Lab Course Machine Learning
Exercise 4

Prof. Dr. Dr. Lars Schmidt-Thieme,
Mofassir ul Islam Arif
Information Systems and Machine Learning Lab
University of Hildesheim
Submission: 22.11.2019 LearnWeb 3115

November 16, 2019

Instructions

Please following these instructions for solving and submitting the exercise sheet.

1. You should submit two things a) python scripts(zipped) / jupyter notebook and b) a pdf document.
2. In the pdf document you will explain your approach (i.e. how you solved a given problem), and present your results in form of graphs and tables.
3. The submission should be made before the deadline, only through learnweb.
4. Unless explicitly mentioned, you are not allowed to use scikit, sklearn or any other library for solve any part. **All implementations must be done yourself**.

1 Exercise Sheet 4

Classification dataset

Tic Tac Toe:

You are required to pre-process given datasets.

1. Convert any non-numeric values to numeric values. For example you can replace a country name with an integer value or more appropriately use hot-one encoding. [Hint: use hashmap (dict) or pandas.get_dummies]. Please explain your solution.
2. This dataset is unbalanced, (**show how we can confirm this**). Explain what is stratified sampling and Implement a stratified sampler.
3. Split the data into a train(80%) and test(20%).

learn-logreg-GA$(\mathcal{D}^{\text{train}} := \{(x_1, y_1), \ldots, (x_N, y_N)\}, \mu, t_{\max} \in \mathbb{N}, \epsilon \in \mathbb{R}^+)$:
  $X := (x_1, x_2, \ldots, x_N)^T$
  $y := (y_1, y_2, \ldots, y_N)^T$
  $\hat{\beta} := 0_M$
  $\ell := \sum_{n=1}^{N} y_n \langle x_n, \hat{\beta} \rangle - \log(1 + e^{\langle x_n, \hat{\beta} \rangle})$
  for $t = 1, \ldots, t_{\max}$:
    $\hat{y} := (1/(1 + e^{-\hat{\beta}^T x_n})_{n \in 1:N}$
    $\hat{\beta} := \hat{\beta} + \mu \cdot X^T(y - \hat{y})$
    $\ell^{\text{old}} := \ell$
    $\ell := \sum_{n=1}^{N} y_n \langle x_n, \hat{\beta} \rangle - \log(1 + e^{\langle x_n, \hat{\beta} \rangle})$
    if $\ell - \ell^{\text{old}} < \epsilon$:
      return $\hat{\beta}$

  raise exception "not converged in $t_{\max}$ iterations"

Figure 1: Algorithm: Learn-logreg-GA

2 Logistic Regression

Exercise 1: Logistic Regression with Gradient Ascent (10 Points)

In this part you are required to implement linear classification with stochastic gradient ascent algorithm. Reference lecture **ml-03-A2-linear-classification.pdf**

- 1. A set of training data $D_{train} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(N)}, y^{(N)})\}$, where $x \in R^M, y \in \{0, 1\}$ N is number of training examples and M is number of features
- Logistic Regression model is given as $\hat{y}^n = \sigma(\beta^T \mathbf{x}^n)$ where σ is a logistic function $\frac{1}{1+e^{-\beta^T \mathbf{x}^n}}$
- Optimize the loglikelihood function $log(L_D^{cond})$ using Gradient Ascent algorithm. Implement **(learn-logreg-GA)**. Choose i_{max} between 100 to 1000.
- You will use *bolddriver* as the step length controller.
  - In each iteration of the algorithm calculate $|f(x_{i-1}) - f(x_i)|$ and at the end of learning, plot it against iteration number i. Explain the graph.
  - In each iteration step also calculate logloss on test set https://www.kaggle.com/wiki/LogarithmicLoss, plot it against iteration number i. Explain the graph.

3 Exercise 2: Implement Newton Algorithm (learning rate) (10 Points)

In this task you have to implement Newton Algorithm given in Fig. 3. Use the Tic-tac-toe dataset.

minimize-Newton($f : \mathbb{R}^N \to \mathbb{R}, x^{(0)} \in \mathbb{R}^N, \mu, t_{\max} \in \mathbb{N}, \epsilon \in \mathbb{R}^+$):
  for $t := 1, \ldots, t_{\max}$:
    $g := \nabla f(x^{(t-1)})$
    $H := \nabla^2 f(x^{(t-1)})$
    $x^{(t)} := x^{(t-1)} - \mu H^{-1} g$
    if $f(x^{(t-1)}) - f(x^{(t)}) < \epsilon$:
      return $x^{(t)}$
  raise exception "not converged in $t_{\max}$ iterations"

$x^{(0)}$ start value
μ (fixed) step length / learning rate
$t_{\max}$ maximal number of iterations
ϵ minimum stepwise improvement
$\nabla f(x) \in \mathbb{R}^N$: gradient, $(\nabla f(x))_n = \frac{\partial}{\partial x_n} f(x)$
$\nabla^2 f(x) \in \mathbb{R}^{N \times N}$: Hessian matrix, $\nabla^2 f(x)_{n,m} = \frac{\partial^2 f}{\partial x_n \partial x_m}(x)$

Figure 2: Algorithm: minimize Newton

learn-logreg-Newton($\mathcal{D}^{\text{train}} := \{(x_1, y_1), \ldots, (x_N, y_N)\}, \mu, t_{\max} \in \mathbb{N}, \epsilon \in \mathbb{R}^+$):
  $\ell := -\log L_{\mathcal{D}}^{\text{cond}}(\hat{\beta}) := \sum_{n=1}^{N} y_n \langle x_n, \hat{\beta} \rangle - \log(1 + e^{\langle x_n, \hat{\beta} \rangle})$
  $\hat{\beta} :=$ minimize-Newton$(\ell, 0_M, \mu, t_{\max}, \epsilon)$
  return $\hat{\beta}$

Figure 3: Algorithm: Newton Algorithm

- In each iteration of the algorithm calculate $|f(x_{i-1}) f(x_i)|$and at the end of learning, plot it against iteration number i. Explain the graph.
- In each iteration step also calculate logloss on test set

  https://www.kaggle.com/wiki/LogarithmicLoss,

  plot it against iteration number i. Explain the graph.

Comment on the behavior of the two methods. which is converging faster? (show this in your plots)

3.1 ANNEX

- You can use numpy or scipy in build methods for doing linear algebra operations
- You can use pandas to read and processing data
- You can use matplotlib for plotting.
- You should not use any machine learning library for solving the problem i.e. scikit-learn etc. If you use them you will not get any points for the task.