CS-584 MACHINE LEARNING

ASSIGNMENT – 3

DISCRIMINATIVE LEARNIGNG

By

JAIMIN SANGHVI(A20344798)

PROBLEM STATEMENT

- **Discriminative models** (conditional models) are a class of models used in machine learning for modeling the dependence of an unobserved variable on an observed variable .
- It is done by modeling conditional probability distribution which can be used for predicting y from x.
- Discriminative models are opposed of generative models. They do not allow to generate samples from the joint distribution of x and y.
- In the given assignment3, We have to implement an algorithm for discriminative learning.
- In addition, We are provided by MNIST dataset for discriminative model evaluations.
- The MNIST dataset is subset to larger set available from NIST set. The MNIST database of handwritten digits, has 70,000 examples. This digit data is size-normalized and centered in fixed-size image.
- To gain high accuracy and performance, I should evaluate an algorithm using K-fold validation.

PROPOSED SOLUTION

- In the given assignment, I have implemented generative learning algorithm for generative learning.
- I have implemented parametric regression algorithms from the scratch using core logic of matrix multiplication in Python.
- I have evaluated an algorithm with K-fold cross validation. In addition, I have change number of iteration and learning rate to achieve best accuracy.
- I have implemented two separate program files for 2Class generative learning [LogisticRegression.ipynb] and KClass generative learning[LogisticRegressionKClass.ipynb]

How algorithm works

- I have implemented generative algorithm for 2Class in the following steps:
  - ✔ Load data into an object and put scale for 2Class and Kclass
  - ✔ Distinguish training and training data according to k-fold
  - ✔ Initialize theta matrix size of features
  - ✔ Find sigmoid function in multiple iteration and store final

data in dictionary
- ✔ Find discriminant function
- ✔ Predict y and compute confusion matrix
- ✔ Compute accuracy, precision, recall and f-measure errors

- I have implemented generative algorithm for KClass in the following steps:
  - ✔ Load data into an object and put scale for Kclass (class = 3 and more)
  - ✔ Distinguish training and training data according to k-fold
  - ✔ Initialize theta matrix size of features
  - ✔ Find softmax function in multiple iteration and store final data in dictionary
  - ✔ Find discriminant function
  - ✔ Predict y and compute confusion matrix
  - ✔ Compute accuracy, precision, recall and f-measure errors

- I have implemented MLP for KClass in the following steps:
  - ✔ Load data into an object and put scale for Kclass (class = 3 and more)
  - ✔ Distinguish training and training data according to k-fold
  - ✔ Initialize theta matrix size of features
  - ✔ Take test data as layer 0
  - ✔ Find sigmoid function and take it as layer 1
  - ✔ Find layer 1 error
  - ✔ Find synapse derivative functions
  - ✔ Find weight for 10000 iteration
  - ✔ Predict y for test data and find an accuracy

- In addition, I have evaluated given data set using Scikit learn libraries and compare it with derived training and testing errors as well as regression co-efficients

IMPLEMENTATION DETAILS

- I have used MNIST and IRIS datasets for each model evaluations. Hence it must need Internet connectivity
- The given implementations are data-oriented. Hence, User may require necessary changes to run same model for different datasets.
- I have implemented model for K-folds and print the single fold output with maximum accuracy at the end of code. In addition, I

have calculated confusion matrix as well as precision, recall and f-measure errors

DESIGN ISSUES

- It is little bit challenging to implement matrix multiplication without using any library
- It take too much time for understanding of MLP algorithms
- I have tried to import MLP libraries for scit-learn but it does not allow me to import libraries

SOLUTION

- To make mathematical calculations simple, I have used numpy library
- There was version issues of scikit library. To import MLP libraries, I have go through git libraries. I have downloaded all libraries and replace with old scikit-learn library for neural network that help me to import updated version libraries.

INSTRUCTION TO RUN

- I have implemented given problem solution in Jupyter Notebook.
- Instruction to run given project file
  i. Load given project in Jupyter notebook
  ii. Run **LogisticRegression.ipynb** file for 2class logistic regression. It will run 2Class logisitic regression for MNIST image dataset
  Note: For logistic regression, I have fetch dataset from web Hence it must need Internet connectivity
  iii. Run **LogisticRegression.ipynb** file for KClass regression. It will run KClass logisitic regression for MNIST image dataset
  Note: For logistic regression, I have fetch dataset from URL. Hence it must need Internet connectivity
  iv. Run **MLP_Kclass.ipynb** file for multilayer perceptron. It will run kClass MLP regression for IRIS dataset
  Note: For logistic regression, I have fetch dataset from URL. Hence it must need Internet connectivity

RESULT AND DISCUSSION

LOGISTIC REGRESSION

Algorithm

Logistic regression is probabilistic, linear classifier. It is parametrized by bias vector(b) and weight matrix(W). Classification is done by projecting vector on set of hyperplanes. Each one is corresponds to class. The resulting distance from input to hyperplane returns the probability that whether input is a member of corresponding class or not.

