Siping Ji

sji

Homework 5

0. Statement of Assurance

I hereby certify that all of the material that I submit is original work that was done only by myself.

1. Gradient Descent Approach (15%)

1.1 Derive the gradient of the likelihood

Derive the formula of the gradient

The log likelihood function of logistic regression with l2-norm regularization is as follows:

$$l(w) = log(P(D \mid w)) - \frac{1}{2} C \cdot \|w\|^2$$

$$l(w) = \sum_{i=1}^{m} [y^i \, log(g(w^T x^i)) + (1-y^i) log(1 - g(w^T x^i))] - \frac{1}{2} C \cdot \|w\|^2$$

$$where \; g(z) \; is \; the \; sigmoid \; function, \; g(z) = \frac{1}{1+e^{-z}}$$

The gradient of w over an instance (x, y) can be derived as follows:

$$\frac{\delta}{\delta w_j} l(w) = \left(y \frac{1}{g(w^T x)} - (1-y) \frac{1}{1 - g(w^T x)} \right) \frac{\delta}{\delta w_j} g(w^T x) - C \cdot w_j$$

$$\frac{\delta}{\delta w_j} l(w) = \left(y \frac{1}{g(w^T x)} - (1-y) \frac{1}{1 - g(w^T x)} \right) \frac{\delta g(w^T x)}{\delta w^T x} \frac{\delta w^T x}{\delta w_i} - C \cdot w_j$$

$$\frac{\delta}{\delta w_j} l(w) = \left(y \frac{1}{g(w^T x)} - (1-y) \frac{1}{1 - g(w^T x)} \right) g(w^T x)(1 - g(w^T x)) \, x_i - C \cdot w_j$$

$$\frac{\delta}{\delta w_j} l(w) = (y - g(w^T x))\ x_j - C \cdot w_j$$

The above formula gives the derivative of the likelihood function for a instance (x,y) over wi. Note that, from above derivation, we utilized a fun fact about the derivative of a sigmoid function:

$$\frac{\delta g(z)}{z} = g(z) \cdot (1 - g(z))$$

The gradient of w can therefore written as:

$$gradient\ of\ l(w)\ over\ w = \left[\frac{\delta}{\delta w_0} l(w), \frac{\delta}{\delta w_1} l(w), \ldots \frac{\delta}{\delta w_n} l(w) \right]^T$$

$$gradient\ of\ l(w)\ over\ w = (y - g(w^T x))x - C \cdot w$$

1.2 Gradient descent algorithm

Outline the gradient descent algorithm

The algorithm for stochastic gradient descent is as follows:

Randomly initialize vector w

repeat {

randomly shuffle the training set

for each instance i {

$$w = w + \alpha \left[(y^{(i)} - g(w^T x^{(i)}))\ x^{(i)} - C.w \right]$$

}

}

until convergence of likelihood function or reaches maximum rounds of iterations.

2. Evaluation Results (40%)

2.1 Logistic Regression

I set different learning rate with respect to different value of C in order to make the optimization converge.

For c >= 10, I set learning rate = 1e-5; for c ~ [1, 10), learning rate = 0.005; for c ~ [0.1, 1), learning rate = 0.005; for c < 0.1, learning rate = 0.02.

Without intercept term

C	MICRO-AVG			MACRO-AVG		
	Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**
0.0001	0.6564246	0.6564246	0.6564246	0.6515447	0.6448380	0.6436140
0.001	0.6410615	0.6410615	0.6410615	0.6448916	0.6245117	0.6177150
0.01	0.5858939	0.5858939	0.5858939	0.6537998	0.5599044	0.5572091
0.1	0.5621508	0.5621508	0.5621508	0.6504899	0.5320641	0.5261052
1	0.5600559	0.5600559	0.5600559	0.6241336	0.5310991	0.5226681
10	0.5467877	0.5467877	0.5467877	0.6687927	0.5165042	0.5139658
50	0.5432961	0.5432961	0.5432961	0.6622043	0.5153366	0.5156215
100	0.5425978	0.5425978	0.5425978	0.6710657	0.5149855	0.5216174

With intercept term, not punishing w0 in the regularization term.

