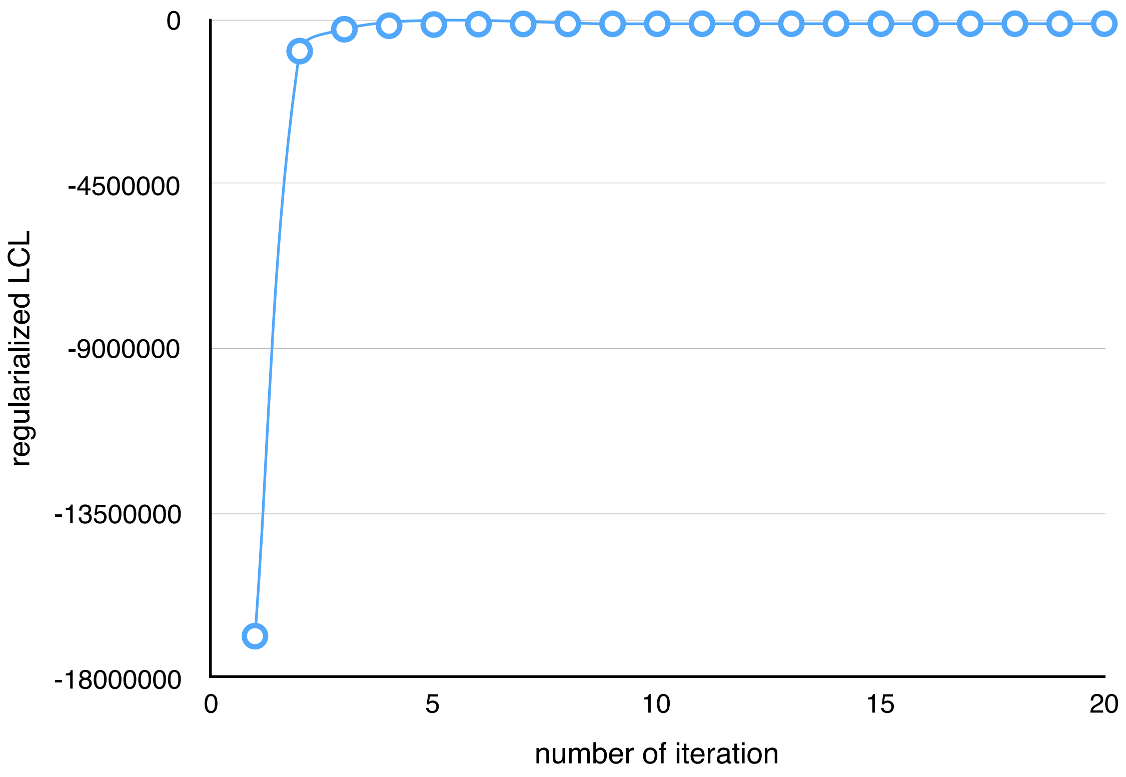
Sunday, October 8, 2017

Report3

Logistic Regression Using Stochastic Gradient Descent

Yao Li—yaol3

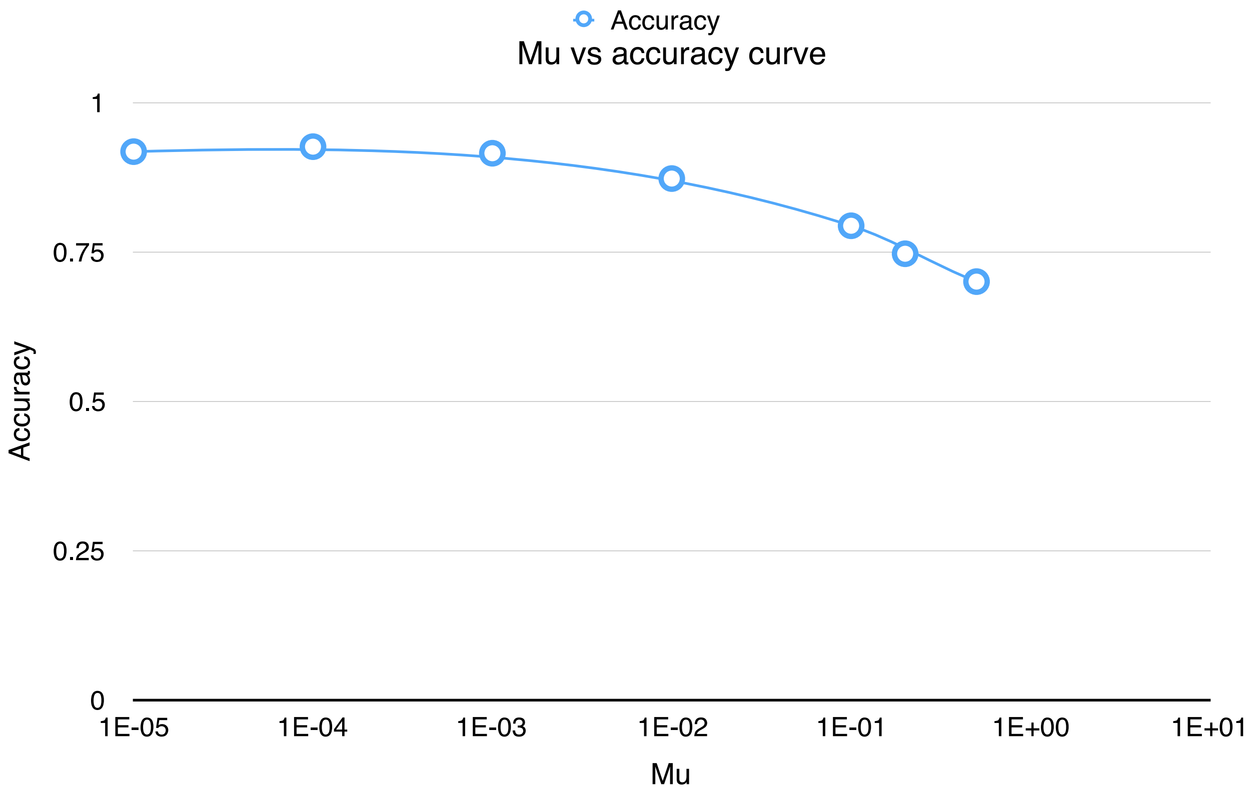
1.



2.

Here I how I calculate the accuracy:

Every classifier will give true to corresponding label if the probability is > 0.5, we will have a label set for each example. Then if any predicted label is one of the true label, we will say the prediction for this example is correct, otherwise it is regarded as incorrect. Accuracy is given by the correct number divided by number of examples.

