :
       : 0.9831
       : 0.9667
       : 0.9824
         : 1.5988
           : 0.0000

    L2          ...
L2          :
       : 0.9831
       : 0.9667
       : 0.9824
         : 1.5988
           : 0.0000

==========================================================================
======

==========================================================================
======
                                                  F1
--------------------------------------------------
0.1      1.0000     0.8939     0.9588     0.9440
0.2      1.0000     0.9672     0.9882     0.9833
0.3      1.0000     0.9672     0.9882     0.9833
0.4      0.9831     0.9667     0.9824     0.9748
0.5      0.9831     0.9667     0.9824     0.9748
0.6      0.9492     0.9655     0.9706     0.9573
0.7      0.9322     0.9649     0.9647     0.9483
0.8      0.8644     0.9808     0.9471     0.9189
0.9      0.8644     1.0000     0.9529     0.9273

       : 0.2
   F1    : 0.9833
```



```
/var/folders/m7/nm2vlx1n4mg_rg3xtr8gbr640000gp/T/ipykernel_21364/157641164
1.py:77: UserWarning: Glyph 26080 (\N{CJK UNIFIED IDEOGRAPH-65E0}) missing
from font(s) DejaVu Sans.
  plt.tight_layout()
/var/folders/m7/nm2vlx1n4mg_rg3xtr8gbr640000gp/T/ipykernel_21364/157641164
1.py:77: UserWarning: Glyph 27491 (\N{CJK UNIFIED IDEOGRAPH-6B63}) missing
from font(s) DejaVu Sans.
  plt.tight_layout()
/var/folders/m7/nm2vlx1n4mg_rg3xtr8gbr640000gp/T/ipykernel_21364/157641164
1.py:77: UserWarning: Glyph 21017 (\N{CJK UNIFIED IDEOGRAPH-5219}) missing
from font(s) DejaVu Sans.
  plt.tight_layout()
/var/folders/m7/nm2vlx1n4mg_rg3xtr8gbr640000gp/T/ipykernel_21364/157641164
1.py:77: UserWarning: Glyph 21270 (\N{CJK UNIFIED IDEOGRAPH-5316}) missing
from font(s) DejaVu Sans.
  plt.tight_layout()
/Users/Zhuanz/Documents/Senior/PRML/.conda/lib/python3.12/site-packages/IP
ython/core/pylabtools.py:170: UserWarning: Glyph 26080 (\N{CJK UNIFIED IDE
OGRAPH-65E0}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
/Users/Zhuanz/Documents/Senior/PRML/.conda/lib/python3.12/site-packages/IP
ython/core/pylabtools.py:170: UserWarning: Glyph 27491 (\N{CJK UNIFIED IDE
OGRAPH-6B63}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
/Users/Zhuanz/Documents/Senior/PRML/.conda/lib/python3.12/site-packages/IP
ython/core/pylabtools.py:170: UserWarning: Glyph 21017 (\N{CJK UNIFIED IDE
OGRAPH-5219}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
/Users/Zhuanz/Documents/Senior/PRML/.conda/lib/python3.12/site-packages/IP
ython/core/pylabtools.py:170: UserWarning: Glyph 21270 (\N{CJK UNIFIED IDE
OGRAPH-5316}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```



```
==========================================================================
======

==========================================================================
======

1.           :
   - L1
   - L2
   -

2.         :
   -
   -
   -          0.2
```

1. sigmoid

- : sigmoid(z) ∈ (0,1) 0 1
- :
- : (0, 0.5)
- : s'(z) = s(z)(1-s(z))

2. MSE

-
  - MSE: $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$
  - : $J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$
-
  - : MSE sigmoid
  - : MSE sigmoid 0
  - :

Recall

= TP/(TP+FN) "

"

softmax

1. softmax

- : $\mathbf{x} = [x_1, x_2, \ldots, x_n]$
- : $\mathbf{p} = [p_1, p_2, \ldots, p_n]$ $\sum_{i=1}^{n} p_i = 1$

2. softmax

- : softmax
- :
- :

3. softmax

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \quad i = 1, \ldots, n$$

```python
def softmax_stable(x):
    #          softmax
    x_max = np.max(x, axis=-1, keepdims=True)
    exp_x = np.exp(x - x_max)  #
    return exp_x / np.sum(exp_x, axis=-1, keepdims=True)
```

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} = \frac{e^{x_i - c}}{\sum_j e^{x_j - c}}$$

$c = \max(x)$

-