STAT3612 Assignment 2

Kevin Choy Guang Ming
3036461793

Q1) C

Q2) B

Q3) $$p(Y=1 \mid X) = \frac{e^{\theta_0 + \theta_1 X_1 + \theta_2 X_2}}{1 + e^{\theta_0 + \theta_1 X_1 + \theta_2 X_2}}$$

$\hat{\theta}_0 = -6$, $\hat{\theta}_1 = 0.05$, $\hat{\theta}_2 = 1$

a) $$P(Y=1 \mid x_1 = 40, x_2 = 3.5) = \frac{e^{-6+0.05(40)+1(3.5)}}{1 + e^{-6+0.05(40)+1(3.5)}}$$

$$= 0.3775$$

b) $P(Y=1 \mid X_1, x_2 = 3.5) = 0.50$

$$0.50 = \frac{e^{-6+0.05(X_1)+3.5}}{1 + e^{-6+0.05X_1+3.5}}$$

$$0.50 + 0.5e^{-2.5+0.05X_1} = e^{-2.5+0.05X_1}$$

$$0.5 = 0.5e^{-2.5+0.05X_1}$$

$$1 = e^{-2.5+0.05X_1}$$

$$0 = -2.5 + 0.05X_1$$

$$0.05X_1 = 2.5$$

$$X_1 = 50 \ \#$$

Q4)

$$L(\hat{w}) = -\frac{1}{n}\sum_{i=1}^{n} \log \frac{e^{\eta_{Y_i}(\hat{x}_i)}}{\sum_{j=1}^{K} e^{\eta_j(\hat{x}_i)}}$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\eta_{Y_i}(\hat{x}_i) - \log\left(\sum_{j=1}^{K} e^{\eta_j(\hat{x}_i)}\right)\right]$$

$$\frac{\partial L(\hat{w})}{\partial \eta_k(\hat{x}_i)} = -\frac{1}{n}\left[1_{\{k=Y_i\}} - \frac{e^{\eta_k(\hat{x}_i)}}{\sum_{j=1}^{K} e^{\eta_j(\hat{x}_i)}}\right], \quad \text{where } 1_{\{k=Y_i\}} = \begin{cases} 1, & \text{if } k = Y_i \\ 0, & \text{otherwise} \end{cases}$$

$$= -\frac{1}{n}\left[1_{\{k=Y_i\}} - \hat{y}_{ik}\right]$$

$$\eta_k(\hat{x}_i) = \hat{w}_k^T \hat{x}_i$$

$$\frac{\partial \eta_k(\hat{x}_i)}{\partial \hat{w}_k} = \hat{x}_i$$

$$\frac{\partial L(\hat{w})}{\partial \hat{w}_k} = \sum_{i=1}^{n} \frac{\partial L(\hat{w})}{\partial \eta_k(\hat{x}_i)} \cdot \frac{\partial \eta_k(\hat{x}_i)}{\partial \hat{w}_k}$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[1_{\{k=Y_i\}} - \hat{y}_{ik}\right]\hat{x}_i$$

Stacking gradients from all K classes to obtain ∇L against weight matrix W:

$$\nabla_{\hat{W}} L(\hat{w}) = \frac{1}{n}\sum_{i=1}^{n}[\hat{y}_i - y_i]\,\hat{x}_i^T$$

$$= \frac{1}{n}\left[\text{softmax}(\hat{W}\hat{X}) - Y\right]X^T$$

#

Q5) C

Q6) $X_{Yes} \sim N(10, 36)$

$X_{No} \sim N(0, 36)$

$\pi_k = 0.8, \ \pi_l = 0.2$

$$\text{Let } Y = \begin{cases} 1, & \text{if company gave dividends} \\ 0, & \text{otherwise.} \end{cases}$$

$$P(Y=1 \mid X=14) = \frac{\pi_k \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu_k)^2}{2\sigma^2}}}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu_l)^2}{2\sigma^2}}}$$