In given 2-class algorithm implementation,

- I have predict initial theta is 0 which is same length of data features.
- In next step, I have calculated sigmoid function and update theta value in multiple iteration.
- After getting final theta, I calculate discriminant function.
- After calculation of discriminant function, I have predicted value by comparing discriminant value.
- In the end, I have calculated confusion matrix, accuracy and errors(precision, recall and f-measure)

In given K-class algorithm implementation,

- I have predict initial theta is 0 which is same length of data features.
- In next step, I have calculated soft-max function and update theta value in multiple iteration.
- After getting final theta, I calculate discriminant function.
- After calculation of discriminant function, I have predicted value by comparing discriminant value.
- In the end, I have calculated confusion matrix, accuracy and errors(precision, recall and f-measure)

Results

2-class logistic regression

Accuracy of implemented model:

Accuracy = 99.8646820027 %
Precision = [0.99848024316109418, 0.99878048780487805]
Recall = [0.99848024316109418, 0.99878048780487805]
F-Measure = [0.99848024316109418, 0.99878048780487805]

I have evaluated logistic regression for 2-class using scikit-learn library.

Accuracy of scikit libraries:

Accuracy = 99.93 %

K-class logistic regression

By implemented algorithm:

Accuracy = 92.647 %
Precision = [0.92848024316109418, 0.92878048780487805]
Recall = [0.92848024316109418, 0.92878048780487805]
F-Measure = [0.92848024316109418, 0.92878048780487805]

I have evaluated logistic regression for 2-class using scikit-learn library. I have compared the above results with inbuild library function. The accuracy by scikit-learn library is as below:

Accuracy of scikit libraries:

Accuracy = 98.47 %

Comments

According to derived results I can say that the implemented model is accurate more 90% in all cases. By comparing results with scikit-learn library, I can say that implemented model is little bit less accurate than inbuild functions.

MULTILAYER PERCEPTION

Algorithm

- A multilayer perceptron is a neural network model that maps set of input data onto a set of appropriate outputs. An MLP consists of multiple layers of node in DG(Directed Graph), with each layer is fully connected to next node.
- The multilayer perceptron consists three or more layers (one input, one output and one or more hidden layers) of nonlinearly activating nodes. Thus it consider as deep neural network. Each node in one layer connects with weight to every node in following layer.
- Learning occurs in perceptron by changing weights after each piece of data is processed, based on amount of error in the output compared to expected results.
- I have back propogation technique.
- The MNIST dataset provides a training set of 60,000 handwritten digits and a validation set of 10,000 handwritten digits.
- The images have a size of 28*28 pixels. We want to train a two-layer perceptron to recognize handwritten digits, that is given a new 28*28 pixels image, the goal is to decide which digit it represents.

Results

I have derived backpropopagation update equation for the weights of output by minimizing the given error function.

The derived solution has started from next page.

Jaimin Sanghvi
A20344798.

Problem : 2 (A)

For

$\{x^{(i)}, y^{(i)}\}_{i=1}^{m}$, $x^{(i)} \in \mathbb{R}^n$
$y^{(i)} \in [0,1]$

$\hat{y}_j = \text{Sigmoid}(v_j^T z)$
$z_j = \text{sigmoid}(w_j^T g)$
$g_j = \text{sigmoid}(s_j^T x)$

error function : $\frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2$

Calculating $\Sigma(\{s_i\}, \{w_i\}, \{v_i\})$

$$\frac{\partial E}{\partial v_j} = \frac{\partial E}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial v_j}$$

$$\frac{\partial E}{\partial s_j} = \sum_{i=1}^{k} \frac{\partial E}{\partial \hat{y}_i} \sum_{a=1}^{b} \frac{\partial \hat{y}_i}{\partial z_a} \frac{\partial z_a}{\partial g_j} \frac{\partial g_j}{\partial s_j}$$

$$\frac{\partial E}{\partial w_j} = \sum_{i=1}^{k} \frac{\partial E}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z_j} \frac{\partial z_j}{\partial w_j}$$

