CSE 4/574 Project 3

Sargur N. Srihari
University at Buffalo, State University of New York
USA

Project 3 Components

1. Classification Task
2. Classifiers
   - Multiclass Logistic Regression (Softmax Regression)
   - Library Functions
     - Neural Network, SVM, Random Forest
   - Classifier Combination
3. Keras Code

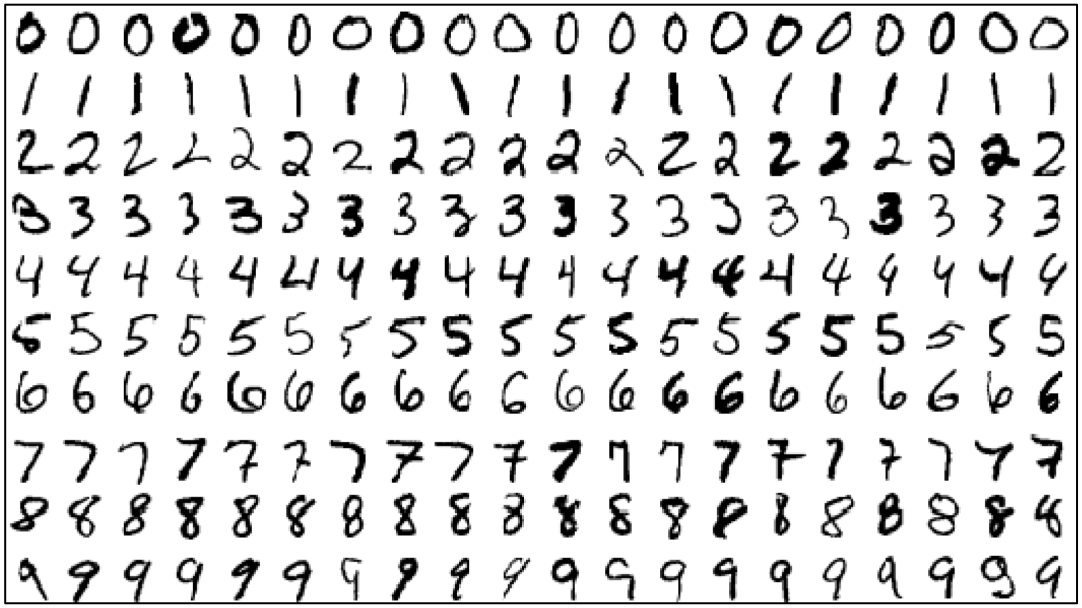
The Task

- This project is to implement machine learning methods for the task of classification.
  - Implement an ensemble of four classifiers for a given task.
  - Then the results of the individual classifiers are combined to make a final decision
- The classification task:
  - Recognizing 28×28 grayscale handwritten digit images
    - Identify it as a digit among $0,\ 1,\ 2,\ \ldots\ ,\ 9$

Data Sets (MNIST)

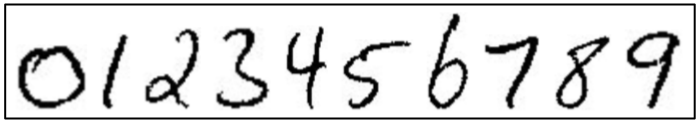
Training data: MNIST

28 x 28 gray-scale images
60,000 training, 10,000 testing



Testing data: MNIST and USPS

2000 samples per digit, 100ppi cropped
Resize to 28 x 28



Classifier Ensemble

- Logistic regression
  - which you implement yourself using backpropagtion and tune hyperparameters.
- Three Library Functions
  - multilayer perceptron neural network, train it on the MNIST digit images and tune hyperparameters.
  - Random Forest package, train it on the MNIST digit images and tune hyperparameters.
  - SVM package, train it on the MNIST digit images and tune hyperparameters.