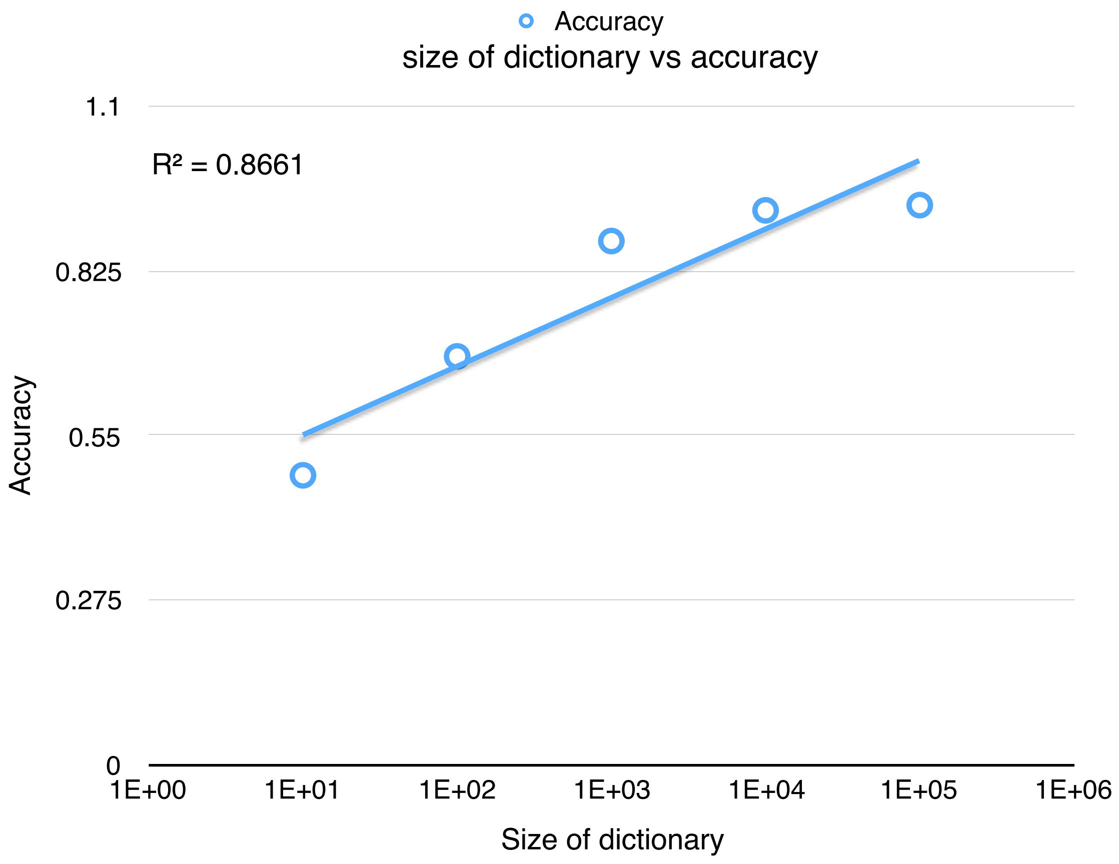


And the curve for problem 2 is given below.

Note if µ is 0, accuracy is also about 0.91, then with the increase of the µ, the accuracy increase slowly, but when µ is larger than 1e-4, it begins to decrease quickly, a larger µ might put too much weight on the regularization and therefore makes the learned parameters fluctuate too much, which makes the learning useless. The optimal value I found is 1e-4.

3

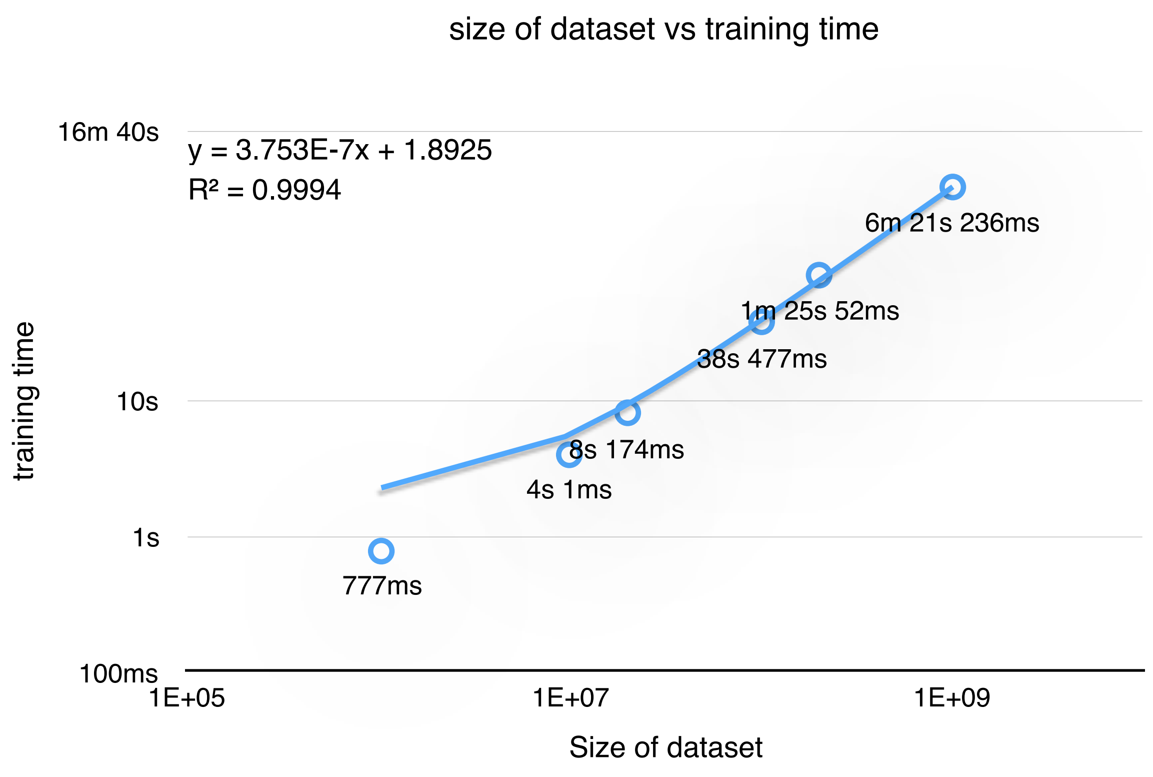
2



As we can see above, the accuracy has a trend to increase with the increase of log(dictionary size), but such increase is not significant between 10000 and 100000, which implies that a further increase of the dictionary size will not help much. While from relative small size(e.g. 100, to 10000), it helps a lot. And for the case it is too small, the model has an accuracy that is too low to use. This tells us the size of dictionary can be optimized to balance the training speed and accuracy, make dictionary too large will not help much while it still can't be too small to overly simplify the model.

4.

The dictionary size chosen is 100000, and the μ is 1e-4 for all size of data.