$$= \frac{0.8\left(\frac{1}{\sqrt{2\pi}(6)} \cdot e^{-\frac{(14-10)^2}{2(36)}}\right)}{0.8\left(\frac{1}{\sqrt{2\pi}(6)} \cdot e^{-\frac{(14-10)^2}{2(36)}}\right) + 0.2\left(\frac{1}{\sqrt{2\pi}(6)} \cdot e^{-\frac{(14-0)^2}{2(36)}}\right)}$$

$$= 0.97989 \ \#$$

Q7 a) • Split X_1 at 4

$(x_1 < 4)$ $\quad Gini_{left} = 1 - (1^2 + 0^2) = 0$

$(x_1 > 4)$ $\quad Gini_{right} = 1 - \left(\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2\right) = 1 - \left(\frac{1}{4} + \frac{1}{4}\right) = \frac{1}{2}$

$$\text{Weighted Gini} = \frac{5}{11}(0) + \frac{6}{11}\left(\frac{1}{2}\right)$$

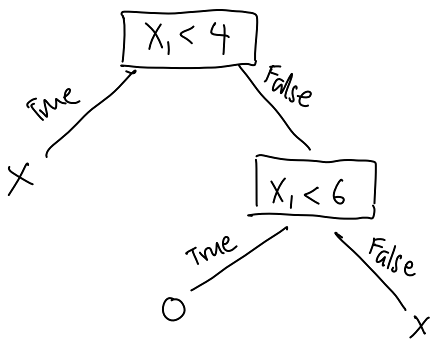
$$= \frac{3}{11}$$

• Split X_1 at 6

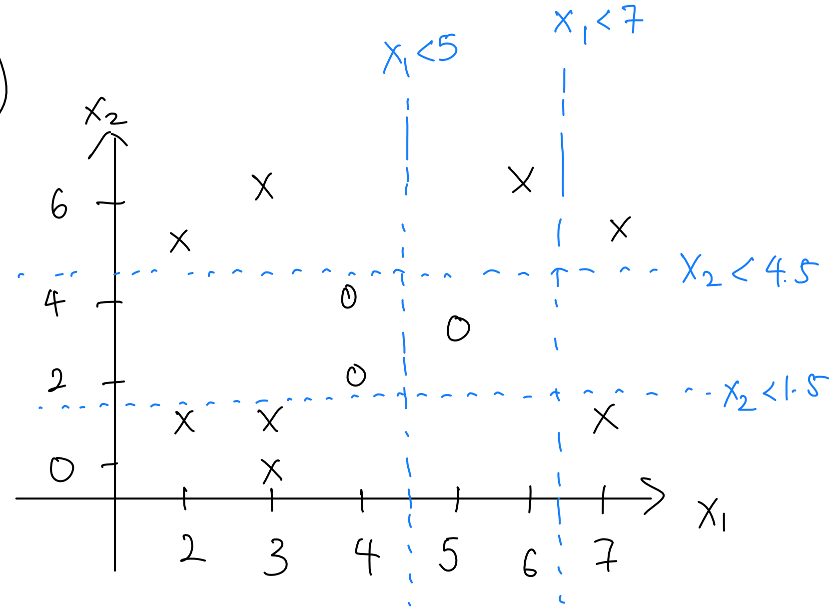
$(x_1 < 5)$ $\quad Gini_{left} = 1 - (0^2 + 1^2) = 0$

$(x_1 > 5)$ $\quad Gini_{right} = 1 - (1^2 + 0^2) = 0$

weighted Gini $= 0$



Q7b)



Q7c) 1) • The data does not look to be linearly separable based on the picture. Therefore, to achieve 0 training error, we could attempt to use Quadratic Discriminant Analysis to fit parabolas that might separate the 2 classes with no training error.

• There are no hyperparameters for quadratic discriminant analysis.

2) Logistic regression with polynomial features for non-linear boundary. Hyperparameter to tune will be learning rate used in gradient descent.

3) K-nearest neighbour with hyperparameter $K=1$. For no training error, each training point can be classified correctly as it will always be its own nearest neighbour.