Logistic Regression with Softmax

- For 10 classes, we work with soft-max function
  Input $\boldsymbol{\phi}=[\phi_1,.., \phi_M]^T$
- Activations $a_k = \boldsymbol{w}_k^T \boldsymbol{\phi} + b_k, \quad k=1,..,10$
  $\boldsymbol{w_k} = [w_{k,1},.., w_{k,10}]^T$ and $\boldsymbol{a}=\{a_1,..a_{10}\}$
- We learn a set of 10 weight vectors $\{\boldsymbol{w}_1,.., \boldsymbol{w}_{10}\}$ and biases $\boldsymbol{b}$
- Arranging weight vectors as a matrix W

$$\boxed{W = \begin{bmatrix} \boldsymbol{w}_1 \\ \cdot \\ \cdot \\ \boldsymbol{w}_K \end{bmatrix} = \begin{bmatrix} w_{11} & & w_{1M} \\ \cdot & & \\ \cdot & & \\ w_{K1} & & w_{KM} \end{bmatrix}}$$

$$\boxed{\boldsymbol{a} = W^T \boldsymbol{\phi} + \boldsymbol{b}}$$

$$\boxed{\begin{aligned} \boldsymbol{y} &= \mathrm{softmax}(\boldsymbol{a}) \\ y_i &= \frac{\exp(a_i)}{\sum_{j=1}^{3} \exp(a_j)} \end{aligned}}$$

$$\boxed{p(C_k \mid \phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}}$$

One-hot vector representation

- Classes $C_1,..C_{10}$ represented by 1-of-10 scheme
  - One-hot vector:
    - class C_k is a 10-dim vector or $[t_1,..,t_{10}]^{\mathrm{T}}$, $t_i\, \varepsilon\, \{0,1\}$
      - With $K=10$, class C_3 is $(0,0,1,0,0,0,0,0,0,0)^T$
    - The class probabilities obey

$$\boxed{\sum_{k=1}^{10} p(C_k) = \sum_{k=1}^{10} t_k = 1}$$

  - If $p(t_k=1) = \mu_k$ then

$$p(C_k) = \prod_{k=1}^{10} \mu_k^{t_k} \text{ where } \boldsymbol{\mu} = (\mu_1,..,\mu_{10})^T$$

e.g., probability of C_3 is

$$p([0,0,1,0,0,0,0,0,0,0]) = \mu_3$$

Target Matrix, T

- Classes have values $1, .., 10$
- Each represented as a 10-dimensional binary vector
- We have N labeled samples
  - So instead of target vector $\boldsymbol{t}$ we have a target matrix T

$$\begin{array}{ccc} & \text{Classes} & \\ T = & \begin{bmatrix} t_{1,1} & \cdot & \cdot & t_{1,10} \\ \cdot & & & \\ \cdot & & & \\ t_{N,1} & & & t_{N,10} \end{bmatrix} & \text{Samples} \end{array}$$

Note that t_{nk} corresponds to sample n and class k

Loss Function & Gradient

- Likelihood of observations $T = \begin{bmatrix} t_{11} & \cdot & \cdot & t_{1K} \\ \cdot & & & \\ \cdot & & & \\ t_{N1} & & & t_{NK} \end{bmatrix}$ is

$$\boxed{p(T \mid \boldsymbol{w}_1,..,\boldsymbol{w}_{10}) = \prod_{n=1}^{N} \prod_{k=1}^{10} p(C_k \mid \boldsymbol{\phi}_n)^{t_{n,k}} = \prod_{n=1}^{N} \prod_{k=1}^{10} y_{nk}^{t_{nk}}}$$

  - Where, for feature vector ϕ_n $\quad y_{nk} = y_k(\boldsymbol{\phi}_n) = \dfrac{\exp(\boldsymbol{w}_k^T \boldsymbol{\phi}_n)}{\sum_j \exp(\boldsymbol{w}_j^T \boldsymbol{\phi}_n)}$