C	MICRO-AVG			MACRO-AVG		
	Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**
0.0001	0.6529330	0.6529330	0.6529330	0.6592572	0.6393491	0.6392032
0.001	0.5914804	0.5914804	0.5914804	0.6688013	0.5647166	0.5660541

0.01	0.1780726	0.1780726	0.1780726	0.2627327	0.1328398	0.0869515
0.1	0.0893855	0.0893855	0.0893855	0.0640441	0.0640441	0.0640441
1	0.0886872	0.0886872	0.0886872	0.0052169	0.0588235	0.0095838
10	0.0886872	0.0886872	0.0886872	0.0052169	0.0588235	0.0095838
50	0.0886872	0.0886872	0.0886872	0.0052169	0.0588235	0.0095838
100	0.0886872	0.0886872	0.0886872	0.0052169	0.0588235	0.0095838

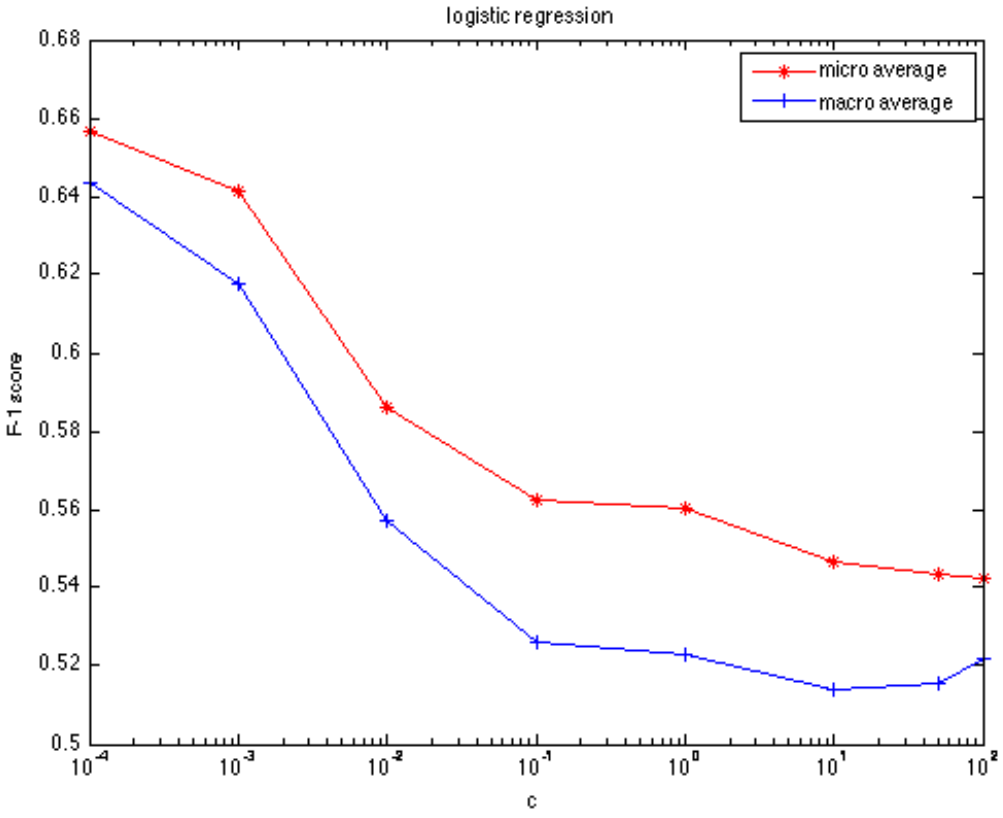
2.2 SVM

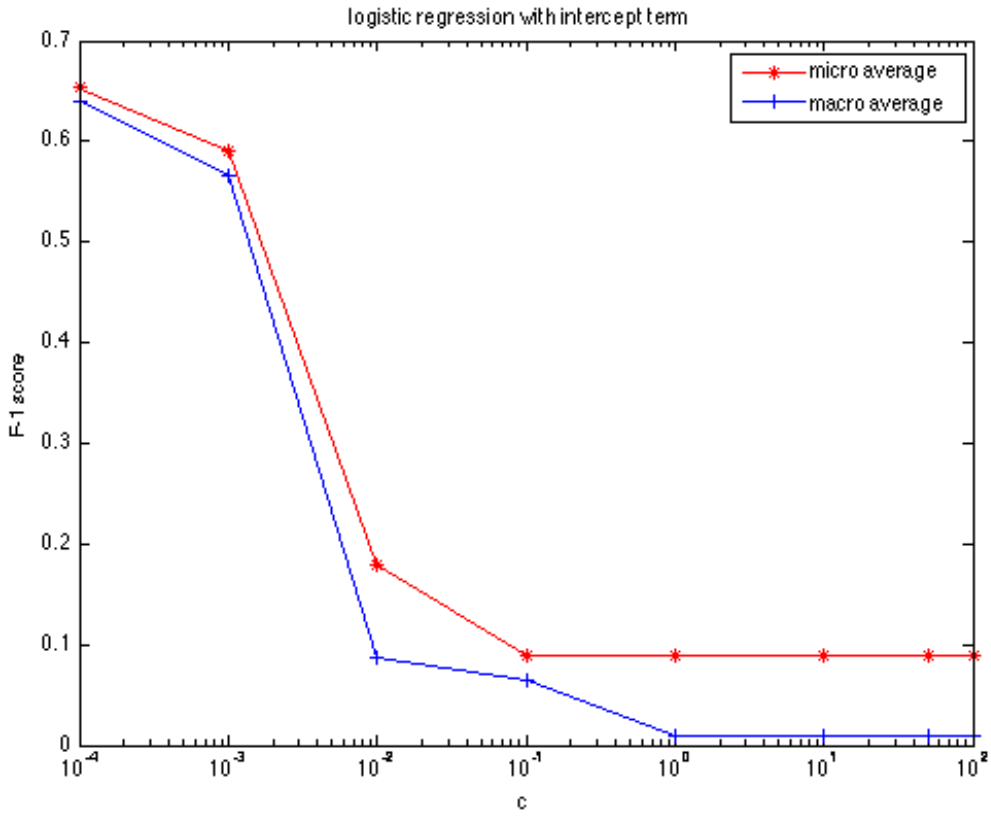
C	**MICRO-AVG**			**MACRO-AVG**		
	Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**
0.0001	0.5432961	0.5432961	0.5432961	0.6206298	0.5422486	0.5369033
0.001	0.6138268	0.6138268	0.6138268	0.5992586	0.6050103	0.5892401
0.01	0.6236034	0.6236034	0.6236034	0.6129197	0.6134529	0.5985171
0.1	0.6256983	0.6256983	0.6256983	0.6136163	0.6161320	0.6010657
1	0.6354749	0.6354749	0.6354749	0.6222243	0.6247685	0.6144372
10	0.6194134	0.6194134	0.6194134	0.6161295	0.6122009	0.6108887
50	0.6173184	0.6173184	0.6173184	0.6144542	0.6108113	0.6094651
100	0.6173184	0.6173184	0.6173184	0.6144542	0.6108113	0.6094651

2.3 Training time

C	LR Time (s)	SVM Time (s)
0.0001	77(75)	8.29686498642
0.001	71(76)	5.75914692879
0.01	75(68)	6.50361514091
0.1	71(75)	6.93973898888
1	66(67)	7.51672315598
10	15(70)	8.77752900124
50	4(69)	8.11134886742
100	4(71)	8.63958692551

