Machine Learning TP2

Félix HENAFF & Alfred DEDUYER, FIPA25

This report focuses on implementing logistic and regularized logistic regression for classification tasks. We develop cost and gradient functions, optimize parameters, and evaluate the model's performance. The effect of regularization is explored to prevent overfitting, and a one-vs-all approach is applied for multi-class classification. The models show strong accuracy, confirming the correctness of the implementation.

3 Logistic Regression

3. 1 Visualizing the data

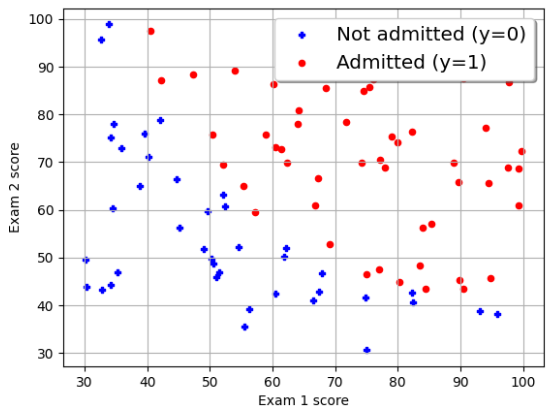

Figure 1 Scatter plot of training data

```python
def plotData(X,y):

    cmap = 'bwr' # 'RdBu'
    cmap = plt.get_cmap(cmap)

    pos = X[(y == 1).flatten(), :]
    neg = X[(y == 0).flatten(), :]

    plt.figure()

    plt.scatter(neg[:, 0], neg[:, 1], s=20, marker='P', color=cmap(0))
    plt.scatter(pos[:, 0], pos[:, 1], s=20, marker='o', color=cmap(cmap.N))

    plt.xlabel('Exam 1 score')
    plt.ylabel('Exam 2 score')

    plt.grid(True)
    plt.legend(['Admitted (y=1)', 'Not admitted (y=0)'], loc='upper right',
shadow=True, fontsize = 'x-large', numpoints = 1)
```

3.2 Implementation

3.2.1 Sigmoid function

```
def sigmoid(z):
    """computes the sigmoid of z."""
    g = 1 / (1 + np.exp(-z))
    return g
```

The sigmoid function is correctly implemented, returning a value of 0.5 for an input of 0, as expected. For large positive inputs like 100, it approaches 1, and for large negative inputs like -100, it approaches 0. These results confirm the proper behavior of the sigmoid function.

3.2.2 Cost function and gradient

```
def costFunction(theta, X, y):
    """ computes the cost of using theta as the
    parameter for logistic regression."""

    m, n = X.shape
    theta = theta.reshape((n, 1))
    m = len(y)
    h = sigmoid(np.dot(X, theta))
    J = (-1 / m) * (np.dot(y.T, np.log(h)) + np.dot((1 - y).T, np.log(1 -
h)))

    return J
```

```
def gradientFunction(theta, X, y):
    """
    Compute cost and gradient for logistic regression with regularization

    computes the cost of using theta as the parameter for regularized
logistic
    regression and the gradient of the cost w.r.t. to the parameters.
    """

    m = X.shape[0]

    n = X.shape[1]
    theta = theta.reshape((n, 1))
    m = len(y)
    h = sigmoid(np.dot(X, theta))
    grad = (1/m) * np.dot(X.T, (h - y))

    return grad
```

The cost function and gradient are correctly implemented, as the cost at the initial theta (zeros) is 0.693147, which matches the expected value. This confirms that the cost and gradient calculations for logistic regression are functioning as expected.