- Loss Function: negative log-likelihood

$$\boxed{E(\boldsymbol{w}_1,...,\boldsymbol{w}_{10}) = -\ln p(T \mid \boldsymbol{w}_1,..,\boldsymbol{w}_{10}) = -\sum_{n=1}^{N} \sum_{k=1}^{10} t_{nk} \ln y_{nk}}$$

  – Known as cross-entropy error for multi-class

- Gradient of error function wrt parameter $\boldsymbol{w}_j$

$$\boxed{\nabla_{\mathrm{w}_j} E(\boldsymbol{w}_1,...,\boldsymbol{w}_{10}) = -\sum_{n=1}^{N} \underbrace{(y_{nj} - t_{nj})\phi_n}_{\text{Error x Feature Vector}}}$$

$\boxed{\text{using } \sum_k t_{nk} = 1}$

Error x Feature Vector

$$y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \qquad a_k = \boldsymbol{w}_k^{\mathrm{T}}\phi$$

$\dfrac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j)$ where I_{kj} are elements of the identity matrix

Gradient Descent for Multi-class Logistic Regression

- Cross-Entropy Error

$$T = \begin{bmatrix} t_{11} & . & . & t_{1K} \\ . & & & \\ . & & & \\ t_{N1} & & & t_{NK} \end{bmatrix}$$

$$\boxed{E(\mathrm{w}_1,\ldots,\mathrm{w}_{10}) = -\ln p(T \mid \mathrm{w}_1,\ldots,\mathrm{w}_{10}) = -\sum_{n=1}^{N}\sum_{k=1}^{10} t_{nk} \ln y_{nk}}$$

$$\boxed{y_{nk} = y_k(\phi(\mathrm{x}_n)) = \frac{\exp(\mathrm{w}_k^T \phi(\mathrm{x}_n))}{\sum_j \exp(\mathrm{w}_j^T \phi(\mathrm{x}_n))}}$$

- Gradient of E

$$\boxed{\nabla_{\mathrm{w}_j} E(\mathrm{w}_1,\ldots,\mathrm{w}_K) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\phi(\mathrm{x}_n)}$$

$$\boxed{\Phi = \begin{pmatrix} \phi_0(\mathrm{x}_1) & \phi_1(\mathrm{x}_1) & \ldots & \phi_{M-1}(\mathrm{x}_1) \\ \phi_0(\mathrm{x}_2) & & & \\ & & & \\ \phi_0(\mathrm{x}_N) & & & \phi_{M-1}(\mathrm{x}_N) \end{pmatrix}}$$

- Weight update

$$\boxed{\mathrm{w}^{\tau+1} = \mathrm{w}^{\tau} - \eta \nabla E_n}$$

$$\boxed{\phi(\mathrm{x}_n) = \begin{pmatrix} \phi_0(\mathrm{x}_n) & \phi_1(\mathrm{x}_n) & \ldots & \phi_{M-1}(\mathrm{x}_n) \end{pmatrix}^T}$$

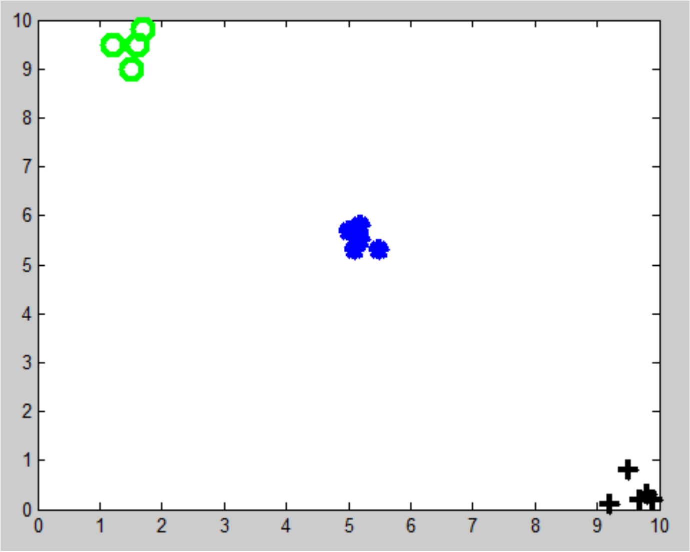
An Example of 3-class Logistic Regression