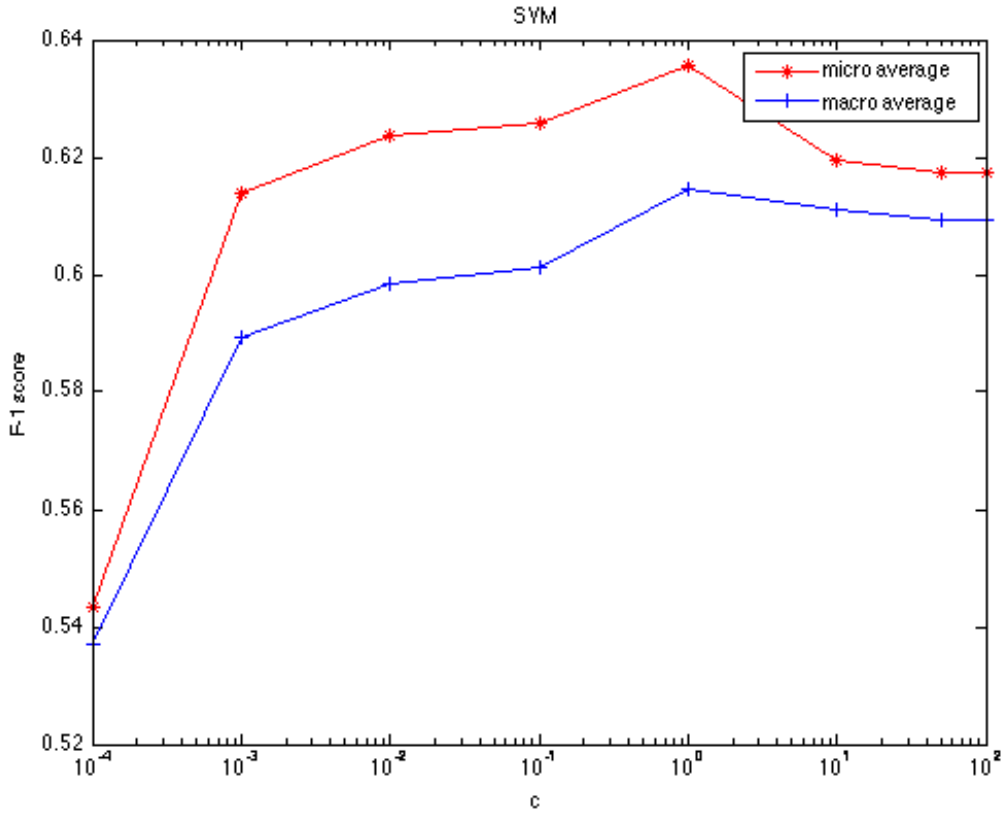
Clarification: For the LR column, the number in the bracket represent the time for training when the intercept term is added.





2.4 Logistic Regression graph

Plot a graph of the Micro-F1, Macro-F1 for different values of "C" for Logistic Regression.



Below is the original plot without adding intercept term.

Logistic Regression with intercept term.

2.5 SVM graph

Plot a graph of the Micro-F1, Macro-F1 for different values of "C" for SVM.

3. Analysis of results (30%)

Compare the SVM and Logistic regression classifiers. Discuss specifically about the performance of each and the time taken to train.

1. Performance of Logistic Regression with respect to parameter C. From the graph I showed in the last section, we can see that we reach the best micro/micro average F1-score when C is set to 0.0001. And the performance monotonically decreases as the C grows larger. It seems that the more weight I put on the regularization term, it further weakens the discriminative power of the classifier. I also tried to train a model of logistic regression without the regularization term, the F1-score reaches 66%, even better. The reason why regularization term deteriorate the model might be two-fold: 1. The model we train only use original feature as basis, and train a linear combination as the hypothesis hyperplane. So it's rather a simple model, not that easy to overfit. 2. We have enough training data (2863), relatively enough to represent the general distribution, i.e. therefore, overfitting does not occur, even we don't have a regularization term to punish model with large w. One thing I noticed about the performance after including the intercept term in the model is that the performance drops to the bottom with large value of C. The probable cause might be that since we are not punishing w0 while punishing other weights with large penalty, the intercept term will play a much larger role than all other original features. In the binary subproblem where positive and negative labels are extremely imbalanced, w0 tends to be a very large negative value after convergence. Therefore, g(w*x + w0) tends to be zero for all binary logistic regression models, thus makes the voting strategy unable to function correctly(all binary classifier predicts that the incoming instance belongs to other classes with 100% confidence).

2. Running time of Logistic Regression with respect to parameter C. One observation about c is that when c grows larger, we need fewer time to train the model. The probable reason might be that, in order to make the optimization process actually converge, I have to choose small value of learning rate for larger C, because when C is too large, the value of weights actually fluctuates if we also use large learning rate. Therefore, with small learning rate, the change of optimization target function

(likelihood function with gaussian prior(l2-norm regularization term)) will have a small change before and after the weight update, and therefore, cause the optimization process faster to stop when convergence occur. When the intercept term is included, this trend disappears, the probable cause is that as we no longer penalize w0 in the intercept term while still optimizing it in the likelihood term, it will take longer to converge as we are getting enough progress in every step of gradient ascent.

3. Performance of SVM with respect to parameter C. Not like the trend we see in the graph for logistic regression, the performance first increases as we set larger value of c; it then reaches its best performance when c = 1, and finally decreases when c grows further larger. This is reasonable if we carefully think about what c represents in the SVM model. It controls how much we punish the slash variable, if we set C with too small value, it will allow a large freedom for many data points are very close to the decision boundary, i.e. we allow large empirical error on training set. Conversely, if we set C with a too large value, we allow very small freedom if $w^T * x < 1$, i.e. allow small empirical risk on training set, causing the model overfitting. Which will make our decision boundary tight, but more prone for overfitting the data. Therefore, the best C should be properly tuned and might be in the middle(neither too large or too small), the best value of C is where we find the best bias-variance tradeoff.

