11-741/11-641/11-441 – Machine Learning for Text Mining **Due: Apr 2, 2020, 11:59pm**

Instructor: Yiming Yang

Homework 4: Support Vector Machines

In this assignment, you will be developing two optimization methods to solve the objective function of Support Vector Machines and compare them empirically on the provided dataset. In addition, you will implement SGD optimization on the PMF method that was introduced in the earlier Collaborative Filtering assignment without relying on automatic differentiation facilities.
This assignment consists of three major sections:

- Writeup: derive certain properties of the objective function and outline the optimization update procedure.
- Implementation: implement stochastic gradient method and Newton method for L2-loss SVM as well as stochastic gradient method on PMF.
- Experiment: compare two optimization methods for SVM with respect to relative function value decrease and classification accuracies.

It is important to do a good job on your report, so please start early to finish the program and leave enough time to write the report for each part. Details for each of the deliverables follow. To simplify your work, a report template is provided. (It's okay to use LaTeX, but you must follow the same format as the template.)

1 Writeup (40 pts)

Given a set of instance-label pairs $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$, $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y = \{-1, +1\}$, the objective functions of linear support vector machines with different loss functions $\xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$ are

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \frac{\lambda}{n}\sum_{i=1}^{n}\xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i) \tag{1}$$

where $\lambda > 0$ is a penalty hyper-parameter. The two common loss functions are

$$\text{L1-loss:} \quad \xi_{L1}(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = \max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0), \tag{2}$$

$$\text{L2-loss:} \quad \xi_{L2}(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = \max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0)^2. \tag{3}$$

Here, we add a feature dimension with the constant 1 into input vectors $\boldsymbol{x}$. Then the bias term is included in the dot product, $\boldsymbol{w}^T\boldsymbol{x}_i$. Notice that it is not included in the data, you should append it manually in your program. In this homework, we will mainly focus on the **L2-loss linear SVM**:

$$\min_{\boldsymbol{w}} \quad f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \frac{\lambda}{n}\sum_{i=1}^{n}\max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0)^2 \tag{4}$$

1.1 Gradient (10 pts)

Show that the gradient of Equation (4) is of the form

$$\nabla f(\boldsymbol{w}) = \boldsymbol{w} + \frac{2\lambda}{n} X_{I,:}^T (X_{I,:}\boldsymbol{w} - \boldsymbol{y}_I), \tag{5}$$

where $I \equiv \{i | 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i > 0\}$ is an index set, $\boldsymbol{y} = [y_1, \ldots, y_n]^T$, and $X = [\boldsymbol{x}_1^T; \ldots; \boldsymbol{x}_n^T] \in \mathbb{R}^{n \times d}$.

1.2 Hessian (10 pts)

Show that the *generalized Hessian* of Equation (4) is of the form

$$\nabla^2 f(\boldsymbol{w}) = I_d + \frac{2\lambda}{n} X^T D X, \tag{6}$$

where I_d is the identity matrix of dimension d and $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix with the following elements:

$$D_{ii} = \begin{cases} 1, & \text{if } i \in I, \\ 0, & \text{otherwise.} \end{cases}$$

1.3 Optimality (10 pts)

Prove that the unconstrained optimization problem (4) is equivalent to the following constrained optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w}, \boldsymbol{\xi}} \quad & \frac{1}{2}\boldsymbol{w}^T \boldsymbol{w} + \frac{\lambda}{n} \sum_{i=1}^{n} \xi_i^2 \\ \text{s.t.} \quad & y_i \boldsymbol{w}^T \boldsymbol{x}_i \geq 1 - \xi_i \\ & \xi_i \geq 0. \end{aligned} \tag{7}$$

Note that two optimization problems are considered equivalent if their **optimal function values** are the same.

1.4 Algorithm Pseudo Code (10 pts)

Outline the pseudo code of the optimization update procedure for mini-batch stochastic gradient method and Newton method.

2 Implementation Details (40 pts)

2.1 Implement mini-batch stochastic gradient solver for SVM (10 pts)

Implement the following components:

- mini-batch updates (5 pts)
- annealing learning rate (5 pts)

Annealing learning rate is a function of update iterations

$$\gamma_t = \frac{\gamma_0}{1 + \beta t}$$

where γ_t is the learning rate at t-th epoch, γ_0 is the initial learning rate, and β is the decreasing rate. Use $t = 0$ for the first update.

2.2 Implement mini-batch stochastic gradient solver for PMF (10 pts)

Implement mini batch stochastic gradient solver for the PMF method described in the Collaborative Filtering assignment, use eq. (4) in the original PMF paper for optimization.

Ruslan Salakhutdinov and Andriy Mnih. "Probabilistic matrix factorization." Advances in Neural Information Processing Systems (NIPS). 2007.

We have provided a template **pmf.py** for you in the provided resource. Please implement the **train** function without making modifications on other part of the code. Also, do not import new external packages. You should be able to implement the algorithm with numpy only. The autograder will run some checks to make sure your code is correct.

For the purpose of this homework, just implement mini-batch version eq 4 without any imputation and rescaling.

2.3 Implement Newton solver for SVM (20 pts)

HINT: for Newton solver to run efficiently on large-scale dataset, you should consider using existing linear equation solver such conjugate gradient method to solve the following linear system

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{d} = -\nabla f(\boldsymbol{w}), \tag{8}$$

where $\boldsymbol{d}$ is the descent direction in each iteration. **You can leverage the *ConjugateGradient.py* provided by the TAs.**

3 Experiments (20 pts)

You can use the realsim dataset to debug your code. The test accuracy should be around 97% for learning rate of 0.001 and batch size of 1000. You need not perform exhaustive parameter search but analyze the below graphs and choose the parameters wisely.

