CSE 569: Project 1.6: Battle of the Momemtums

Anurag Agrawal, ID: 1207687399
and Venkata Vamsikrishna Meduri, ID:1208577223

Abstract—Two-class classification is a well-studied problem in the machine learning literature. In this work, we pick the basic binary classifier flavor of the logistic regression machine and study the properties of its momentum. We implement the negative log likelihood loss and hinge-loss methods into logistic regression to form the primary baselines. Subsequently, we learn the gradient descent learning procedure using Polyak's classical momentum [1] and Nesterov's accelerated gradient [2] [3]. We compare all these variants of logistic regression among themselves to understand the properties of the momentum and how they contribute to convergence by an iterative observation of the parameters learnt such as weights and losses. We also study how Polyak and Nesterov compare to using L2-regularization w.r.t. the speed and stability of learning. We demonstrate the effectiveness of each of the optimization techniques through a case-by-case experimental analysis for the various samples drawn from the real-world datasets for Entity Resolution and the synthetic datasets that we craft manually.

Index Terms—logistic regression, Polyak, Nesterov, momentum, regularization, classification, resolution

1 Introduction

Logistic regression (LR) uses the logistic function to compute the probability with which a data point expressed as a feature vector can be classified as belonging to a particular class. As LR is predominantly used for the binary classification problem (although LR can as well be used for multi-classification), the probabilities of a feature vector belonging to either of the classes are compared and the larger probability among them determines the classification output.

For instance, let a feature vector to be classified be $\boldsymbol{D} = [d_1, d_2, \ldots, d_n]^T$. The logistic (or the sigmoid) function can be written as

$$h(x) = \frac{1}{1+e^{-x}} \tag{1}$$

We can notice that the feature vector $\boldsymbol{D}$ needs to be changed to a scalar value x to be substituted in Equation 1. The computation of the scalar, x, can be done by a weighted combination of all the features in $\boldsymbol{D}$ which can be written as $x = w_0 + \sum_{i=1}^{n} w_i * d_i$. The weights ranging from w_0 to w_n need to be learnt from the training data in order to compute the scalar, x for each of the test data and thereby classify the test feature vectors.

One of the well-known methods to learn the weights (or in general the parameters needed) is to express the learning process as an optimization function that can be approximated to the target value iteratively. In each iteration, we can learn the parameters incrementally thereby refining them through a step-by-step process. An example to such approaches is the famous Stochastic Gradient Descent (SGD) that learns the parameters by descending gradually on the gradient of a curve which is formed out of the cost function to be minimized. An important thing to note here is that SGD expresses the optimization as a minimization though it is not necessarily the case in LR. This is because, in LR, we want to maximize the joint probability of the positively and negatively classified training vectors thereby ensuring that the classification outcome matches with the expected labeling provided in the training data. There are two approaches that formulate this maximization process as a function f_{opt} that can be minimized:

- Negative log likelihood based approach computes the joint probability (likelihood) on all the i.i.d training samples in the data, applies log on the likelihood, and minimizes its negation.
- Hinge loss based solution minimizes the error associated with mis-prediction on the training data which is also called as loss or misclassification penalty.

The basic methodology of SGD using one of the two cost functions listed above involves guessing an initial set of the $n+1$ weights, w_0, w_1,..,w_n, and iteratively updating them such that the value of the cost function decreases in each iteration by using the updated set of weights from the previous iteration. At any given iteration, the weight corresponding to the ith feature is updated by using $w_{i+1} = w_i - (\alpha * \frac{\partial f_{opt}}{\partial w_i})$. As we can notice from the expression, the weight at the $i+1$th iteration is computed by subtracting a finite multiple (or step) of the gradient of the function to be minimized. The step is dependent on the learning rate, α which determines the speed of convergence in the number of iterations as well as the local minimum that is obtained at convergence.

We borrow two approaches from the literature to modify the weight updation step in each iteration of SGD. These approaches are termed momentums and we compare two such solutions - Polyak's classical momentum [1] and Nesterov's momentum [2] which is the core focus of this work.

2 Optimization functions

As we described in Section 1, we use two optimization functions to be minimized in SGD, i.e., negative log likelihood and hinge loss for an LR classification machine which we detail out here.