STAT3612: Statistical Machine Learning

Assignment 2: Classification

DUE: Nov 10, 2024, Sunday, 11:59 PM

In [1]:
```
! pip install numpy pandas matplotlib scikit-learn keras
```

```
Requirement already satisfied: numpy in c:\users\kevin\appdata\local\programs\pyth
on\python310\lib\site-packages (2.0.2)
Requirement already satisfied: pandas in c:\users\kevin\appdata\local\programs\pyt
hon\python310\lib\site-packages (2.2.2)
Requirement already satisfied: matplotlib in c:\users\kevin\appdata\local\programs
\python\python310\lib\site-packages (3.5.2)
Requirement already satisfied: scikit-learn in c:\users\kevin\appdata\local\progra
ms\python\python310\lib\site-packages (1.3.0)
Requirement already satisfied: keras in c:\users\kevin\appdata\local\programs\pyth
on\python310\lib\site-packages (3.8.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\kevin\appdata\lo
cal\programs\python\python310\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\kevin\appdata\local\progra
ms\python\python310\lib\site-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\kevin\appdata\local\prog
rams\python\python310\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: cycler>=0.10 in c:\users\kevin\appdata\local\progra
ms\python\python310\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\kevin\appdata\local\p
rograms\python\python310\lib\site-packages (from matplotlib) (4.34.4)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\kevin\appdata\local\p
rograms\python\python310\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\kevin\appdata\local\pro
grams\python\python310\lib\site-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\kevin\appdata\local\progr
ams\python\python310\lib\site-packages (from matplotlib) (9.2.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\kevin\appdata\local\pr
ograms\python\python310\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: scipy>=1.5.0 in c:\users\kevin\appdata\local\progra
ms\python\python310\lib\site-packages (from scikit-learn) (1.9.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\kevin\appdata\local\progr
ams\python\python310\lib\site-packages (from scikit-learn) (1.3.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\kevin\appdata\loca
l\programs\python\python310\lib\site-packages (from scikit-learn) (3.2.0)
Requirement already satisfied: absl-py in c:\users\kevin\appdata\local\programs\py
thon\python310\lib\site-packages (from keras) (2.1.0)
Requirement already satisfied: rich in c:\users\kevin\appdata\local\programs\pytho
n\python310\lib\site-packages (from keras) (13.7.1)
Requirement already satisfied: namex in c:\users\kevin\appdata\local\programs\pyth
on\python310\lib\site-packages (from keras) (0.0.8)
Requirement already satisfied: h5py in c:\users\kevin\appdata\local\programs\pytho
n\python310\lib\site-packages (from keras) (3.11.0)
Requirement already satisfied: optree in c:\users\kevin\appdata\local\programs\pyt
hon\python310\lib\site-packages (from keras) (0.14.1)
Requirement already satisfied: ml-dtypes in c:\users\kevin\appdata\local\programs
\python\python310\lib\site-packages (from keras) (0.4.1)
Requirement already satisfied: six>=1.5 in c:\users\kevin\appdata\local\programs\p
ython\python310\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Collecting numpy
  Downloading numpy-1.24.4-cp310-cp310-win_amd64.whl.metadata (5.6 kB)
Requirement already satisfied: typing-extensions>=4.5.0 in c:\users\kevin\appdata
\local\programs\python\python310\lib\site-packages (from optree->keras) (4.10.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in c:\users\kevin\appdata\loc
al\programs\python\python310\lib\site-packages (from rich->keras) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\kevin\appdata\r
oaming\python\python310\site-packages (from rich->keras) (2.15.1)
Requirement already satisfied: mdurl~=0.1 in c:\users\kevin\appdata\local\programs
\python\python310\lib\site-packages (from markdown-it-py>=2.2.0->rich->keras) (0.
1.2)
Downloading numpy-1.24.4-cp310-cp310-win_amd64.whl (14.8 MB)
   ---------------------------------------- 0.0/14.8 MB ? eta -:--:--
   ---- ----------------------------------- 1.8/14.8 MB 8.4 MB/s eta 0:00:02
   --------------------- ------------------ 8.1/14.8 MB 20.2 MB/s eta 0:00:01
   ------------------------------------- --- 13.6/14.8 MB 22.0 MB/s eta 0:00:01
```

```
   ---------------------------------------- 14.8/14.8 MB 21.7 MB/s eta 0:00:00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
Successfully installed numpy-1.24.4
```

```
ERROR: pip's dependency resolver does not currently take into account all the pack
ages that are installed. This behaviour is the source of the following dependency
conflicts.
tensorflow-intel 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy 1.24.4 w
hich is incompatible.
```

