Machine Learning Final Open Test

- **MSSV:** 19127191
- **Họ tên:** Ngô Văn Anh Kiệt
- **Lớp:** 19KHMT

1. Import các thư viện cần thiết

In [ ]:
```
import numpy as np
from matplotlib import pyplot as plt
from scipy import optimize # To run gradient descent on the cost function optimally
from scipy.special import expit
from sklearn.datasets import make_circles
from sklearn.metrics import classification_report
from dataclasses import dataclass
```

2. Giới thiệu về overfitting

Quá khớp (overfitting) là hiện tượng mô hình học quá khớp với dữ liệu huấn luyện, mất đi tính tổng quát hóa và linh hoạt của mô hình học máy. Điều này có thể dẫn đến dự báo sai khi mô hình được dùng cho dữ liệu mới mà chưa từng gặp qua. Có nhiều nguyên nhân dẫn đến overfitting, tiêu biểu như: không đủ dữ liệu huấn luyện, dữ liệu huấn luyện không đa dạng, mô hình học quá phức tạp, sử dụng quá nhiều đặc trưng không cần thiết,...
Hiện nay, hầu như các mô hình học máy đều phải quan tâm đến overfitting và nghiên cứu ra nhiều giải pháp để hạn chế vấn đề này. Bài báo cáo này sẽ giới thiệu một trong những phương pháp giảm overfitting thông dụng nhất, đó chính là kỹ thuật L2 Regularization, cụ thể là áp dụng cho mô hình hồi quy logistic để phân lớp dữ liệu.

3. Bài toán giả định

Giả sử chúng ta có tập dữ liệu huấn luyện gồm 100 mẫu. Mỗi mẫu có 2 đặc trưng X_1 với X_2, và được gắn cho một nhãn y có thể là 0 hoặc 1 (true hoặc false). Có thể thấy đây là kiểu bài toán thông dụng trong nhiều lĩnh vực đời sống khi ta cần phân loại các mẫu dữ liệu xem có thuộc một lớp đối tượng hay không. Ta sẽ sử dụng một mô hình học bằng hồi quy logistic để giải quyết bài toán này.

Trước hết, ta sẽ tạo một tập dữ liệu mẫu ngẫu nhiên để minh họa bài toán.

In [ ]:
```
# Random dataset of 100 samples with binary labels
X, y = make_circles(100, noise=0.15, factor=0.5, random_state=42)
print("X:", X.shape)
print("y:", y.shape)
print("Sample X[0]:", X[0])
print("Sample y[0]:", y[0])
```

```
X: (100, 2)
y: (100,)
Sample X[0]: [-0.17246763 -0.55817206]
Sample y[0]: 1
```

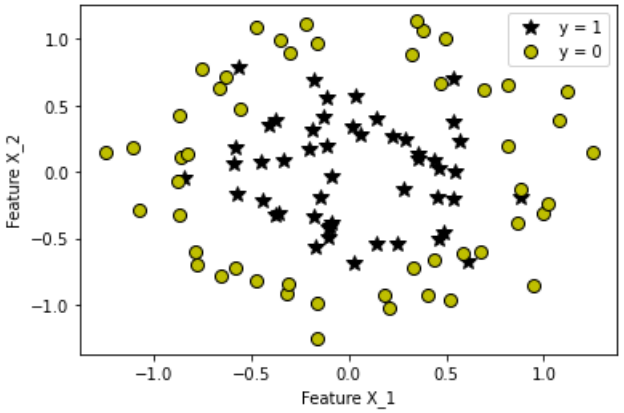
Trước hết, ta sẽ thử trực quan hóa dữ liệu để xem thử decision boundary (ranh giới quyết định giữa 2 nhãn) có hình dạng ra sao. Vẽ tập dữ liệu ra biểu đồ bằng hàm plotData.