2.1 Negative log likelihood

Assuming that all the sample vectors drawn from the training data $\boldsymbol{D}$ are independent and identically distributed (i.i.d), we can compute the joint probability of each of the training samples belonging to the same class as determined by the class label which can be maximized.

Let us assume that there are P positively labeled points belonging to class 1 and N negatively labeled points belonging to class -1. Corresponding to this labeling, we can compute the probabilities from the logistic function in Equation 1 as $\prod_P h(x) \prod_N 1 - h(x)$ maximizing which is equivalent to minimizing the negation of it. To convert the product into a sum that is friendly to differentiate for gradient computation, we get $-\sum_P log(h(x)) - \sum_N log(1-h(x))$ which stands for f_{opt} as the minimization function to be optimized using SGD as mentioned in Section 1.

2.2 Hinge loss

The hinge loss is computed as the penalty of misclassification that we want to minimize. We find the weights which minimize the loss function that is written as follows:

$$w^* = \arg\min_w loss_{hinge} = \arg\min_w \sum_i max\{0, 1 - y_i h(x)\} \tag{2}$$

We know that h(x) from Equation 1 is dependent on $x = w_0 + \sum_{i=1}^n w_i * d_i$ that helps in inferring the value of w_0 to w_n by computing the gradient on $loss_{hinge}$ using partial derivatives w.r.t. each of the weight variables. The basic idea of Equation 2 is to penalize when the sign of y_i (such that $\forall y_i \in \{-1, +1\}$), the true label, is different from the estimation by the sigmoid function $h(x)$. Hence the derivative of $loss_{hinge}$ w.r.t. the ith weight parameter in LR , w_i is written as:

$$\frac{\partial loss_{hinge}}{\partial w_i} = \begin{cases} 0 & y_i h(x) \geq 1 \\ -y_i \frac{\partial h(x)}{\partial w_i} & y_i h(x) < 1 \end{cases} \tag{3}$$

The gradient w.r.t. hinge loss is taking as the summation of the partial gradients computed over all the weights, w_i, as per Equation 3.

$y_i h(x) < 1$ implies that the training vector x has been misclassified which takes to the second case in Equation 3. A non-zero gradient is computed only for all the misclassified training data and the accompanying updation of weight vectors in the SGD algorithm. As against the standard maximum margin machine hinge loss using $h(x) = w_0 + \sum_{i=1}^n w_i * d_i$, we use h(x) from Equation 1 which is a sigmoid function. Hence, $\frac{\partial h(x)}{\partial w_i}$ can be computed as follows:

$$\frac{\partial h(x)}{\partial w_i} = h(x) * (1 - h(x)) * x_i \tag{4}$$

3 L2 Regularization

As described in Section 1, the weight updation step at any given iteration in SGD is done as follows:

$$w_{i+1} = w_i - (\alpha * \frac{\partial f_{opt}}{\partial w_i}) \tag{5}$$

α in Equation 5 refers to the learning rate and f_{opt} refers to the possible optimization functions that can be used. Two such example optimization functions that can be used for logistic regression are negative log likelihood and hinge loss that have been described in Section 2. However it is crucial to note that by the virtue of using SGD with an optimization function (minimization function in this context), we necessarily need not reach a stable optimum quickly. Stability is characterized by a solution that does not overfit to the training data and is sufficiently generic. It also implies that the performance on the test data is stable and does not change significantly with minor changes to the training set. The speed of convergence also needs to be high which is dependent on the learning rate α and the gradient of f_{opt}.

With an α that is greedily set to a high value in order to reach the convergence quickly, an oscillation phenomenon is observed as the update can be so huge that it will cross-over the solution region. This is because the weight will be updated back and forth as the step size in each iteration of the gradient descent is too big. This can be fixed by using a regularization parameter which will reduce the update to the weight vector in each SGD iteration. Though it may seem to slow down the convergence, it prevents oscillation and reaps a benefit in the longer run for convergence though the local step seems to be small.

