Project 1: Logistic Regression

This Project gives extra points for the final grade

Due 27.11.2025, 16:00

Overview

Submit your project solution as a group of 2-4 people.

In this project, you will train a **Logistic Regression** model on a non-linear dataset. The following steps serve as a guide:

Tasks

1. **Feature Engineering**
    - Since Logistic Regression is a linear classifier, you need to perform feature engineering to make the data linearly separable
    - Consider preprocessing original features
    - Figure out what extra features you should add to the data
2. **Implement Training Loop**
    - Implement the training loop for your Logistic Regression model using **Gradient Descent**
    - Use **BCE Loss** (Binary Cross-Entropy)
    - Incorporate a **regularization** technique to prevent overfitting
3. **Hyperparameter Tuning**
    - Train your model and find good hyperparameters using the **validation set**
4. **Visualization**
    - Visualize the **loss and accuracy curves** of the training process
5. **Testing & Evaluation**

- Test your trained model on the given **test set**
- Output the final accuracy rate

For full points you need to atleast achieve 73% accuracy on the testing set

NOTE:

You are not allowed to use sklearn, pytorch or similar Deep Learning frameworks. Numpy is allowed.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
DATA_PATH = "C:/Users/t00002k9/OneDrive - Trench Group/Desktop/doc/hhu/machine

df = pd.read_csv(DATA_PATH)
X = df[["x1", "x2"]].values.astype(float)
y = df["y"].values.astype(int)
print(f"Dataset shape: {X.shape}")
print(f"Classes: {np.unique(y)}")
print(f"\nFirst 5 samples:")
print(df.head())

train_split = int(len(X) * 0.7)
val_split = int(len(X) * 0.85)

x_train = X[:train_split]
x_val = X[train_split:val_split]
x_test = X[val_split:]
y_train = y[:train_split]
y_val = y[train_split:val_split]
y_test = y[val_split:]
print(f"\nTrain set: {x_train.shape[0]} samples")
print(f"Validation set: {x_val.shape[0]} samples")
print(f"Test set: {x_test.shape[0]} samples")
```

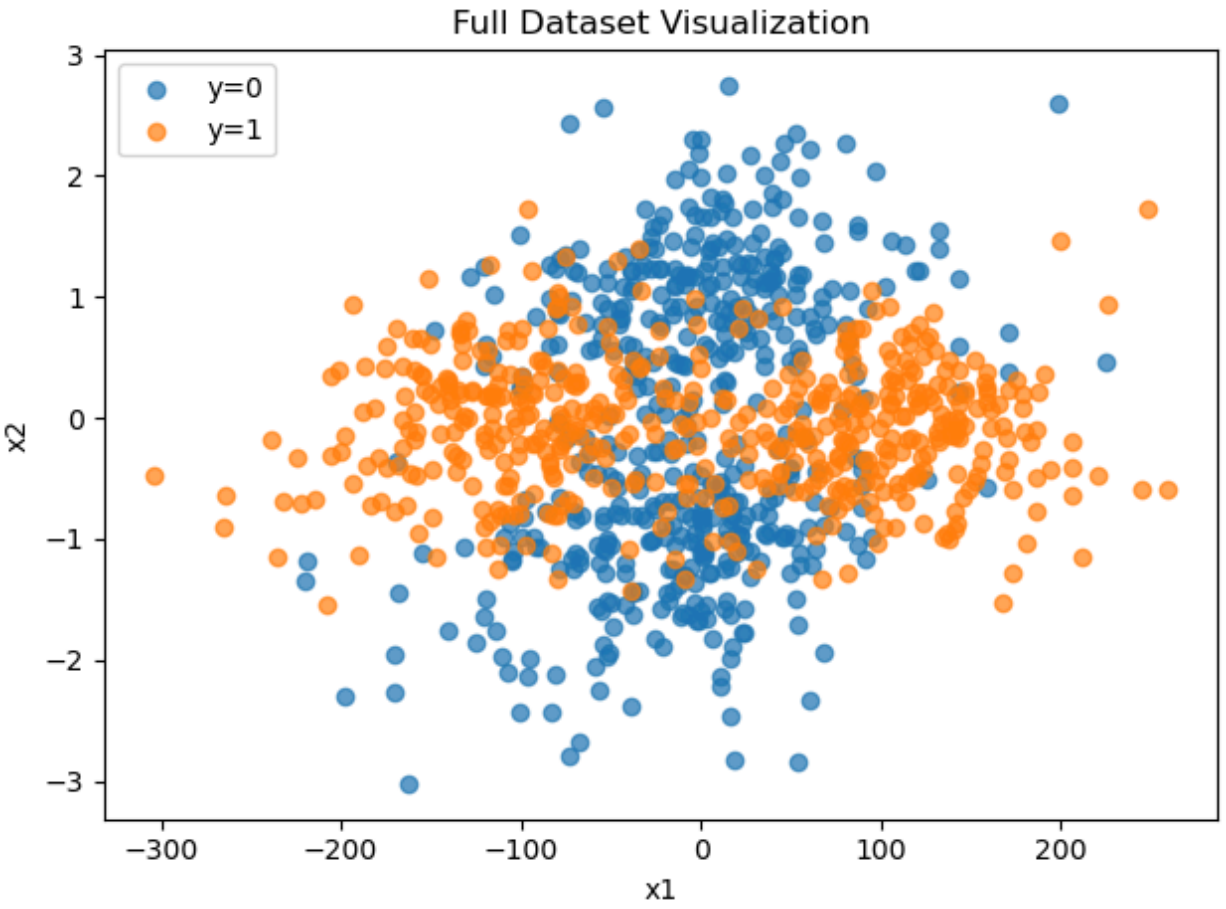
```
Dataset shape: (1000, 2)
Classes: [0 1]