3.2.3 Learning parameters using fmin_tnc

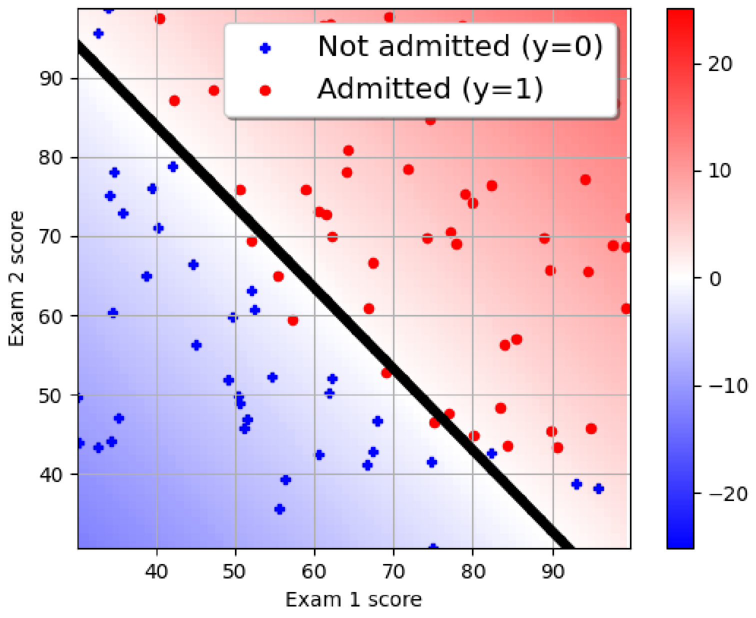

Figure 2 Training data with decision boundary

The optimization using fmin_tnc successfully minimized the cost to 0.203498, which matches the expected cost. Additionally, the learned theta values closely align with the expected ones, confirming that the implementation of the logistic regression model is correct and functioning as expected.

3.2.4 Evaluating logistic regression

The logistic regression model predicts an admission probability of 0.776291 for a student with exam scores of 45 and 85, which matches the expected value. This consistency indicates that the model is accurately calculating probabilities, confirming the correctness of the implementation.

4 Regularized logistic regression

4.1 Visualizing the data

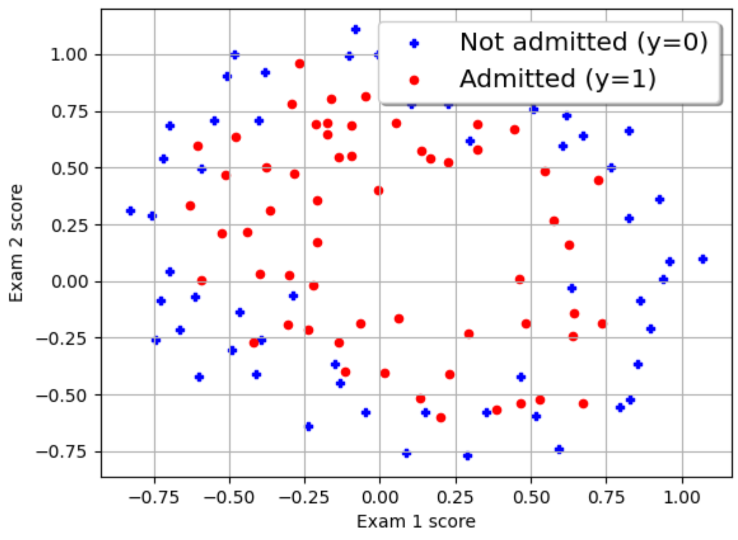

Figure 3 Plot of training data.

4.3 Cost function and gradient

```
def costFunctionReg(theta, X, y, Lambda):
    """
    Compute cost for logistic regression with regularization

    Computes the cost of using theta as the parameter for regularized
logistic regression.
    """

    theta = theta.reshape((n, 1))

    h = sigmoid(X @ theta)

    term1 = -y.T @ np.log(h)
    term2 = -(1 - y).T @ np.log(1 - h)
    cost = (term1 + term2) / m

    reg_term = (Lambda / (2 * m)) * np.sum(np.square(theta[1:]))
    J = cost + reg_term

    return J
```

```python
def gradientFunctionReg(theta, X, y, Lambda):
    """
    Compute cost and gradient for logistic regression with regularization

    computes the gradient of the cost w.r.t. to the parameters.
    """

    # Initialize some useful values
    m, n = X.shape  # number of training examples and parameters
    theta = theta.reshape((n, 1))  # due to the use of fmin_tnc

    # Calculate the hypothesis
    h = sigmoid(X @ theta)

    # Compute the gradient
    error = h - y

    # Compute the gradient for theta_0 (no regularization)
    grad = (X.T @ error) / m

    # Add regularization to the remaining theta terms (except theta_0)
    grad[1:] = grad[1:] + (Lambda / m) * theta[1:]

    return grad
```

The implemented cost and gradient functions for regularized logistic regression work as expected, as the initial cost at theta = 0 matches the expected value of approximately 0.693147, confirming the correctness of the implementation.