The introduction of an L2 regularization parameter into the minimization step of an arbitrary optimization function can be written as follows:

$$w_{opt} = argmin_w(\sum_{x \in T} f_{opt}(w, x) - \lambda/2 * ||w||^2) \tag{6}$$

Differentiating the minimization term in Equation 6 gives us the following:

$$\partial f_{opt}/\partial w - \lambda * ||w|| \tag{7}$$

Substituting the gradient of the minimization term obtained from Equation 7 into the weight update step, we get:

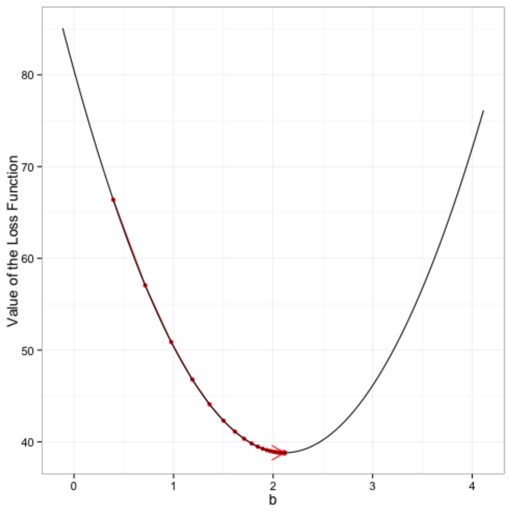
$$w_{i+1} = w_i - \partial f_{opt}/\partial w + \lambda * ||w|| \tag{8}$$

From the weight update step in Equation 8, it is evident that the regularization parameter is the third component $\lambda * ||w||$ which slows down the descent step of moving w_i towards the optimum. Though it seems to slow down the convergence superficially, this indeed helps in a quicker convergence because of avoiding any impending oscillations thus making L2 regularized SGD a robust baseline to compare stability and speed of convergence against.

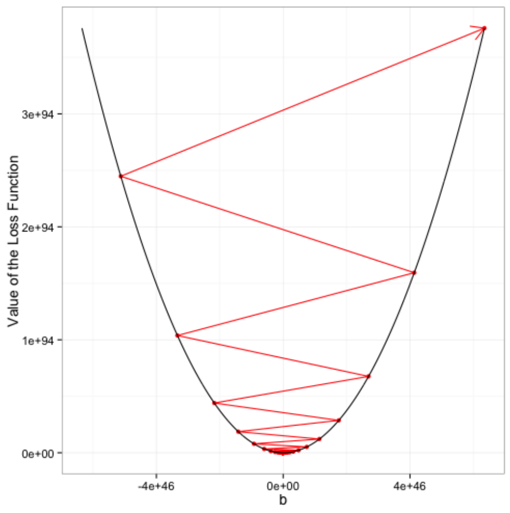
4 Momentums

It is not uncommon that second order methods are preferred in the gradient descent step especially when the curve is changing its slope at each step. This is because, second order methods do not persist weights and rather update them. However, when the curvature is smooth, second order methods are not ideal because they do not realize the value in persisting the gradient so far.

The traditional first order based weight update methods also need a slight modification to be a fit to apply SGD on



(a) Example of a persistent parameter updation



(b) An oscillating gradient descent approach

smooth curves. They need to take a weighted combination of the gradient seen so far (or in some sense the one from the previous iteration) and the current gradient. This is because, in the previous iteration, an exceptionally large or a miniscule gradient might have been obtained in the light of which the current update might be seen as a perturbation and vice-versa. A momentum coefficient term denoted by μ denotes the weight assigned to the previous update.

We describe the methods of momentums employed in SGD in the weight updation step. Momentum according to theoretical physics indicates the product of the mass and velocity and an object persists its velocity and in turn, its momentum, unless an external force is applied on it. Though the term does not borrow the definition per se, it indicates that the rate of change of the weight vector obtained so far or the gradient needs to be persisted as shown in Figure 1a unless there is a radically different gradient obtained in the current iteration. The notion of momentum is discussed hereafter in the context of either of the gradient functions discussed in Section 2. As described in Section 1, $w_{i+1} = w_i - (\alpha * \frac{\partial f_{opt}}{\partial w_i})$. This is modified according to the various kinds of momentums.