In [10]:

```python
import numpy as np
from sklearn.datasets import fetch_openml, load_digits
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score, precision_score, f1_score, recall_score
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.utils import shuffle
```

In [2]:

```python
# Q7 (a)

# ------------------
# Write your code here
# Load the MNIST dataset
X, y = fetch_openml('mnist_784', version=1, return_X_y=True, as_frame=False)
y = y.astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=60000, test_si
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_train[0])

# Fit LDA model
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

# Make prediction and access accuracy
y_pred = lda.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Testing Data Accuracy: {accuracy * 100:.2f}%")

# ------------------
```

```
c:\Users\kevin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\d
atasets\_openml.py:1002: FutureWarning: The default value of `parser` will change
from `'liac-arff'` to `'auto'` in 1.4. You can set `parser='auto'` to silence this
warning. Therefore, an `ImportError` will be raised from 1.4 if the dataset is den
se and pandas is not installed. Note that the pandas parser may return different d
ata types. See the Notes Section in fetch_openml's API doc for details.
  warn(
(60000, 784)
(10000, 784)
(60000,)
3
Testing Data Accuracy: 86.21%
```

In [ ]:
```
# Q7 (b)

# ------------------
# Write your code here
# Apply filter
y_7b = y[(y==0) | (y==1)]
X_7b = X[(y==0) | (y==1)]

X_train_7b, X_test_7b, y_train_7b, y_test_7b = train_test_split(X_7b, y_7b, test_si

# Initialize LDA
lda = LinearDiscriminantAnalysis()
lda.fit(X_train_7b, y_train_7b)

y_pred_7b = lda.predict(X_test_7b)

# Compute metrics and display results
precision = precision_score(y_test_7b, y_pred_7b)
recall = recall_score(y_test_7b, y_pred_7b)
f1 = f1_score(y_test_7b, y_pred_7b)

print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
# ------------------
```

```
Precision: 0.9950
Recall: 0.9944
F1-Score: 0.9947
Accuracy: 0.9942
```

In [14]:
```
from keras.datasets import cifar10

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
print('X_train shape:', X_train.shape)
print('X_test shape:', X_test.shape)
print('y_train shape:', y_train.shape)
print('y_test shape:', y_test.shape)

X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

# Normalise pixel values to [0,1]
X_train = X_train / 255
X_test = X_test / 255
```

```
X_train shape: (50000, 32, 32, 3)
X_test shape: (10000, 32, 32, 3)
y_train shape: (50000, 1)
y_test shape: (10000, 1)
```

Consider flatten the images to 1D vectors and make the shape of the X be (Batch, dim).