4.3.1 Learning parameters using fmin_tnc

The optimal theta values found using fmin_tnc result in a cost of 0.529003, which closely matches the expected cost of approximately 0.5290. This indicates that the optimization process successfully minimized the cost function, and the model is performing as expected.

4.4 plotting the decision boundary

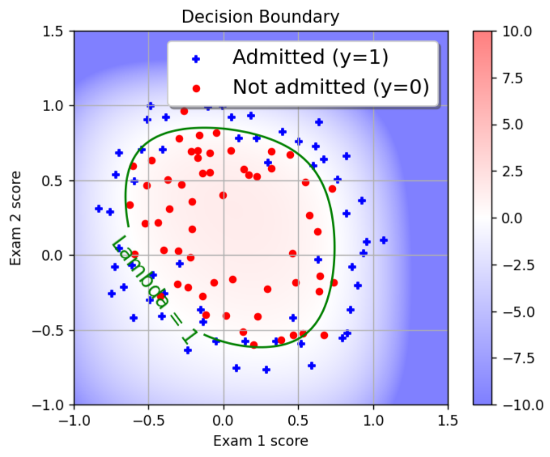

Figure 4 Training data with decision boundary Lamda = 1

4.5 Impact of λ

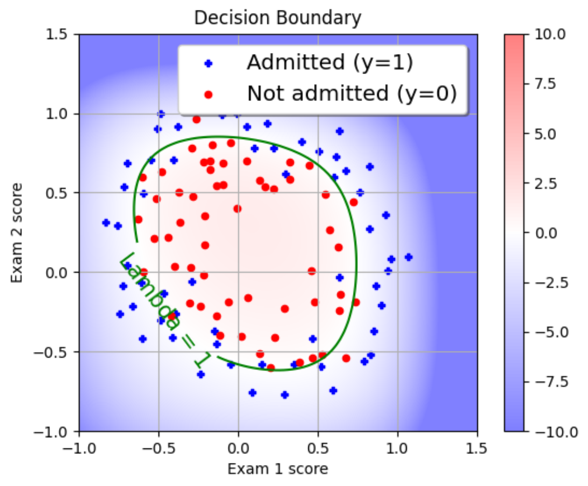

Figure 5 Training data with decision boundary (λ = 1)

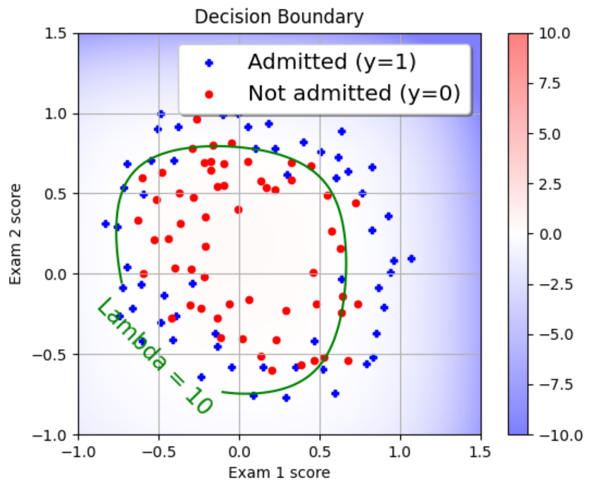

Figure 6 Too much regularization (Underfitting) (λ = 10)