In [ ]:
```
def plotData(X, y, ax):
    """
    Plots the data points X and y into a figure/subplot. Plots the data
    points with * for the positive samples and o for the negative samples.

    Parameters
    ----------
    X : array_like
        An Mx2 matrix representing the dataset.

    y : array_like
        Label values for the dataset. A vector of size (M, ).
    """

    # Find indices of positive and negative samples
```

```
    pos = y == 1
    neg = y == 0

    # Plot the samples
    ax.plot(X[pos, 0], X[pos, 1], 'k*', lw=2, ms=10)
    ax.plot(X[neg, 0], X[neg, 1], 'ko', mfc='y', ms=8, mec='k', mew=1)
```

In [ ]:

```
fig, ax = plt.subplots()
plotData(X, y, ax)
# Labels and legend
ax.set(xlabel="Feature X_1", ylabel="Feature X_2")

# Specified in plot order
ax.legend(['y = 1', 'y = 0'], loc='upper right')
plt.show()
```



Ta có thể thấy đường ranh giới phân loại chắc chắn phải có hình dạng gần giống đường tròn, sao cho các điểm dữ liệu có nhãn 1 nằm bên trong đường tròn và các điểm có nhãn 0 nằm bên ngoài đường tròn ấy.
Như vậy, rõ ràng là đường ranh giới này khá phức tạp để có thể được khớp chỉ với 2 đặc trưng. Vì thế, thông thường ta cần phải tự tạo hoặc tìm ra thêm các đặc trưng mới từ tập dữ liệu huấn luyện. Ở đây ta sẽ dùng kỹ thuật Feature Mapping để ánh xạ từ 2 đặc trưng ban đầu thành các đặc trưng mới sao cho là các đa thức của tích X_1X_2 từ bậc 0 đến 8.

In [ ]:

```
def mapFeature(X1, X2, degree=6):
    """
    Maps the two input features to quadratic features.

    Returns a new feature array with more features, comprising of
    X1, X2, X1.^2, X2.^2, X1*X2, X1*X2.^2, etc..

    Parameters
    ----------
    X1 : array_like
        A vector of shape (m, 1), containing one feature for all examples.

    X2 : array_like
        A vector of shape (m, 1), containing a second feature for all examples.
        Inputs X1, X2 must be the same size.

    degree: int, optional
        The polynomial degree.

    Returns
    -------
    : array_like
        A matrix of of m rows, and columns depend on the degree of polynomial.
    """
    if X1.ndim > 0:
        out = [np.ones(X1.shape[0])]
    else:
        out = [np.ones(1)]

    for i in range(1, degree + 1):
        for j in range(i + 1):
            out.append((X1 ** (i - j)) * (X2 ** j))
```

```
    if X1.ndim > 0:
        return np.stack(out, axis=1)
    else:
        return np.array(out)
```

In [ ]:
```
X = mapFeature(X[:, 0], X[:, 1], 8)
print("Shape of X:", X.shape)
print("Sample X[0]:", X[0])
```

```
Shape of X: (100, 45)
Sample X[0]: [ 1.00000000e+00 -1.72467631e-01 -5.58172063e-01  2.97450838e-02
  9.62666135e-02  3.11556052e-01 -5.13006415e-03 -1.66028748e-02
 -5.37333343e-02 -1.73901884e-01  8.84770012e-04  2.86345849e-03
  9.26726088e-03  2.99924460e-02  9.70671735e-02 -1.52594188e-04
 -4.93853903e-04 -1.59830253e-03 -5.17272612e-03 -1.67409455e-02
 -5.41801845e-02  2.63175582e-05  8.51738128e-05  2.75655452e-04
  8.92127822e-04  2.88727121e-03  9.34432808e-03  3.02418654e-02
 -4.53892692e-06 -1.46897257e-05 -4.75416428e-05 -1.53863172e-04
 -4.97960827e-04 -1.61159413e-03 -5.21574288e-03 -1.68801644e-02
  7.82817974e-07  2.53350220e-06  8.19939452e-06  2.65364168e-05
  8.58821242e-05  2.77947822e-04  8.99546820e-04  2.91128196e-03
  9.42203617e-03]
```

Có thể thấy, từ 2 đặc trưng ban đầu, ta đã biến tập dữ liệu thành 45 đặc trưng mới. Chắc chắn với tập dữ liệu này, mô hình của chúng ta sẽ khá khớp với hình dạng phức tạp của đường ranh giới. Tuy nhiên với số lượng đặc trưng nhiều thế này thì việc mô hình học trở nên quá khớp với dữ liệu huấn luyện là không thể tránh khỏi. Vì thế ta sẽ áp dụng kĩ thuật L2 Regularization vào mô hình hồi quy logistic để giảm tác động của overfitting lại.