From the above plot, we can see the algorithm scales well, as the regressed R^2 is very close to 1.

Since the size of dictionary is fixed, basically if the average length of the text remains will not change a lot, our lazy update of the parameter will be finished in O(n) time as we only update the words appear in the text and the calculation of the prediction(p) will not vary with the variation of the dataset size.

5.

Basically, what we really need to change will be the derivatives of the regularizer. The lazy regularized SGD optimization algorithm for Lasso will basically follow the same step.

For the function $f = |\beta_j|$

its derivative is f' = 1, if $\beta_j > 0$, else if $\beta_j < 0$, f' = -1, it does not have derivative on 0.

We separate the regularization and gradient descent into two part, while the gradient descent will remain the same, the regularization update will be :

```
for (int i = 0; i < k - A[j]; i++) {
    B[j] -= sign[B[j]] · λμ
}
```

Given λ > 0 and μ > 0, to improve the efficiency, we can change it to be

```
for (int i = 0; i < k - A[j]; i+= Math.min(Math.max(Math.abs(B[j] / λμ), 1), k - A[j] - i)) {
    B[j] -= sign[B[j]] · λμ
}
```

This function will ensure that we update at least one time, and up to Math.abs(B[j] / λμ

) times, and the total time is bounded by k - A[j].

This way, we would largely decrease the runtime for a large k-A[j].

However, if λμ is too large, it will still not be efficient since the sign of B[j] will frequently flip. But one observation is that, if the update number(k - A[j]) is large, it is very likely that the parameter B[j] is also a large number, since a large B[j] is usually caused by a infrequent feature. And if the B[j] has a small absolute value, it is very likely to be updated frequently. Therefore, out conclusion is, this algorithm will work fine for both case.

Here is the algorithm:

1. Let k = 0, and let A and B be empty hashtables. A will record the value of k last time B[j] was updated.

```
2. Fort=1,...,T
    • For each example $x_i$, $y_i$: – Let k = k + 1
    – For each non-zero feature of $x_i$ with index j and value $x_j$:
        ∗ If j is not in B, set B[j] = 0.
        ∗ If j is not in A, set A[j] = 0.
        ∗ Simulate the “regularization” updates that would have been
        performed.
        for the k − A[j] examples since the last time a non-zero $x_j$ was
        encountered by setting
        ∗ Update B[j] using the function described.
        ∗ SetB[j]=B[j]+$\lambda(y-p)x_j$ ∗ SetA[j]=k
3. For each parameter $\beta_1, \ldots, \beta_d$,
Update B[j] using the function described.
4. Output the parameters $\beta_1, \ldots, \beta_d$.
```

6.

For batch style, our gradient will be the accumulation of a batch of texts divided by the batch size. Therefore, before we perform updation on our parameter, we need to aggregate the derivative for each classifier.And here, we can make calculation of p efficient enough by calculate it first, since we update the B after we finished processing one batch of text. And the real update for the gradient part will be,

$$\lambda(\mathrm{sum}(y)-p \cdot n)x_i/n$$

n is the size of the batch, sum(y) is the aggregation of the real label(0, 1).

Then the process will be almost the same as the on-line style.

```
1. Let $k = 0$, and let A and B be empty hashtables. A will record the value of
   k last time B[j] was updated.

2. Fort=1,...,T

   Let $n = 200$(batch size)

   • Calculate p.

   • Let $k = k + 1$
   • For each batch($i = 0$; $i < n$; i++): $x_i$, $y_i$:

         * accumulate the y value for each classifier.
   – For each non-zero feature of $x_i$ with index j and value $x_j$:

         ∗ If j is not in B, set B[j]=0.

         ∗ If j is not in A, set A[j]=0.

         ∗ Simulate the “regularization” updates that would have been
         performed
         for the $k − A[j]$ examples since the last time a non-zero $x_j$ was
         encountered by setting
         $B[j] = B[j] \cdot (1 - 2\lambda\mu)^{k-A[j]}$

         ∗ SetB[j]=$B[j]+\lambda/n \cdot (\text{sum}(y)-np)x_j$

         ∗ SetA[j]=k

3. For each parameter $\beta_1, \ldots, \beta_d$, set
$B[j] = B[j] \cdot (1 - 2\lambda\mu)^{k-A[j]}$

4. Output the parameters $\beta_1, \ldots, \beta_d$.
```

7.

- Did you receive any help whatsoever from anyone in solving this assignment?

No

- Did you give any help whatsoever to anyone in solving this assignment?

No