m3-hw3

February 9, 2024

1 Problem 3 (20 Points)

1.1 Problem description

So far, we have worked with ~2 dimensional problems with 2-3 classes. Most often in ML, there are many more explanatory variables and classes than this. In this problem, you'll be training logistic regression models on a database of grayscale images of hand-drawn digits, using SciKit-Learn. Now there are 400 (20x20) input features and 10 classes (digits 0-9).

As usual, you can use any code from previous problems.

1.2 Summary of deliverables

- OvR model accuracy on training data
- OvR model accuracy on testing data
- Multinomial model accuracy on training data
- Multinomial model accuracy on testing data

1.2.1 Imports and Utility Functions:

[13]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

def visualize(xdata, index, title=""):
    image = xdata[index,:].reshape(20,20).T
    plt.figure()
    plt.imshow(image,cmap = "binary")
    plt.axis("off")
    plt.title(title)
    plt.show()
```

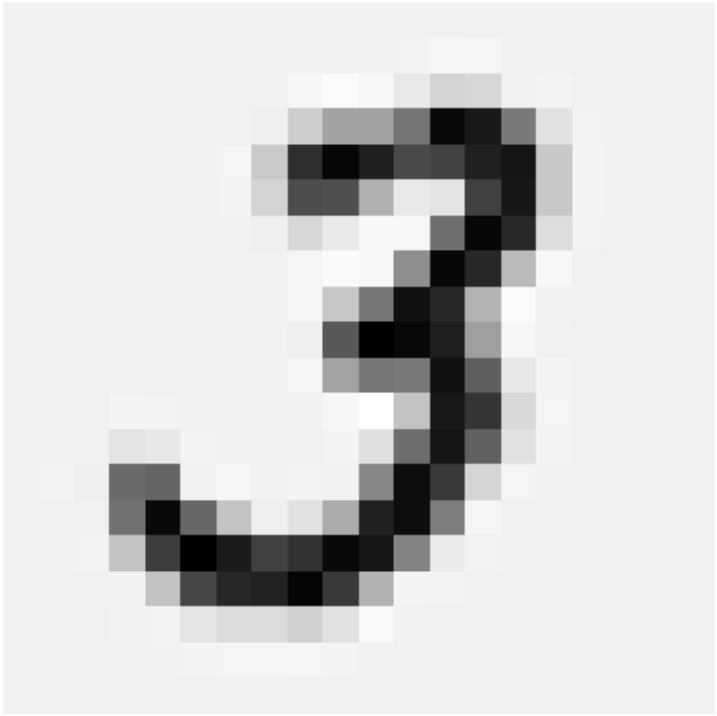
1.3 Load data

The following cell loads in training and testing data into the following variables: - `x_train`: 4000x400 array of input features, used for training - `y_train`: Array of ground-truth classes for each point in `x_train` - `x_test`: 1000x400 array of input features, used for testing - `y_test`: Array of ground-truth classes for each point in `x_test`

You can visualize a digit with the `visualize(x_data, index)` function.

[14]:
```
x_train = np.load("data/w3-hw3-train_x.npy")
y_train = np.load("data/w3-hw3-train_y.npy")
x_test = np.load("data/w3-hw3-test_x.npy")
y_test = np.load("data/w3-hw3-test_y.npy")

visualize(x_train,1234)
```



1.4 Logistic Regression Models

Use sklearn's `LogisticRegression` to fit a multinomial logistic regression model on the training data. You may need to increase the `max_iter` argument for the model to converge.

Train 2 models: one using the One-vs-Rest method, and another that minimizes multinomial loss. You can do these by setting the `multi_class` argument to "ovr" and "multinomial", respectively.

More information: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.ht

[15]:
```
# YOUR CODE GOES HERE (sklearn models)

# One-vs-rest logistic regression
model_ovs = LogisticRegression(multi_class="ovr", max_iter=1000)
model_ovs.fit(x_train, y_train)

# Multinomial logistic regression
```

```
model_mlr = LogisticRegression(multi_class="multinomial", max_iter=1000)
model_mlr.fit(x_train, y_train)
```

[15]:
```
LogisticRegression(max_iter=1000, multi_class='multinomial')
```

1.5 Accuracy

Compute and print the accuracy of each model on the training and testing sets as a percent.

[16]:
```
# YOUR CODE GOES HERE (print the 4 requested accuracy values)

# calculate the accuracy of the models
train_score_ovs = model_ovs.score(x_train, y_train) * 100
test_score_ovs = model_ovs.score(x_test, y_test) * 100
train_score_mlr = model_mlr.score(x_train, y_train) * 100
test_score_mlr = model_mlr.score(x_test, y_test) * 100

# print the 4 requested accuracy values
print(f"[One-vs-Rest Model] Training accuracy: {train_score_ovs:.3f}%, Testing␣
 ↪accuracy: {test_score_ovs:.3f}%")
print(f"[Multinomial Loss Model] Training accuracy: {train_score_mlr:.3f}%,␣
 ↪Testing accuracy: {test_score_mlr:.3f}%")
```

```
[One-vs-Rest Model] Training accuracy: 94.675%, Testing accuracy: 90.800%
[Multinomial Loss Model] Training accuracy: 96.450%, Testing accuracy: 91.400%
```