Hannah Provenza

StatNLP

Programming Assignment 2 - MaxEnt Classifier with Stochastic Gradient Descent

21 October 2016

Experiment 1

x-axis: number of instances trained on

y-axis: accuracy

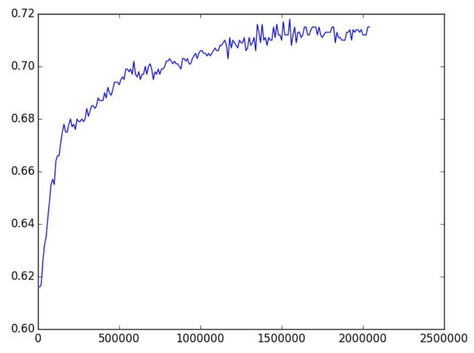
Learning rate: .0001

Training set size: 10000

Test set size: 1000

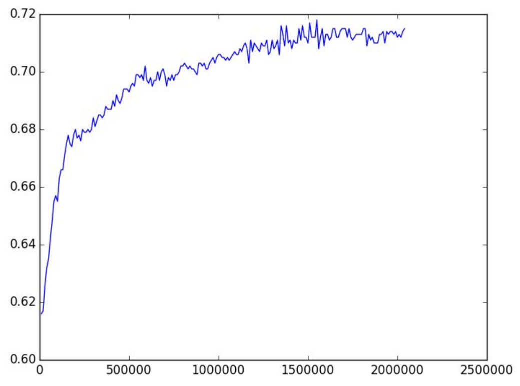
Batch size = 1

Datapoint taken after each epoch.



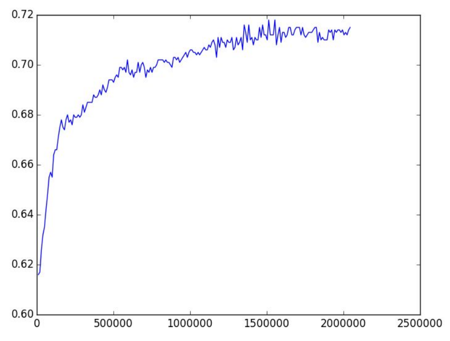
Batch size = 10

Datapoint taken after each epoch.



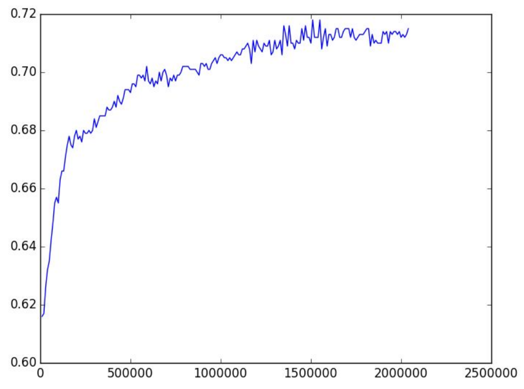
Batch size = 30

Datapoint taken after each epoch.



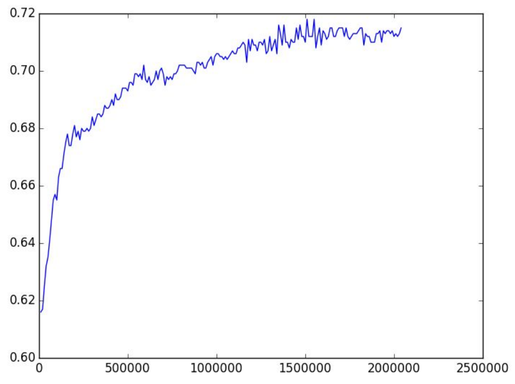
Batch size = 50

Datapoint taken after each epoch.



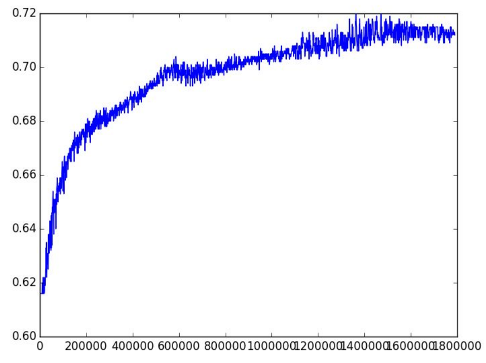
Batch size = 100

Datapoint taken after each epoch.



Batch size = 1000

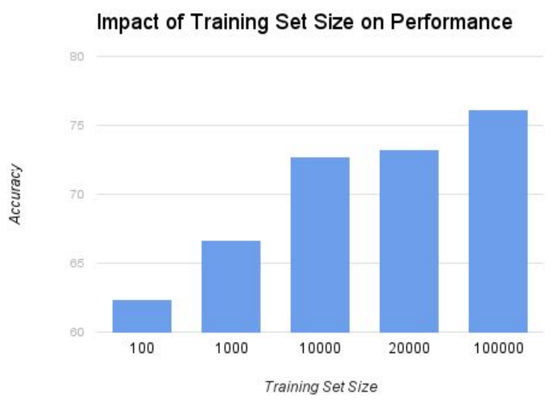
Data point taken after each minibatch.



Hannah Provenza
StatNLP
Programming Assignment 2 - MaxEnt Classifier with Stochastic Gradient Descent
21 October 2016

It's really interesting to me that the graphs look so similar - *really* similar - despite the different batch size, even with little peaks and stalls and dips in the same places. It seems clear to me that minibatch produces nearly identical results to training each gradient on the full dataset. However, the way the question was phrased here led me to believe that the computational expense of the minibatches was a sacrifice to achieve some other gain, and I'm not sure I understand what the benefit is. Minibatching requires the computationally expensive gradient calculation *more* often than the alternative, and the results are indistinguishable.

Experiment 2
This second set of experiments were all conducted with batch size 100 and learning rate .0001. Constant test and dev set size of 1000 and 3000 instances respectively



My results show that there is definitely a performance improvement correlating with size. However, this correlation also shows the effect of the law of diminishing returns. A training set of only 100 instances was enough to achieve more than 60% accuracy; increasing that training set 1000-fold resulted in less than a 15% improvement in performance..

Feature Engineering
In training, the total feature set is placed in a dictionary; each feature has a unique ID number. After all ID numbers are generated, feature vectors are created; each is essentially the same as the feature list but replaced with the ID number. The feature set is all features except those words which are in the NLTK English stopwords list.

Gradient Descent
The initial *negative_log_likelihood* is set to the maximum integer; it is our goal to minimize this value. The main loop of the gradient descent algorithm is as follows: The *train_instances* dataset list is split into a list of lists, with each inner list containing *batch_size* number of instances. For each minibatch, the *gradient* is calculated with *compute_gradient()*, and the product of *gradient* and *learning_rate* is added to the model. After the loop over the minibatches is complete, *compute_negative_log_likelihood()* is calculated on *dev_instances.*
If the *new_negative_log_likelihood* is less than the old one, it replaces the old one, *no_change_count* is set to 0, and the main loop repeats. Otherwise, *no_change_count* is incremented and the main loop repeats. If no_change_count reaches 5, meaning that 5 consecutive iterations of the main loop did not generate a lower log likelihood, we conclude that the model has converged; the permanent model is the last one generated before the 5 iterations of the loop that did not improve.