In [ ]:
```
# Q8 (a)

# ------------------
# Write your code here
# FLatten images to 1D vector
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

# Normalise pixel values to [0,1]
X_train = X_train / 255
X_test = X_test / 255
```

```
print(X_train)

# Standardize the data since L1 penalty is to be used.
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define a range of regularization strengths (C values)
C_values = np.logspace(-2, 2, 5)  # Logarithmic range from 1e-4 to 1e4
print(C_values)
accuracies = []

# Train and evaluate the model for each C value
for C in C_values:
    print(f"Training with C={C:.4f}...")
    model = LogisticRegression(
        multi_class="multinomial",
        solver="saga",
        penalty="l1",
        C=C,  # Inverse of regularization strength in sklearn
        max_iter=100,
        random_state=42,
        warm_start=True,
    )
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
    print(f"Accuracy: {accuracy:.4f}")

# ------------------
```

```
X_train shape: (50000, 3072)
X_test shape: (10000, 3072)
[[0.23137255 0.24313725 0.24705882 ... 0.48235294 0.36078431 0.28235294]
 [0.60392157 0.69411765 0.73333333 ... 0.56078431 0.52156863 0.56470588]
 [1.         1.         1.         ... 0.31372549 0.3372549  0.32941176]
 ...
 [0.1372549  0.69803922 0.92156863 ... 0.04705882 0.12156863 0.19607843]
 [0.74117647 0.82745098 0.94117647 ... 0.76470588 0.74509804 0.67058824]
 [0.89803922 0.89803922 0.9372549  ... 0.63921569 0.63921569 0.63137255]]
[1.e-02 1.e-01 1.e+00 1.e+01 1.e+02]
Training with C=0.0100...
```

```
c:\Users\kevin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\u
tils\validation.py:1184: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), for exampl
e using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\kevin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\l
inear_model\_sag.py:350: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
  warnings.warn(
```

```
Accuracy: 0.4025
Training with C=0.1000...
```

```
c:\Users\kevin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\u
tils\validation.py:1184: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), for exampl
e using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\kevin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\l
inear_model\_sag.py:350: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
  warnings.warn(
```

```
Accuracy: 0.4124
Training with C=1.0000...
```

```
c:\Users\kevin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\u
tils\validation.py:1184: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), for exampl
e using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\kevin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\l
inear_model\_sag.py:350: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
  warnings.warn(
c:\Users\kevin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\u
tils\validation.py:1184: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), for exampl
e using ravel().
  y = column_or_1d(y, warn=True)
```

```
Accuracy: 0.3923
Training with C=10.0000...
```

```
c:\Users\kevin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\l
inear_model\_sag.py:350: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
  warnings.warn(
```

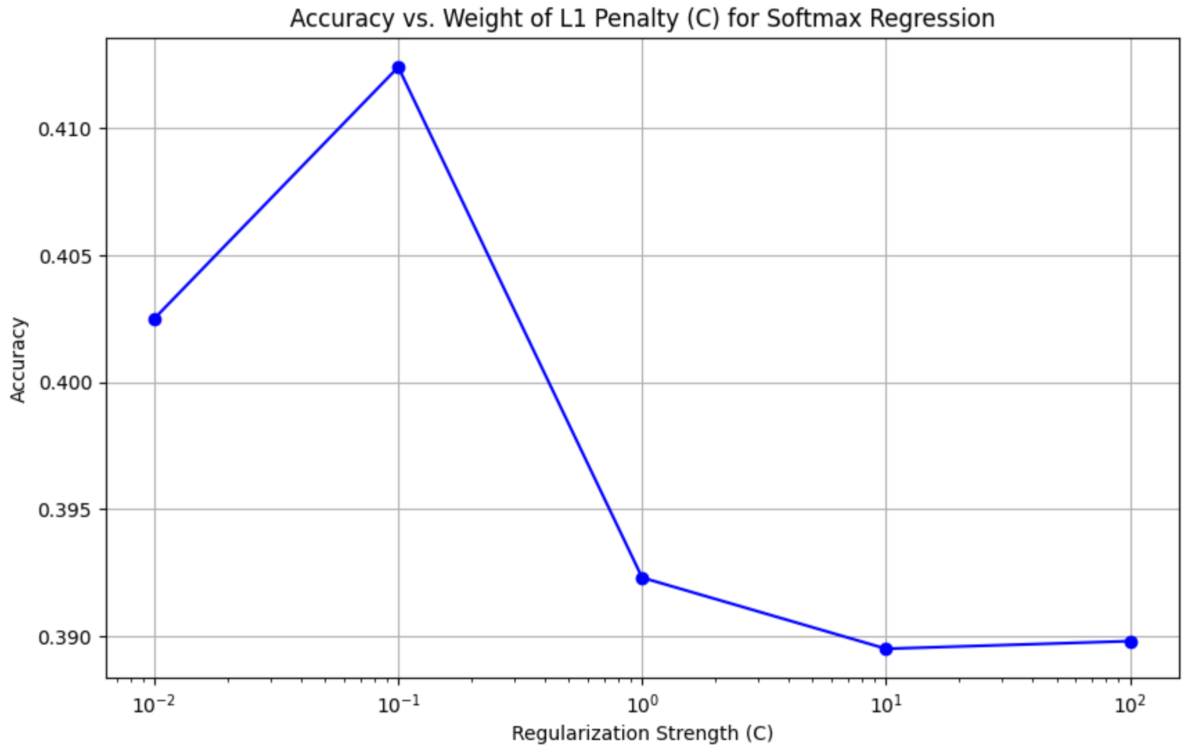
```
Accuracy: 0.3895
Training with C=100.0000...
```

```
c:\Users\kevin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\u
tils\validation.py:1184: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_samples, ), for exampl
e using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\kevin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\l
inear_model\_sag.py:350: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
  warnings.warn(
```

```
Accuracy: 0.3898
```