- Input Data

```
C1 =

    1.7000    9.8000
    1.2000    9.5000
    1.5000    9.0000
    1.6000    9.5000
    1.5000    9.0000
```

```
C2 =

    5.1000    5.3000
    5.2000    5.5000
    5.5000    5.3000
    5.2000    5.8000
    5.0000    5.7000
```

```
C3 =

    9.7000    0.2000
    9.8000    0.3000
    9.2000    0.1000
    9.5000    0.8000
    9.9000    0.2000
```



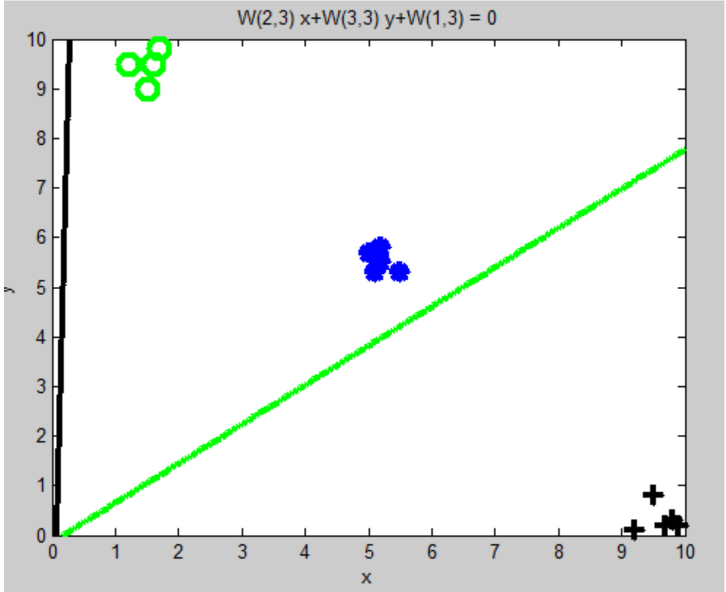
$\Phi_0(x)=1$, dummy feature

Three-class Logistic Regression

- Three weight vectors (Initial)

W =

0.4709	0.5486	0.1839
-2.5932	6.0889	-2.3215
3.2777	-1.2836	0.0483



- Gradient

Delta_E =

-0.0000	-0.0000	-0.0000
19.7000	1.2000	-20.9000
-21.4667	-2.2667	23.7333

Final Weight Vector, Gradient and Hessian (3-class)