First 5 samples:
          x1        x2  y
0   9.492170  0.272589  0
1  16.829156 -0.787517  0
2   0.175471  0.412100  1
3  80.509510 -0.688343  1
4  68.193111  1.447592  0

Train set: 700 samples
Validation set: 150 samples
Test set: 150 samples
```

In [3]:
```python
def visualize(X, y, title="Data Visualization"):
    cls0 = y == 0
    cls1 = y == 1
    plt.figure()
    plt.scatter(X[cls0, 0], X[cls0, 1], label="y=0", alpha=0.7)
    plt.scatter(X[cls1, 0], X[cls1, 1], label="y=1", alpha=0.7)
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.title(title)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

In [4]:
```python
visualize(X, y, title="Full Dataset Visualization")
```



(1) Feature Engineering: 2 Points

Since logistic regression is a linear classifier, one can not expect to perform very well on data that is not linearly seperable.

Try to figure out how to transform the data, e.g preprocessing original features and adding new features, such that the data becomes linearly seperable.

In [ ]:
```python
# TODO: (1) Feature Engineering to prepare data for logistic regression
```

(2) Training Logic: 2 Points

Implement the training logic by finding the best model parameters for logistic regression. Since we have only 2 classes, BCE Loss is a proper loss function.

Consider implementing regularization to avoid overfitting.

The training loop should take the training set (- and validation set for hyperparameter tuning later), aswell as the hyperparameters, initialize weights and do gradient descent.

You should track the losses and accuracies throughout the epochs for later visualization.

In [ ]:
```
# TODO: (2) implement training loop with BCE Loss
# hint: consider regularization to avoid overfitting
# you should return the loss and accuracy values per epoch for later visualiza


# These following functions are only meant to be a guideline, you can implemen
def sigmoid(z):
    return

def predict(X, W, b):
    return

def bce_loss(W, b, X, y):
    return

def grad_bce_with_l2(W, b, X, y, lam=0.0):
    return

def accuracy(y, p):
    return

def train_model(X_tr, y_tr, X_val, y_val, lr=0.1, lam=0.0, epochs=100):
    return
```

(3) Hyperparameter tuning: 0.5 Points

Your logistic regression model has hyperparameters, e.g. learning rate/ regularization lambda, and since there are no best hyperparameters in general, one has to find them by grid searching through them for the specific model.

Implement the grid search function by going through all possible defined parameters, train the model and evaluate on validation set.

Save the best hyperparameters and return them.

In [ ]:
```python
# TODO: (3) implement hyperparameter tuning

# Fill in learning rates and lambda values to search
lr_list = []
lambda_list = []
def grid_search(X_tr, y_tr, X_val, y_val, lr_list, lambda_list,  epochs=100):
    best_lr = None
    best_lambda = None
    return best_lr, best_lambda
```

(4) Training and (5) Testing: 0.5 Points

- Use grid search to find the best hyperparameters
- train your model with found hyperparameters
- visualize loss and accuracy curves
- test your model on the testing set

In [ ]:
```python
# TODO: (4) use grid search to find the best hyperparameters

best_lr, best_lambda = grid_search(x_train, y_train, x_val, y_val, lr_list, la

print(f"Best hyperparameters: lr={best_lr}, lam={best_lambda}")
```

In [ ]:
```python
# TODO: train your model with found hyperparameters
... = train_model(...)
```

In [ ]:
```python
# visualize loss and accuracy curves
def plot_curves(train_values, val_values, ylabel, title):
    plt.figure()
    plt.plot(train_values, label="Train")
    plt.plot(val_values, label="Validation")
    plt.xlabel("Epochs")
    plt.ylabel(ylabel)
    plt.title(title)
    plt.legend()
    plt.tight_layout()
    plt.show()


plot_curves(train_losses, val_losses, ylabel="Loss", title="Training and Valic
plot_curves(train_accs, val_accs, ylabel="Accuracy", title="Training and Valic
```

In [ ]:
```python
# TODO: (5) test your model on the test set
```