Plot figures for the following metrics for both realsim and covtype datasets:

- Relative function value difference versus training time (5 pts)
- gradient norm versus training time (5 pts)
- test set accuracies versus training time (5 pts)

where the relative function value difference is defined as

$$\frac{f(\boldsymbol{w}^t) - f(\boldsymbol{w}^*)}{f(\boldsymbol{w}^*)}$$

Hint: Report the gradient norm $\|\nabla f(w)\|_2$ of the gradient in each update, i.e., the gradient is computed based on a mini-batch in minibatch-SGD, but it is computed based on the whole dataset in Newton method.

Hint: For the relative function, you should compute $f(\boldsymbol{w}^t)$ based on the whole training set like the given $f(\boldsymbol{w}^*)$.

Hint: For one metric, you should plot two methods in one figure which can better compare two methods. So you have two datasets and three metrics, totally 6 figures.

Discuss the difference between mini-batch SGD and Newton method in terms of the three types of figures (5 pts).

Dataset	n	d	λ	5-fold CV	$f(\boldsymbol{w}^*)$
realsim	57847	20958	7230.875	97.3655%	669.664812
covtype	464809	54	3631.3203125	75.6661%	2541.664519

Table 1: Data statistics.

4 Handout

A handout.zip is provided to you with all the required files.

```
handout.zip
├── data
│   ├── realsim.scale.trn.libsvm
│   ├── realsim.scale.tst.libsvm
│   ├── covtype.scale.trn.libsvm
│   └── covtype.scale.tst.libsvm
├── src
│   ├── main.py
│   ├── hw4.sh
│   ├── pmf.py
│   └── conjugateGradient.py
├── resources
│   └── optval.txt
└── HW4-Template.docx
```

There are two datasets, realsim and covtype, stored in the LIBSVM format. Each row is of the form:

$$< label > < index1 >:< value1 > < index2 >:< value2 > \ldots$$

The pair $< index >:< value >$ gives a feature value: $< index >$ is an integer starting from 1 and $< value >$ is a real number. The postfix $*.trn.libsvm$ indicates the training data while $*.tst.libsvm$ indicates the test data. **Hint:** you can utilize the *load_svmlight_file* and *dump_svmlight_file* from Sklearn library[1] to read and write LIBSVM file in python.

ConjugateGradient.py is a workable function interface that solves the linear equation of Newton method

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{d} = -\nabla f(\boldsymbol{w})$$

optval.txt contains the necessary data statistics, as shown in Table 1. n is the number of instances, d is the number of features, λ is the penalty hyper-parameters in the objective function Eq. (4) and $f(\boldsymbol{w}^*)$ is the optimal function value solved by LIBLINEAR solver. In this assignment, **Make sure you set λ according to Table 1 in order to have the same optimal function value**.

Details about the script can be found in the Source Code section.

5 What to Turn In

5.1 Written report [60pts]

Write your report in PDF format. Please include your **name and Andrew ID** at the top of the first page of your report. Your report must contain the following sections, **each clearly labeled as an independent section**.

[1] http://scikit-learn.org/stable/

1. **Statement of Assurance:** You must certify that all of the material that you submit is original work that was done only by you. If your report does not have this statement, it will not be graded.
2. **Writeup [40pts]**
3. **Experiments [20pts]**

5.2 Source Code [40pts]

The TAs will look at your source code, so make sure that it is legible, has reasonable documentation, and can be understood by others. This is a Computer Science class lesson - the instructor will actually care about your source code. By default, the code will be tested on Ubuntu Linux 18.04 and Python 3.6.7, please make sure your code can run successfully under such environment. If you plan to use a different version please document the choice in a README.txt file. If you plan to use other programming languages, please discuss with TAs in advance.

6 Restrictions

1. You should not include datasets in the submission. For the SVM, TAs will run only your hw4.sh with two arguments, first argument specifying the path of train file and second argument specifying the path of test file i.e., **./hw4.sh train-file test-file**. Sample script and code file are also provided for your reference. You may use the filenames to determine the appropriate parameters to be used.
2. For PMF, you should modify the provided pmf.py template by implementing the train function. Therefore, for this part, it is required to use Python 3.
3. Your code should run for both solvers on the dataset in the argument using the parameters you have choosen and print the final test accuracy and total time taken. Your system must do this task entirely automatically. No manual intervention is allowed. The TAs **will not** modify your source code to run the experiment.
4. Please test the script to verify that you include everything necessary to run it.You must write all of the software yourself. No external optimization package is allowed.
5. You are encouraged to use Python 3 for the homework. It's okay to use scientific packages such as Numpy and Scipy for matrix multiplication, but you must write the algorithm by yourself. If you are unsure whether a particular API could be used, ask about it on Piazza. You must include the package dependency files such as requirements.txt or environment.yml if you used an package.

7 Submission Checklist

The report and the source code should be submitted separately.

For the source code, please compress all the following files into a .zip file for submission.

1. All source code in **src** folder

Sample folder structure:

```
HW4.zip
└── src
    ├── source-files
    ├── hw4.sh
    └── pmf.py
```