In [4]:

```
# Plot the accuracy vs. weight of the L1 penalty
plt.figure(figsize=(10, 6))
plt.plot(C_values, accuracies, marker='o', linestyle='-', color='b')
plt.xscale('log')  # Use logarithmic scale for C values
plt.xlabel("Regularization Strength (C)")
plt.ylabel("Accuracy")
plt.title("Accuracy vs. Weight of L1 Penalty (C) for Softmax Regression")
plt.grid(True)
plt.show()
```



The L1 penalty term is defined as:

$$L_{\mathrm{L1}} = \lambda \|\mathbf{W}\|_1 = \lambda \sum_{i,j} |w_{ij}|$$

The gradient of the L1 penalty with respect to the weights is:

$$\nabla_{\mathbf{W}} \mathcal{L}_{\mathrm{L1}} = \lambda \cdot \mathrm{sign}(\mathbf{W})$$

, where $\mathrm{sign}(\mathbf{W})$ is the element-wise sign of the weight matrix.

In [54]:

```
# Q8 (b)
np.random.seed(0)

def softmax(x):
    return np.exp(x) / np.sum(np.exp(x), axis = 1, keepdims=True)

def sgd_step(w, X_train_mini, y_train_mini, lr, minibatch_size, lambda_l1):
    """
    Perform a single step of Stochastic Gradient Descent (SGD) to update the weight

    Parameters:
    w: ndarray
        Weight matrix of the model. This is the parameter we want to update.
    X_train_mini: ndarray
        A mini-batch of training data. Each row corresponds to a training example,
    y_train_mini: ndarray
        One-hot encoded labels for the mini-batch. Each row is the label for the cc
    lr: float
        Learning rate for the gradient descent step.
    minibatch_size: int
        The size of the mini-batch used in the update step.
    lambda_l1: float
        Regularization strength for the L1 penalty.

    Returns:
    w: ndarray
        Updated weight matrix after the SGD step.
    """
```

```
    # Compute the gradient of the cross-entropy loss with respect to the weights
    dw = (softmax((w @ X_train_mini.T).T).T - y_train_mini.T) @ X_train_mini / mini

    # Add the gradient of the L1 regularization term
    dw += lambda_l1 * np.sign(w)

    # Update the weights by subtracting the learning rate times the gradient
    w = w - lr * dw

    # Return the updated weight matrix
    return w

def sgd(w, X_train, y_train, lr, lambda_l1, minibatch_size=64):
    """
    Perform Stochastic Gradient Descent (SGD) over the entire dataset with L1 regul

    Parameters:
    w: ndarray
        Weight matrix of the model. This is the parameter we want to update.
    X_train: ndarray
        Training data. Each row corresponds to a training example, and each column
    y_train: ndarray
        One-hot encoded labels for the training data. Each row is the label for the
    lr: float
        Learning rate for the gradient descent step.
    minibatch_size: int, optional
        The size of the mini-batch used in each update step. Default is 64.
    lambda_l1: float, optional
        Regularization strength for the L1 penalty. Default is 0.01.

    Returns:
    w: ndarray
        Updated weight matrix after processing the entire training data using SGD.
    """
    # Shuffle the dataset to ensure that the mini-batches are selected randomly in
    X_train, y_train = shuffle(X_train, y_train)

    # Iterate over the dataset in mini-batches
    for i in range(0, X_train.shape[0], minibatch_size):
        # Extract the current mini-batch from the training data
        X_train_mini = X_train[i:i + minibatch_size]
        # Extract the corresponding mini-batch of labels
        y_train_mini = y_train[i:i + minibatch_size]

        # Perform a single SGD step to update the weights using the mini-batch
        w = sgd_step(w, X_train_mini, y_train_mini, lr, minibatch_size, lambda_l1)

    # Return the updated weights after all mini-batches have been processed
    return w

def train_with_SGD(model, X_train, y_train, lr, epoch_num, lambda_l1):
    """