4. Sử dụng L2 Regularization (Chính quy hóa) cho mô hình hồi quy logistic

Nhắc lại công thức sigmoid, cost và gradient của hồi quy logistic:

- Sigmoid:

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

- Cost:

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

- Gradient:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Vậy L2 Regularization là gì, và nó được áp dụng thế nào?

L2 regularization là một kỹ thuật thông dụng để giảm overfitting trong các lĩnh vực học máy như hồi quy hoặc mạng nơ-ron. Trong lĩnh vực hồi quy, nó còn được gọi là *Ridge Regression*; còn trong mạng nơ-ron, nó được gọi là *Weight Decay* (tiêu biến trọng số).

Kỹ thuật này thực chất là một phần công thức mới được thêm vào công thức cost và gradient ban đầu:

- Cost:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

- Gradient:

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)} \qquad \text{với } j = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j \qquad \text{với } j \geq 1$$

Trong đó, λ là tham số huấn luyện do người huấn luyện mô hình tự đặt ra.

Ý tưởng cơ bản của L2 Regularization là giảm đi trọng số của các đặc trưng gây ảnh hưởng mạnh đến mô hình trong quá trình Gradient Descent, mà các đặc trưng có ảnh hưởng mạnh này vốn là nguyên nhân gây overfitting. Điều này nghĩa là, nếu trọng số nào ban đầu càng lớn, thì sau một lần lặp Gradient Descent, trọng số đó sẽ bị giảm đi càng mạnh. Nhưng các trọng số bị giảm mạnh đó không biến mất hoàn toàn, mà chỉ tiệm cận 0. Như vậy, chỉ với một thay đổi đơn giản vào công thức cost và gradient, L2 regularization giúp ta cân bằng độ ảnh hưởng của các đặc trưng trong quá trình huấn luyện, nhờ vậy mà mô hình học sẽ không bị overfitting.

Một điểm quan trọng cần lưu ý là ta sẽ không áp dụng regularization cho trọng số θ_0 (như công thức trên) vì đây thực chất là giá trị bias của mô hình học chứ không phải là trọng số của một đặc trưng trong dữ liệu huấn luyện.

Giờ ta sẽ tạo một mô hình hồi quy logistic có áp dụng L2 regularization để thực hiện hồi quy trên tập dữ liệu đã biến đổi mà hạn chế được overfitting. Ta chỉ việc tạo hàm hàm tính cost và gradient, sau đó sẽ để cho hàm optimize.minimize của thư viện scipy thực hiện Gradient Descent. Hàm tính sigmoid sẽ dùng hàm thư viện scipy.special.expit để tính vì được tối ưu tốt hơn.

In [ ]:

```
def L2RegCost(theta, X, y, lambda_):
    """
    Compute cost and gradient for logistic regression with L2 regularization.

    Parameters
    ----------
    theta : array_like
        Logistic regression parameters. A vector with shape (n, ). n is
        the number of features including any intercept. If we have mapped
        our initial features into polynomial features, then n is the total
        number of polynomial features.

    X : array_like
        The data set with shape (m x n). m is the number of examples, and
        n is the number of features (after feature mapping).

    y : array_like
        The data labels. A vector with shape (m, ).

    lambda_ : float
        The regularization parameter.

    Returns
    -------
    J : float
        The computed value for the regularized cost function.

    grad : array_like
        A vector of shape (n, ) which is the gradient of the cost
        function with respect to theta, at the current values of theta.
    """
    # Initialize some useful values
    m = y.shape[0]  # number of training examples
    J = 0
    grad = np.zeros(theta.shape)
    epsilon = 1e-5 # To fix when the input of log function is too small that it becomes zero

    # Calculate sigmoid hypothesis
    h = expit(X @ theta)
    # # Temporarily replace theta_0 with 0 for regularization
    theta_0 = theta[0]
    theta[0] = 0
    # Calculate cost J and gradient
    regTerm = (lambda_ / (2.0 * m)) * np.sum(theta ** 2)
    J = (-1.0 * np.transpose(y) @ np.log(h + epsilon) - np.transpose(1.0 - y) @ np.log(1.0 - h + epsilon)) / m
    J += regTerm
    grad = (X.T @ (h - y)) / m + (lambda_ * theta) / m
    # Restore the original theta_0
    theta[0] = theta_0

    return J, grad
```