4.1 Polyak's Classical Momentum

Polyak's classical momentum aims at persisting the velocity or the direction and magnitude of change to the weight vector in the existing direction. Hence, the change to the weight parameter, w_t, corresponding to the tth SGD iteration, can be expressed as a weighted combination of the velocity seen so far v_t, and the gradient of f_{opt} at w_t, to obtain the change to be made, v_{t+1}.

$$v_{t+1} = \mu v_t - \alpha \frac{\partial f_{opt}(w_t)}{\partial w} \tag{9}$$

$$w_{t+1} = w_t + v_{t+1} \tag{10}$$

However, Polyak's momentum is not devoid of limitations. The momentum coefficient μ can impede convergence in cases where it is greedily set to a high value. The details of how setting a parameter to a high value results in crossover of the solution region and subsequent back and forth oscillation has also been discussed in Section 3. This can result in oscillations as shown in Figure 1b.

4.2 Nesterov's Accelerated Gradient (NAG)

NAG aims at addressing the demerits of Polyak's momentum caused by large values of the momentum coefficient μ. It makes a subtle change to the Polyak's momentum by computing the gradient of f_{opt} at $w_t + \mu v_t$ instead of w_t keeping the remaining part of the update equation the same as the classical momentum in Equation 9. Thus it can be formulated as follows:

$$v_{t+1} = \mu v_t - \alpha \frac{\partial f_{opt}(w_t + \mu v_t)}{\partial w} \tag{11}$$

Equation 11 of NAG differs from Equation 9 of Polyak in the aspect that the optimization function over which the gradient is computed takes $w_t + \mu v_t$ in NAG as against w_t in Polyak. This ensures that the change in weight caused by large μ is compensated by the value of the gradient which acts as a regularizing factor in NAG. This can be seen from the fact that $\frac{\partial f_{opt}(w_t+\mu v_t)}{\partial w} \geq \frac{\partial f_{opt}(w_t)}{\partial w}$. Hence, NAG lessens the scope for oscillations as compared to Polyak during the gradient descent phase.

5 Implementation

All the implementation was done in Python 2.7.11 using numPy and a few standard Python libraries on an iMac running 10.11.6 OS X with a 2.8 GHz Intel Core i5 processor and a 16 GB memory. The implementation involved the creation of the datasets and a basic logistic regression implementation with SGD upon which all the optimization function and momentum variants have been developed. A few of the non-trivial challenges involved during the implementation were to handle the oscillation issues associated with the algorithms over the datasets, the numerous iterations that the algorithms were running for, and the identification of suitable configuration parameters for learning rate and convergence rate of SGD.

6 Datasets

We have conducted experiments on two different datasets: a real world Entity Resolution dataset suitable for a binary classification problem and a synthetic dataset crafted by us.

6.1 Amazon-GoogleProducts

We have used an Entity Resolution dataset from Amazon-GoogleProducts [4] http://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution to report our initial results. The dataset contains the product details from Amazon and Google manufactured products and the entity resolution task is to identify tuple pairs referring to the same product from both the individual datasets. The provided dataset contains 1300 matching tuple pairs whereas the negative tuples are filtered from the remaining tuple pairs by choosing non-trivial pairs which look similar enough however referring to different products summing up to 96K tuple pairs. The dataset contains 4 attributes that expand into 116 attributes owing to the 29 similarity functions used to construct the feature vectors.

6.2 Synthetic dataset

We have created a linearly separable and a non-linearly separable synthetic dataset as follows: For each tuple, we ensure that the value range is divided into two fragments: one for each class. If we homogeneously assign the attribute values of a tuple from the fragment corresponding to the intended class, that results in a linearly separable dataset. Contrary to that, when the feature values are randomly assigned, this naturally results in a non-linearly separable dataset as there is no natural correlation between the feature values and the class labels. This was also manually verified subsequent to the dataset generation. We created both the linearly separable and non-linearly separable datasets each with 20,000 tuples and 50 attributes.

7 Experiments

We ran 5-fold Cross Validation (CV) experiments upon all the three datatsets - Synthetic Linearly Separable, Synthetic Non-Linearly Separable and Amazon-GoogleProducts. We first compare log-likelihood with hinge loss and then present the comparison of Polyak and Nesterov upon one of the Logistic Regression (LR) implementations.