So now calculate

$$\frac{\partial E}{\partial v_j} = \frac{\partial E}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial v_j}$$

$$= 1/2 \times 2 \sum_{i=1}^{m} (y_j^{(i)} - \hat{y}_j^{(i)}) \cdot y_j^{(i)} (1 - y_j^{(i)}) z^{(i)}$$

$$= \sum_{i=1}^{m} (y_j^{(i)} - \hat{y}_j^{(i)}) \hat{y}^{(i)} (1 - \hat{y}_j^{(i)}) z^{(i)}$$

$$\therefore v_j \leftarrow v_{j-1} \eta \frac{\partial E}{\partial v_j}$$

Now,

$$\frac{\partial E}{\partial w_j} = \sum_{i=1}^{k} \frac{\partial E}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_j} \frac{\partial z_j}{\partial w_j}$$

$$w_j \leftarrow w_{j-1} - \eta \left(\frac{\partial E}{\partial w_j} \right)$$

$$w_j \leftarrow w_{j-1} - \eta \left(\sum_{i=1}^{m} \sum_{j=1}^{k} (y_i^{(i)} - \hat{y}_j^{(i)}) \cdot y_j^{(i)} \right.$$

$$\left. (1 - \hat{y}_j^{(i)}) \, v_j \, g_i^{(i)} (1 - z_j^{(i)}) g^{(i)} \right)$$

Now,

$$\frac{\partial E}{\partial s_j} = \sum_{j=1}^{k} \frac{\partial E}{\partial y_j} \cdot \sum_{a=1}^{b} \frac{\partial y_j}{\partial z_a} \cdot \frac{\partial z_a}{\partial g_j} \frac{\partial g_j}{\partial g_j}$$

$$\frac{\partial E}{\partial s_j} = \sum_{i=1}^{m} \sum_{j=1}^{k} (y_i^{(i)} - \hat{y}_j^{(i)}) \cdot \sum_{a=1}^{b} \hat{y}^{(a)}(1-\hat{y}^{(i)})$$

$$V_{ij} z_a (1-z_a) w^{(i)} \cdot g_j$$

$$(1-g_j) \cdot x^{(i)}$$

$$\delta_j \leftarrow s_{j-1} \cdot \eta \frac{\partial E}{\partial s_j}$$

Classification problem solved in class.

All parameters are same as used in regression but the objective function

$$E = \sum_{i=1}^{m} \sum_{j=1}^{k} P(y=j/x^{(i)}) 1(y^{(i)}=j)$$

which can be simplified as,

$$E = \sum_{i=1}^{m} \sum_{j=1}^{k} 1(y^{(i)}=j) \log(\hat{y}^{(i)})$$

Calculated parameters in regression

$$\frac{\partial E}{\partial v_j} = \frac{\partial E}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial v_j} = \sum_{i=1}^{m} \sum_{j=0}^{k} \frac{I(y^{(i)}=j)}{\hat{y}^{(i)}}$$

$$\therefore \quad v_j \leftarrow v_{j-1} - \eta \frac{\partial E}{\partial v_j}$$

Now:

$$\frac{\partial E}{\partial w_j} = \sum_{a=1}^{b} \frac{\partial E}{\partial \hat{y}_a} \cdot \frac{\partial \hat{y}_a}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_j}$$

$$\therefore \quad \frac{\partial E}{\partial w_j} = \sum_{i=1}^{m} \sum_{j=1}^{k} \sum_{a=1}^{b} \frac{I(y^{(i)}=j)}{y_{a,j}^{(i)}} \cdot \hat{y}_a^{(i)} (I - \hat{y}_a^{(i)}) \cdot v_{aj} \cdot z_a^{(i)} (1 - z_a^{(i)}) g^{(i)}$$

$$\therefore \quad w_j \leftarrow w_{j-1} - \eta \frac{\partial E}{\partial w_j}$$

$$\frac{\partial E}{\partial s_j} = \sum_{a=1}^{b} \frac{\partial E}{\partial \hat{y}_a} \sum_{c=1}^{d} \frac{\partial \hat{y}_a}{\partial z_c} \cdot \frac{\partial z_c}{\partial q_j} \cdot \frac{\partial q_j}{\partial s_j}$$

$$= \sum_{i=1}^{m} \sum_{l=1}^{*} \cdot \sum_{a=1}^{b} 1\left(\frac{y_a^{(i)}}{\hat{y}_j^{(i)}} = j\right) \cdot \sum_{c=1}^{d} \hat{y}_a^{(i)} \cdot (1 - \hat{y}_a^{(i)})$$

$$\cdot v_{cj}^{T} \cdot z_c^{(i)} (1 - z_c^{(i)}) \cdot w_j^{(i)} q_j (1 - q_j) x$$

$$s_j \leftarrow s_{j-1} \cdot \eta \frac{\partial E}{\partial s_j}$$

Results by implemented algorithm:

Accuracy is : 67.1428571429 %

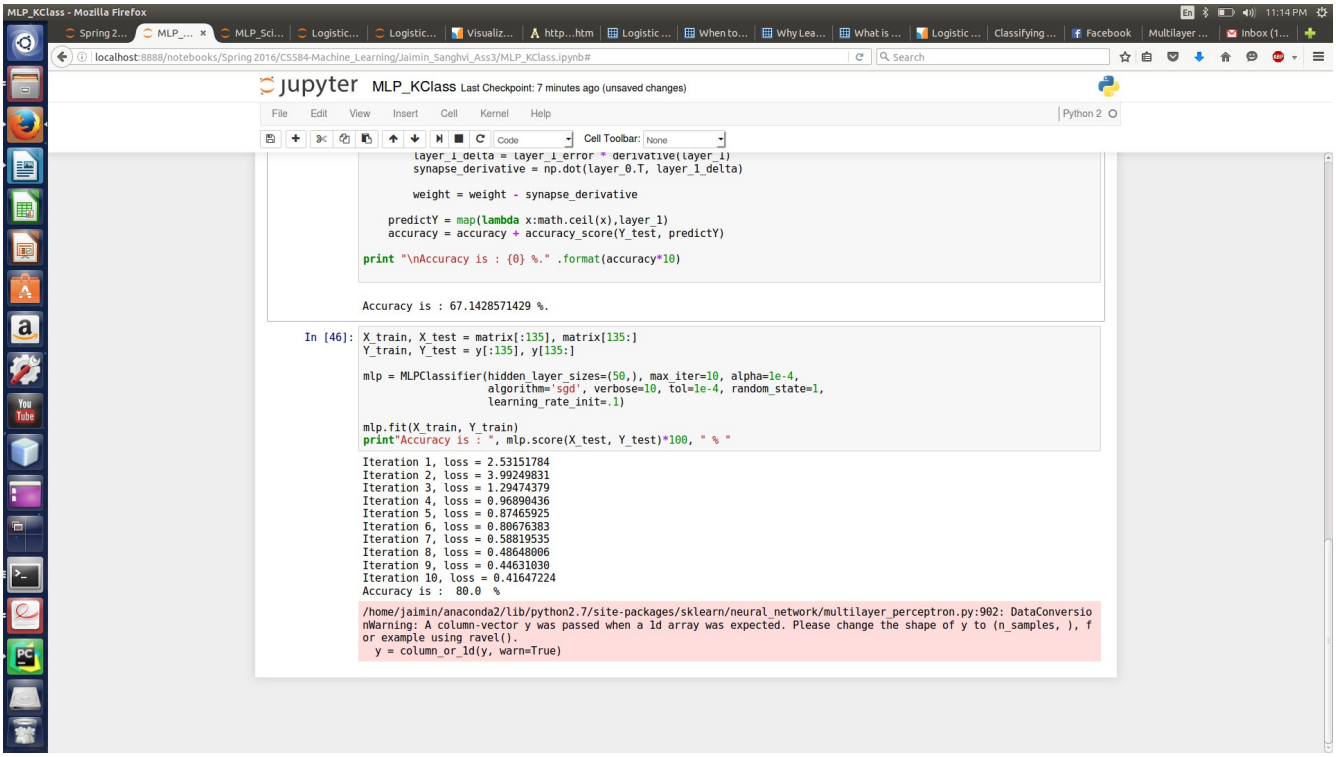
I have evaluated MLP using **scikit-learn** libraries and compare the results with evaluated results. The output by scikit-learn library is as below

Result by scikit-learn library:

```
Iteration 1, loss = 2.53151784
Iteration 2, loss = 3.99249831
Iteration 3, loss = 1.29474379
Iteration 4, loss = 0.96890436
Iteration 5, loss = 0.87465925
Iteration 6, loss = 0.80676383
Iteration 7, loss = 0.58819535
Iteration 8, loss = 0.48648006
Iteration 9, loss = 0.44631030
Iteration 10, loss = 0.41647224
```

Accuracy is : 80.0 %

Output



Comments

According to derived results I can say that the implemented model is accurate around 65% in all cases. By comparing results with scikit-learn library, I can say that implemented model is little bit less accurate than inbuild functions.

REFERENCES

1.http://www.astro.ufl.edu/~warner/prog/python.html
2.http://www.holehouse.org/mlclass/10_Advice_for_applying_machine_learning.html
3.https://en.wikipedia.org/wiki/Multilayer_perceptron
4.http://deeplearning.net/tutorial/logreg.html
5.http://deeplearning.net/tutorial/mlp.html