Sau khi có các hàm trên, ta có thể dùng hàm scipy.optimize.minimize để huấn luyện mô hình học

In [ ]:
```
@dataclass
class Model:
    cost: float
    theta: np.ndarray

    def predict(self, X) -> np.ndarray:
        p = expit(X @ self.theta.T) >= 0.5
        return p
```

In [ ]:
```
def fit(X, y, lambda_, iterations, minimize_method="TNC"):
    """Fit the logistic regression model using the training data with L2 regularization.

    Parameters:
        X (array_like): The feature matrix of the training data
        y (array_like): The label vector of the training data
        lambda_ (float): The hyperparameter for tuning the L2 regularization, minimum is 0.
        iterations (int): Maximum number of iterations for the fitting process
        minimize_method (str, optional): The method to fit the data. Defaults to "TNC"
                                         (See scipy.optimize.minimize for available methods).

    Returns:
        A model with these attributes:
        - cost (float): the value of cost function at optimized theta
        - theta (array_like): the optimized theta is in the x property of the result
    """
    # Initialize fitting parameters
    initial_theta = np.zeros(X.shape[1])
    if lambda_ < 0:
        lambda_ = 0

    # set options for optimize.minimize
    options = {'maxiter': iterations}

    res = optimize.minimize(L2RegCost,
                            initial_theta, (X, y, lambda_),
                            jac=True,
                            method=minimize_method,
                            options=options)

    # the fun property of OptimizeResult object returns:
    #   fun: the value of cost function at optimized theta
    #   x: the optimized theta is in the x property of the result
    return Model(res.fun, res.x)
```

Ta sẽ huấn luyện 3 mô hình với 3 giá trị tham số λ khác nhau để đánh giá mức độ ảnh hưởng của L2 Regularization đến kết quả dự đoán. Các mô hình này sẽ có cùng số lần huấn luyện, cùng tập dữ liệu huấn luyện và cùng phương pháp minimize.

In [ ]:
```
models: list[Model] = [] # List of trained models
lambdas = [0.0, 1.0, 100.0]
for i in range(3):
    models.append(fit(X, y, lambdas[i], 100))
```

5. Thử nghiệm và đánh giá mô hình

Để có thể quan sát kết quả dự đoán của mô hình một cách trực quan, ta sẽ viết một hàm dùng để vẽ đường ranh giới phân lớp của tập dữ liệu dựa theo vector trọng số mà mô hình đã học được.