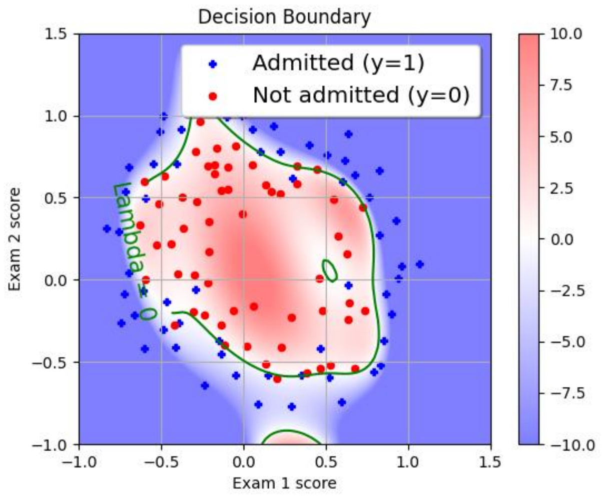

Figure 7 No regularization (Overfitting) (λ = 0)

5 Multi-class classification

5.1 Dataset

5.2 Visualizing the data

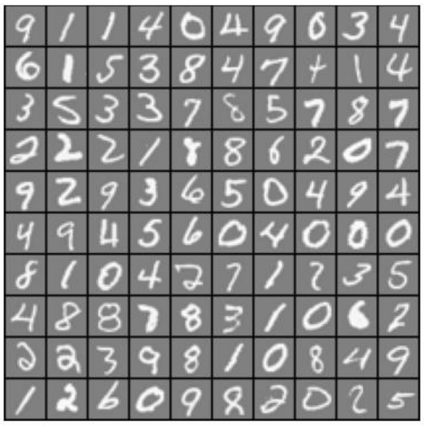

Figure 8 Examples from the dataset

5.4 Learning a one-vs-all classifier

```
def predictOneVsAll(all_theta, X):
    """will return a vector of predictions
    for each example in the matrix X. Note that X contains the examples in
    rows. all_theta is a matrix where the i-th row is a trained logistic
    regression theta vector for the i-th class. You should set p to a
vector
    of values from 1..K (e.g., p = [1 3 1 2] predicts classes 1, 3, 1, 2
    for 4 examples) """

    m = X.shape[0]

    p = np.zeros((m, 1))

    p = sigmoid(X.dot(all_theta.T))
    p = np.argmax(p, axis=1)

    return np.atleast_2d(p).T
```

The results show that the One-vs-All logistic regression classifier successfully optimized the model for each digit class, achieving a training set accuracy of 96.46%, which matches the expected accuracy. This indicates that the classifier is well-trained and performs as anticipated on the dataset.

5.6 Some questions, you have to answer…

1. Identifiez les différences d'approches entre le TP1 et le TP2 et les différences de résolution du problème.

Le TP1 est axé sur la prédiction de valeurs continues avec une régression linéaire classique.

Le TP2 introduit des concepts plus avancés comme la régression logistique, la classification, la régularisation…

Cette distinction montre la progression du traitement de données continues simples à des problèmes plus complexes de classification avec différentes classes.

2. A quoi sert le principe de régularisation ?

La régularisation aide à éviter le surapprentissage en pénalisant les grands coefficients du modèle. Cela empêche le modèle de s'ajuster trop étroitement aux données d'entraînement, améliorant ainsi sa capacité à généraliser sur de nouvelles données.

3. Décrivez le problème du sur-apprentissage et comment on y répond dans ce TP. Quelle est l'influence de la valeur de λ?

Le sur-apprentissage se produit lorsque le modèle s'ajuste trop aux données d'entraînement, capturant le bruit, ce qui nuit à sa performance sur de nouvelles données. Dans ce TP, on y répond par la régularisation, qui contrôle la taille des coefficients pour éviter un modèle trop complexe.

Le paramètre λ contrôle cette régularisation :

Petit λ : risque de sur-apprentissage.
Grand λ : risque de sous-apprentissage si trop élevé.

4. Dans quelles circonstances et pour quelles raisons utilise-t-on l'approche multi classe ? Quelles sont les différentes possibilités pour cette approche ?

L'approche multiclasse est utilisée lorsque le problème implique plusieurs catégories. On utilise des techniques comme One-vs-All ou One-vs-One pour diviser le problème en classifications binaires, ou des algorithmes capables de traiter plusieurs classes directement.