    Train the model using Stochastic Gradient Descent (SGD) with L1 regularization.

    Parameters:
    model: dict
        Dictionary containing the weight matrix.
    X_train: ndarray
        Training data. Each row corresponds to a training example, and each column
    y_train: ndarray
        One-hot encoded labels for the training data. Each row is the label for the
    lr: float
        Learning rate for the gradient descent step.
    epoch_num: int
```

```
        Number of epochs to train the model.
    lambda_l1: float
        Regularization strength for the L1 penalty.

    Returns:
    model: dict
        Updated model after training.
    """
    # Extract the weight matrix from the model
    w = model['w']

    # Train the model for the specified number of epochs
    for epoch in range(epoch_num):
        w = sgd(w, X_train, y_train, lr, lambda_l1=lambda_l1)
        if (epoch + 1) % 10 == 0:
            print(f"Epoch {epoch + 1}/{epoch_num}")

    # Update the model with the trained weights
    model['w'] = w
    return model


def test(model, X_test, y_test):
    """
    Evaluate the model on the test set.

    Parameters:
    model: dict
        Dictionary containing the model parameters (e.g., weight matrix).
    X_test: ndarray
        Test data. Each row corresponds to a test example, and each column correspo
    y_test: ndarray
        One-hot encoded labels for the test data. Each row is the label for the cor

    Returns:
    accuracy: float
        Accuracy of the model on the test set.
    """
    # Extract the weight matrix from the model
    w = model['w']

    # Compute the predicted probabilities
    scores = np.dot(X_test, w.T)
    y_pred = np.argmax(scores, axis=1)
    y_true = np.argmax(y_test, axis=1)

    # Compute the accuracy
    accuracy = np.mean(y_pred == y_true)
    return accuracy


# ------------------
# Write your code here
# One-hot encode labels
encoder = OneHotEncoder(sparse=False)
y_train_onehot = encoder.fit_transform(y_train.reshape(-1, 1))
y_test_onehot = encoder.fit_transform(y_test.reshape(-1, 1))


# Initialize the model
n_features = X_train.shape[1]
n_classes = y_train.shape[1]
model = {'w': np.random.randn(n_classes, n_features) * 0.01}
```

```
# Train the model
model = train_with_SGD(model, X_train, y_train_onehot, lr=0.002, epoch_num=100, lam

# Test the model
accuracy = test(model, X_test, y_test_onehot)
print(f"Test Accuracy: {accuracy:.4f}")

# ------------------
```

```
c:\Users\kevin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\p
reprocessing\_encoders.py:972: FutureWarning: `sparse` was renamed to `sparse_outp
ut` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless y
ou leave `sparse` to its default value.
  warnings.warn(
c:\Users\kevin\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\p
reprocessing\_encoders.py:972: FutureWarning: `sparse` was renamed to `sparse_outp
ut` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless y
ou leave `sparse` to its default value.
  warnings.warn(
Epoch 10/100
Epoch 20/100
Epoch 30/100
Epoch 40/100
Epoch 50/100
Epoch 60/100
Epoch 70/100
Epoch 80/100
Epoch 90/100
Epoch 100/100
Test Accuracy: 0.4031
```

In [55]:

```
# Find the class with the highest accuracy
class_accuracies = []
for k in range(n_classes):
    mask = np.argmax(y_test, axis=1) == k
    class_accuracies.append(np.mean(y_pred[mask] == k))

best_class = np.argmax(class_accuracies)
print(f"Class with highest accuracy: {best_class}")

# Visualize an image from the best class
best_class_indices = np.where(np.argmax(y_test, axis=1) == best_class)[0]
sample_index = best_class_indices[0]
sample_image = X_test[sample_index].reshape(32, 32, 3)

plt.imshow(sample_image)
plt.title(f"Class {best_class}")
plt.axis("off")
plt.show()
```

```
Class with highest accuracy: 0
```

Class 0