In [ ]:
```
def plotDecisionBoundary(plotData, theta, X, y, lambda_, ax):
    """
    Plots the data points X and y into a new figure with the decision boundary defined by theta.
    Plots the data points with * for the positive samples and o for the negative samples.

    Parameters
    ----------
    plotData : func
        A function reference for plotting the X, y data.

    theta : array_like
        Parameters for logistic regression. A vector of shape (n+1, ).

    X : array_like
```

```
        The input dataset. X is assumed to be  a either:
            1) Mx3 matrix, where the first column is an all ones column for the intercept.
            2) MxN, N>3 matrix, where the first column is all ones.

    y : array_like
        Vector of data labels of shape (m, ).
    """
    # make sure theta is a numpy array
    #theta = np.array(theta)

    # Plot Data (remember first column in X is the intercept)
    plotData(X[:, 1:3], y, ax)

    # Here is the grid range
    u = np.linspace(-1.5, 1.5, 50)
    v = np.linspace(-1.5, 1.5, 50)

    z = np.zeros((u.size, v.size))
    # Evaluate z = theta*x over the grid
    for i, ui in enumerate(u):
        for j, vj in enumerate(v):
            temp = np.array([[ui, vj]]).reshape((1,2))
            temp = mapFeature(temp[:,0], temp[:,1], 8)
            z[i, j] = temp @ theta

    z = z.T  # important to transpose z before calling contour

    # Plot z = 0
    ax.contour(u, v, z, levels=[0], linewidths=2, colors='g')
    ax.contourf(u, v, z, levels=[np.min(z), 0, np.max(z)], cmap='Greens', alpha=0.4)

    # Set labels, legend and title
    ax.set(xlabel="Feature X_1", ylabel="Feature X_2")
    ax.legend(['y = 1', 'y = 0'], loc='upper right')
    ax.grid(False)
    ax.set_title('lambda = %0.2f' % lambda_)
```

Đầu tiên, ta có kết quả phân loại của 3 mô hình đã được huấn luyện

In [ ]:

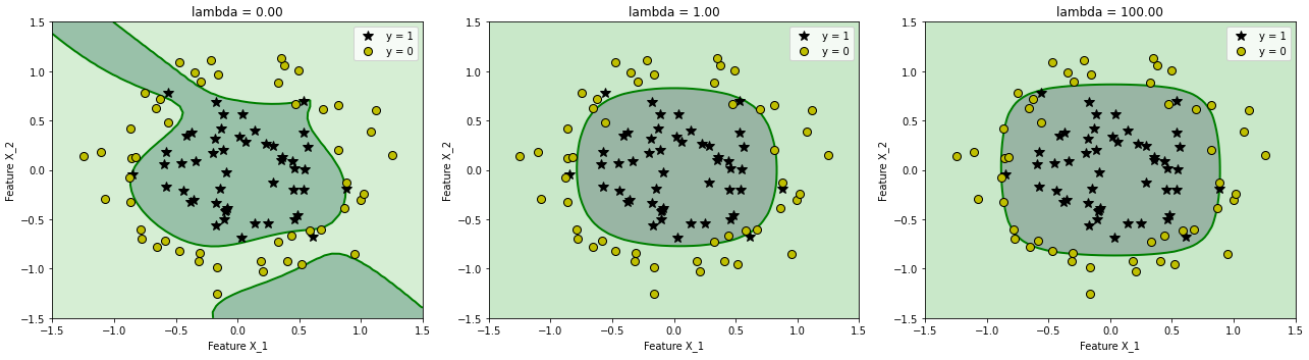
```
for i in range(3):
    print("Kết quả của mô hình", i + 1, "với lambda =", lambdas[i])
    print(classification_report(y, models[i].predict(X)))
    print()
```

```
Kết quả của mô hình 1 với lambda = 0.0
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        50
           1       0.98      0.96      0.97        50

    accuracy                           0.97       100
   macro avg       0.97      0.97      0.97       100
weighted avg       0.97      0.97      0.97       100


Kết quả của mô hình 2 với lambda = 1.0
              precision    recall  f1-score   support

           0       0.92      0.88      0.90        50
           1       0.88      0.92      0.90        50

    accuracy                           0.90       100
   macro avg       0.90      0.90      0.90       100
weighted avg       0.90      0.90      0.90       100


Kết quả của mô hình 3 với lambda = 100.0
              precision    recall  f1-score   support

           0       1.00      0.60      0.75        50
           1       0.71      1.00      0.83        50

    accuracy                           0.80       100
   macro avg       0.86      0.80      0.79       100
weighted avg       0.86      0.80      0.79       100
```