- Weight Vector

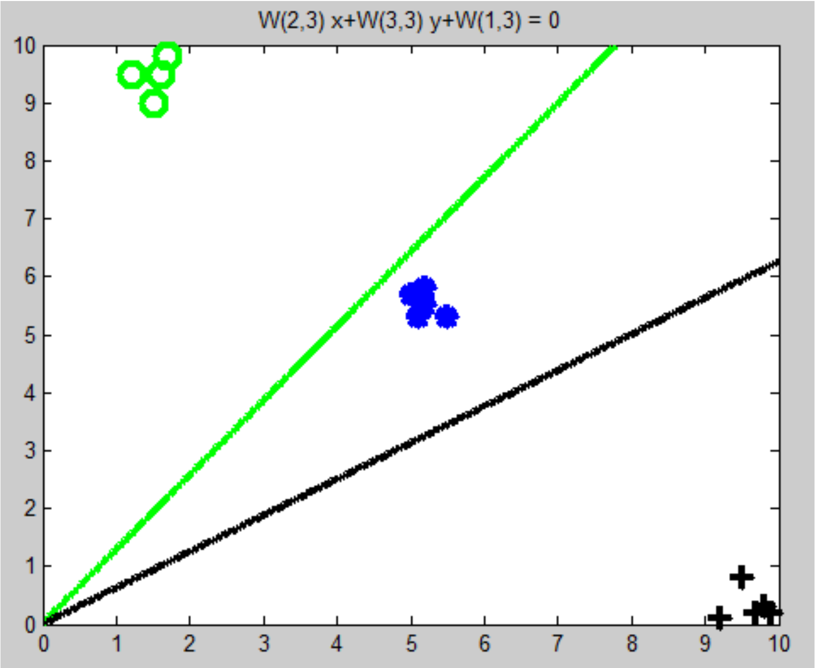
W =

0.4685	0.7553	0.1523
-1.4499	1.0309	1.9186
1.8775	0.7173	-1.3135

- Gradient

Delta_E =

0.0016	-0.0020	0.0004
0.0603	-0.0174	-0.0429
-0.0446	-0.0089	0.0535



Number of iterations : 6

Error (Initial and Final): 38.9394, 1.5000e-009

Keras Code invoking Library Functions

- About Keras
  - an open source neural network library written in Python
  - It is capable of running on top of TensorFlow.
  - Contains implementations of neural network building blocks
    - such as layers, objectives,activation functions, optimizers,
    - Tools to make working with image and text data easier.

Keras for MNIST Neural Network

- # Neural Network
- import keras
- from keras.datasets import mnist
- from keras.layers import Dense
- from keras.models import Sequential
- (x_train, y_train), (x_test, y_test) = mnist.load_data()
- num_classes=10
- image_vector_size=28*28
- x_train = x_train.reshape(x_train.shape[0], image_vector_size)
- x_test = x_test.reshape(x_test.shape[0], image_vector_size)
- y_train = keras.utils.to_categorical(y_train, num_classes)
- y_test = keras.utils.to_categorical(y_test, num_classes)
- image_size = 784 model = Sequential()
- model.add(Dense(units=32, activation='sigmoid', input_shape=(image_size,)))
- model.add(Dense(units=num_classes, activation='softmax'))
- model.compile(optimizer='sgd', loss='categorical_crossentropy',metrics=['accuracy'])
- history = model.fit(x_train, y_train, batch_size=128, epochs=10, verbose=False,validation_split=.1)
- loss,accuracy = model.evaluate(x_test, y_test, verbose=False)

Keras for MNIST SVM and Random Forest

- # SVM & RandomForest
- import numpy as np
- from sklearn.svm import SVC
- from sklearn.ensemble import RandomForestClassifier
- from sklearn.datasets import fetch_mldata
- mnist = fetch_mldata('MNIST original')
- n_train = 60000
- n_test = 10000
- indices = arange(len(mnist.data))
- train_idx = arange(0,n_train)
- test_idx = arange(n_train+1,n_train+n_test)
- X_train, y_train = mnist.data[train_idx], mnist.target[train_idx]
  X_test, y_test = mnist.data[test_idx], mnist.target[test_idx]
- # SVM
- classifier1 = SVC(kernel='rbf', C=2, gammma = 0.05);
  classifier1.fit(X_train, y_train)
- #RandomForestClassifier
- classifier2 = RandomForestClassifier(n_estimator=10);
  classifier2.fit(X_train, y_train)

MNIST Performance

- Neural Network: 0.980
- SVM: 0.918
- Random Forest: 0.950

Combining Models

- The simplest method of combining models is to average the predictions of the classifiers
- Such as by taking a vote on the decisions
  - One such method is the Borda count

Ranking	Candidate	Formula	Points	Relative points
1st	Andrew	n	5	1.00
2nd	Brian	n–1	4	0.80
3rd	Catherine	n–2	3	0.60
4th	David	n–3	2	0.40
5th	Elizabeth	n–4	1	0.20