4. Running time of SVM with respect to C. I don't observe any trend of running time with respect to parameter C.

5. Comparison for SVM with Logistic Regression. for this specific classification task, the logistic regression outperforms SVM with respect to the best f1-score they achieve. However, we can't generalize that logistic regression will beat SVM for any tasks. Both of them are very popular classifiers and have many applications. Here, we only use the linear kernel for SVM. We will definitely exploit more power of SVM for using non-linear kernels in order to get more complicated model and decision boundary. It's also very efficient to use kernel as a method to project original data to higher dimensional space, while for logistic regression, we might need to manually project data to new feature space, which introduces more computations. Logistic regression, on the other hand, is great for its nice probabilistic interpretation, since it's output is a probability value that conveys the confidence, it's very common to apply this technique to application like predicting the CTR in online advertising. Further, by using different priors, we can get different regularization terms in the optimization target function. For example, by using l1-norm (lasso) regularization, we can incorporate sparsity into the solution.

6. Macro average - micro average. One last thing I observe that is not relevant with specific classifier is that micro-average tends to be higher than the macro average. The reason for this might be that, micro average will be more biased for the class that has more data, while macro average treats all classes the same. And since by using more training data, we are more likely to get a good model. Therefore, those class with more data tends to show better performance than those with fewer data. Therefore, micro-average tends to have a higher value than macro average. (Note that here I use an assumption that the distribution of labels in the training set is about the same for the label distribution in the test set.)

4. The software implementation (15%)

1. Logistic Regression

I use java to implement the logistic regression. I started to think about using Matlab as programming language because it's a natural fit for numerical computation. The reason why I discard Matlab is that I have to preprocess the data in the given format, and Matlab really sucks in doing things other than numerical computation.

Since Java doesn't support storage of a matrix, we need another way to represent the sample and feature. The way I handle it is to use a class called instance to represent a sample, it has a field that represents the label of instance, and a hash map to store the features. For each non-zero feature, it's stored as a feature_id value pair in the hash map. Since there are lots of empty cell in the given corpus, we can save a lot of memory with this sparse representation of features.

The implementation of training and testing for logistic regression is trivial, since most of the method is merely numerical computations. The optimization method I adopted in this implementation is stochastic gradient descent, which is an online learning method that is usually a better choice than batch gradient descent when training set is large. The speed of original implementation is pretty slow, which needs 10 minutes to train 17 models. To speed up the process, I change the implementation of weight updating process. Originally, I scan through the weight array, and check the feature hash map. Since the feature is sparse, a lot of computation is wasted on the hash map look up. For the current version of implementation, I first iterate through the weights array, and update each weight with respect to the derivative of the regularization term. Then iterate through the key set of the feature hash map and update the corresponding weights with the derivative of likelihood term.

Since logistic regression is a binary classifier,we need to design appropriate strategy to apply it to the problem of multi-class classification. The way I did it is to implement a multi-class wrapper using one-vs-rest strategy. For the k-class classification problem, I train k binary logistic regression models, where ith model discriminate between ith class and all other classes. The prediction uses winner-takes-all strategy which utilizes

the probability output of the logistic regression, where the predicted label is the class that the sub-classifier has the most confidence.

2. SVM

Since sum light already implements the binary SVM classifier, what we need to do is to write a multi-class classification wrapper. There are mainly two issues regarding the implementation of this multi-class SVM:

1. Multi-class strategy. I use the same multi-class classification strategy as the one I use for logistic regression, i.e. one-vs-rest strategy. To do this, I need to replicate the training set 17 times, and modify their labels to fit the binary classifier.

2. Calling SVM-light from program. In order to call the external program (i.e. svm_learn, svm_classifiy) from the program, I use the call method in the subprocess module in python. But this is not very convenient, since the input and output for svm_learn and svm_classify is file on disk, not from memory. I have to partition the data, store the model, predictions in files, and re-read them to process in the program. A better choice might be to use the SVM Light's python interface, so I can handle everything purely in memory after readming the original training set and test set files.