HW3

March 28, 2024

1 HW3

1.1 Problem15

[ ]:
```python
import gzip
import numpy as np

def load_images(path_to_images):
    with gzip.open(path_to_images, 'rb') as file:
        # The first 16 bytes contain the magic number, the number of images,
        # the number of rows, and the number of columns respectively.
        # We skip this information.
        file.read(16)
        # The rest are the image pixels
        images = np.frombuffer(file.read(), dtype=np.uint8)
    # The images are 28 pixels in each dimension.
    return images.reshape(-1, 784)

def load_labels(path_to_labels):
    with gzip.open(path_to_labels, 'rb') as file:
        # The first 8 bytes contain the magic number and the number of labels.
        # We skip this information.
        file.read(8)
        # The rest are the labels
        labels = np.frombuffer(file.read(), dtype=np.uint8)
    return labels

# Use the paths to your downloaded MNIST gzip files.
images_path = 't10k-images-idx3-ubyte.gz'
labels_path = 't10k-labels-idx1-ubyte.gz'

# Load the images and labels.
test_images = load_images(images_path)
test_labels = load_labels(labels_path)
```

[ ]:
```python
# Filter for images with a label of 3
indices_of_3s = np.where(test_labels == 3)[0]
```

```
train_images_3s = test_images[indices_of_3s[:400]]  # First 400 images for␣
 ↪training
test_images_3s = test_images[indices_of_3s[400:800]]  # Next 400 images for␣
 ↪testing

# Filter for images with a label of 5
indices_of_5s = np.where(test_labels == 5)[0]
train_images_5s = test_images[indices_of_5s[:400]]  # First 400 images for␣
 ↪training
test_images_5s = test_images[indices_of_5s[400:800]]  # Next 400 images for␣
 ↪testing

# Create label vectors for the training and testing set
train_labels_3s5s = np.array([3]*400 + [5]*400)  # Labels for the training set
test_labels_3s5s = np.array([3]*400 + [5]*400)  # Labels for the testing set

# Combine the training and testing images
train_images = np.concatenate((train_images_3s, train_images_5s), axis=0)
test_images = np.concatenate((test_images_3s, test_images_5s), axis=0)
```

1.1.1 (a)

[ ]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Assuming 'train_images' and 'train_labels_3s5s' are your training data and␣
 ↪labels
# and 'test_images' and 'test_labels_3s5s' are your testing data and labels,
# and they have been properly loaded and preprocessed.

# Initialize the logistic regression model
logistic_model = LogisticRegression(max_iter=500)

# Fit the model to the training data
logistic_model.fit(train_images, train_labels_3s5s)

# Predict labels for both training and testing sets
train_predictions = logistic_model.predict(train_images)
test_predictions = logistic_model.predict(test_images)

# Calculate accuracy for both training and testing sets
train_accuracy = accuracy_score(train_labels_3s5s, train_predictions)
test_accuracy = accuracy_score(test_labels_3s5s, test_predictions)

# Calculate misclassification rates (error rates)
train_error_rate = 1 - train_accuracy
```

```
test_error_rate = 1 - test_accuracy

train_error_rate, test_error_rate
```

[ ]: (0.0, 0.08999999999999997)

1.1.2 (b)

```
[ ]: logistic_model_lasso = LogisticRegression(penalty='l1', C=10, solver='saga',␣
      ↪max_iter=5000)

     # Fit the model to the training data
     logistic_model_lasso.fit(train_images, train_labels_3s5s)

     # Predict labels for both training and testing sets
     train_predictions_lasso = logistic_model_lasso.predict(train_images)
     test_predictions_lasso = logistic_model_lasso.predict(test_images)

     # Calculate accuracy for both training and testing sets
     train_accuracy_lasso = accuracy_score(train_labels_3s5s,␣
      ↪train_predictions_lasso)
     test_accuracy_lasso = accuracy_score(test_labels_3s5s, test_predictions_lasso)

     # Calculate misclassification rates (error rates)
     train_error_rate_lasso = 1 - train_accuracy_lasso
     test_error_rate_lasso = 1 - test_accuracy_lasso

     train_error_rate_lasso, test_error_rate_lasso
```

[ ]: (0.0, 0.08625000000000005)

1.1.3 (c)

```
[ ]: # Generating C values (inverse of lambda)
     C_values = np.logspace(-6, 3, 10)

     best_score = 0
     best_C = None

     for C in C_values:
         # Initialize the logistic regression model with L1 penalty
         model = LogisticRegression(penalty='l1', C=C, solver='saga', max_iter=5000,␣
      ↪random_state=0)

         # Fit the model to the training data
         model.fit(train_images, train_labels_3s5s)
```

```
    # Evaluate the model on the test set
    test_score = accuracy_score(test_labels_3s5s, model.predict(test_images))

    # If the score for this C is better than the best score we've seen, update
 ↪the best score and best C
    if test_score > best_score:
        best_score = test_score
        best_C = C

# best_lambda is the inverse of best_C, since lambda = 1/C
best_lambda = 1 / best_C if best_C else None

best_lambda, 1 - best_score
```

[ ]: (10.0, 0.08374999999999999)

1.1.4 (d)

[ ]:
```
from sklearn.svm import SVC

# Create a support vector classifier with C=1.0
# The SVC class automatically uses the 'scale' kernel coefficient which is
 ↪equivalent to scale=FALSE
svm_classifier = SVC(C=1.0, kernel='linear', random_state=0)

# Fit the classifier to the training data
svm_classifier.fit(train_images, train_labels_3s5s)

# Predict labels for both the training set and the test set
train_predictions_svm = svm_classifier.predict(train_images)
test_predictions_svm = svm_classifier.predict(test_images)

# Calculate accuracy for both the training and the test set
train_accuracy_svm = accuracy_score(train_labels_3s5s, train_predictions_svm)
test_accuracy_svm = accuracy_score(test_labels_3s5s, test_predictions_svm)

# Calculate misclassification rates
train_error_rate_svm = 1 - train_accuracy_svm
test_error_rate_svm = 1 - test_accuracy_svm

train_error_rate_svm, test_error_rate_svm
```

[ ]: (0.0, 0.08499999999999996)