One-vs-All Logistic Regression Ensemble

May 20, 2024

[ ]:
```
import numpy as np
from scipy.stats import multivariate_normal
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

[ ]:
```
# Data Preprocessing
TRAIN = './mnist_train.csv'
TEST = './mnist_test.csv'

# Function to load data and split into features and labels
def load_data_and_split(filepath):
    data = np.genfromtxt(filepath, delimiter=',', skip_header=1, dtype='int')
    X = data[:, 1:]  # All columns except the first one
    y = data[:, 0]  # Only the first column
    return X, y

# Load and split the training and testing data
X_train, y_train = load_data_and_split(TRAIN)
X_test, y_test = load_data_and_split(TEST)

# Normalize the training and testing data
X_train = X_train.astype(np.float64)
X_test = X_test.astype(np.float64)

X_train /= 255
X_test /= 255
```

[ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import warnings

# Settings the warnings to be ignored
warnings.filterwarnings('ignore')

classes = np.unique(y_train)

# Initialize the models
logistic_models = []
```

```python
# Train one-vs-all models
for idx, c in enumerate(classes):
    # Prepare the target vector for the current class
    y_train_binary = (y_train == c).astype(int)
    # Train the model
    model = LogisticRegression(multi_class='ovr')
    model.fit(X_train, y_train_binary)

    # Save the model
    logistic_models.append(model)

# Evaluate the models
ensemble_predictions = np.zeros((X_test.shape[0], 10))
for class_idx, model in enumerate(logistic_models):
    # Predict the probability for the positive class
    ensemble_predictions[:, class_idx] = model.predict_proba(X_test)[:, 1]

# Choose the class with the highest probability
y_pred_ensemble = np.argmax(ensemble_predictions, axis=1)

# Compute individual accuracies
for class_idx, model in enumerate(logistic_models):
    y_test_binary = (y_test == class_idx).astype(int)
    y_pred_binary = model.predict(X_test)
    accuracy = np.mean(y_test_binary == y_pred_binary)
    print(f"Accuracy for class {class_idx}: {accuracy * 100:.3f}%")

# Compute ensemble accuracy
ensemble_accuracy = np.mean(y_test== y_pred_ensemble)
print(f"Ensemble accuracy: {ensemble_accuracy * 100:.3f}%")
print(f"One-Zero Error: {(1-ensemble_accuracy)*100 :.3f}%")
```

```
Accuracy for class 0: 99.170%
Accuracy for class 1: 99.360%
Accuracy for class 2: 98.060%
Accuracy for class 3: 97.760%
Accuracy for class 4: 98.350%
Accuracy for class 5: 97.810%
Accuracy for class 6: 98.600%
Accuracy for class 7: 98.370%
Accuracy for class 8: 96.110%
Accuracy for class 9: 96.750%
Ensemble accuracy: 92.120%
One-Zero Error: 7.880%
```