- Có thể thấy, khi λ càng tăng thì hiệu quả của L2 regularization càng mạnh, độ chính xác của mô hình sẽ càng giảm. Một điều đáng lưu ý là dựa trên công thức hàm cost với regularization, khi $\lambda = 0$ thì cũng đồng nghĩa với việc regularization bị vô hiệu, mô hình cũng giống như khi chưa dùng kĩ thuật đó. Chính vì điều này nên ta thấy độ chính xác của mô hình 1 là cao nhất với 97%. Đây là dấu hiệu của overfitting.
- Ngược lại, độ chính xác của mô hình 3 thì thấp hơn nhiều so với 2 mô hình trước chỉ với khoảng 80%, đây là dấu hiệu của underfitting.
- Mô hình 2 được huấn luyện với giá trị $\lambda = 1$ có vẻ cho độ chính xác lý tưởng nhất với 90%.

Để thấy rõ hơn tác động của L2 Regularization, ta sẽ vẽ ra đường ranh giới phân loại của 3 mô hình này.

In [ ]:
```
fig, ax = plt.subplots(1, 3, figsize=(5 * 3 + 3, 5))
for i in range(3):
    plotDecisionBoundary(plotData, models[i].theta, X, y, lambdas[i], ax[i])
plt.tight_layout()
plt.show()
```



- Như đã thấy ở 3 hình trên, mô hình 1 cho ra đường ranh giới rất bừa bãi khi nó cố bao hết tất cả mẫu dữ liệu của nhãn y = 1, thậm chí còn có vùng chẳng có nhãn 1 nào nhưng vẫn được tính, vậy là mô hình 1 thật sự bị overfitting do đã không dùng L2 Regularization.
- Ngược lại, ở mô hình 3 do đặt giá trị λ quá cao để huấn luyện, mô hình đã bị underfitting. Quan sát thấy đường ranh giới của mô hình này cũng cố bao trọn các mẫu mang nhãn 1 như mô hình đầu, nhưng khi làm thế, nó lại bao luôn cả rất nhiều các mẫu nhãn 0.
- Cuối cùng, ta có mô hình 2. Đường ranh giới của mô hình này không cố bao trọn tất cả mẫu nhãn 1 mà chỉ gói gọn quanh vùng quan trọng mà phần lớn các mẫu nằm trong đó. Các mẫu nhãn 1 nằm ngoài có thể được coi là ngoại lệ. Như vậy, mô hình này có đường ranh giới tương đối khớp với dữ liệu huấn luyện.

Từ các kết quả trên, ta có thể kết luận rằng mô hình 2 là tốt nhất và sử dụng tốt L2 Regularization với tham số huấn luyện $\lambda = 1$.

6. Kết luận

- L2 Regularization là một trong những phương pháp chống overfitting thông dụng nhất được ứng dụng cho mô hình hồi quy và mạng nơ-ron. Tuy nhiên, ta cũng phải lưu ý về tham số huấn luyện khi sử dụng phương pháp này. Nếu ta đặt λ quá thấp thì mô hình vẫn sẽ bị overfitting, ngược lại nếu λ quá cao thì sẽ dẫn đến underfitting. Thông thường thì giá trị λ mà ta nên bắt đầu thử nghiệm nên là 1 sau đó tùy theo kết quả huấn luyện mà tăng hoặc giảm.
- L2 Regularization thích hợp để sử dụng khi ta muốn huấn luyện một mô hình phức tạp với dữ liệu có nhiều đặc trưng. Kỹ thuật này sẽ giúp mô hình khớp được các dữ liệu tương đối phức tạp hoặc thậm chí không tuyến tính, đặc biệt là khi hàm đầu ra sử dụng hết tất cả đặc trưng của tập dữ liệu.
- Kỹ thuật này không linh hoạt với các mẫu dữ liệu ngoại lệ (như đã thấy trong hình vẽ ranh giới của 3 mô hình trên).
- Nhìn chung thì L2 Regularization giúp tăng độ chính xác cho mô hình rất tốt trong hầu hết các trường hợp và được ứng dụng rộng rãi.

7. Tham khảo

- Bài giảng lý thuyết của môn học (Slide và video record) - Thầy Bùi Tiến Lên.
- Khóa học Machine Learning trên Coursera - bởi Andrew Ng.