7.1 Effect of (Learning Rate, Convergence Threshold)

In this experiment, we chose the synthetic linearly separable dataset and vary the learning rate and convergence threshold for SGD from 10^{-2} to 10^{-6}. We can notice from Figure 2 that the number of iterations grew with decreasing values of these parameters.

7.2 Negative log likelihood Vs Hinge loss

In the case of synthetic linearly separable dataset, we noticed that the effectiveness of both hinge loss and negative log likelihood in terms of F-measure did not vary much. This was owing to the fact that linearly separable data was easy to classify. However, a difference was noticed in the fact that hinge loss was much slower than negative log likelihood in its convergence rate as the cost function varied too slowly for hinge loss. This can be observed because hinge loss does one comparison at a time with the label and the prediction value and also the cost function being formulated as a logarithmic measure for log likelihood makes its convergence faster than the cost function of hinge loss.

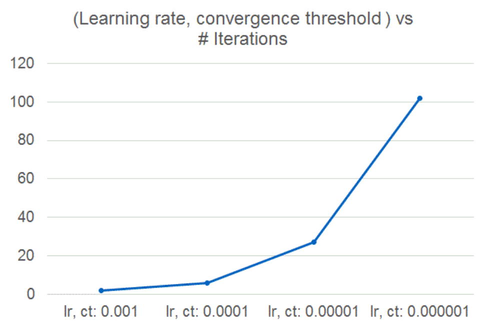
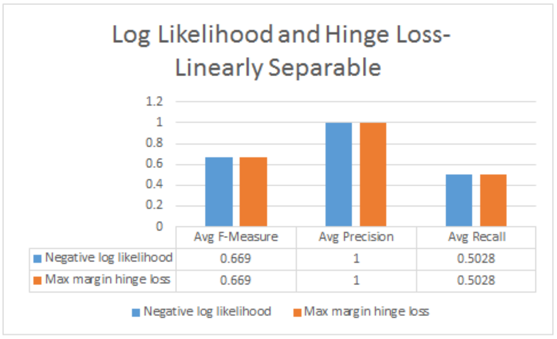
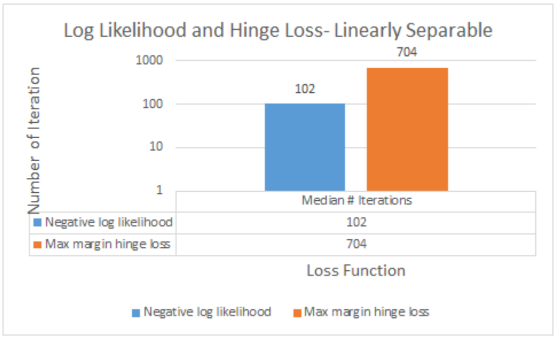


Fig. 2: Effect of (Learning Rate,Convergence Rate)



	Avg F-Measure	Avg Precision	Avg Recall
Negative log likelihood	0.669	1	0.5028
Max margin hinge loss	0.669	1	0.5028

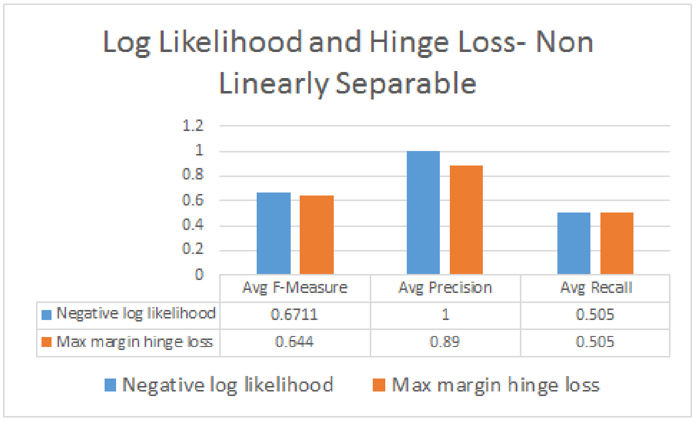
Fig. 3: Loglh Vs hinge: Linearly separable



	Median # Iterations
Negative log likelihood	102
Max margin hinge loss	704

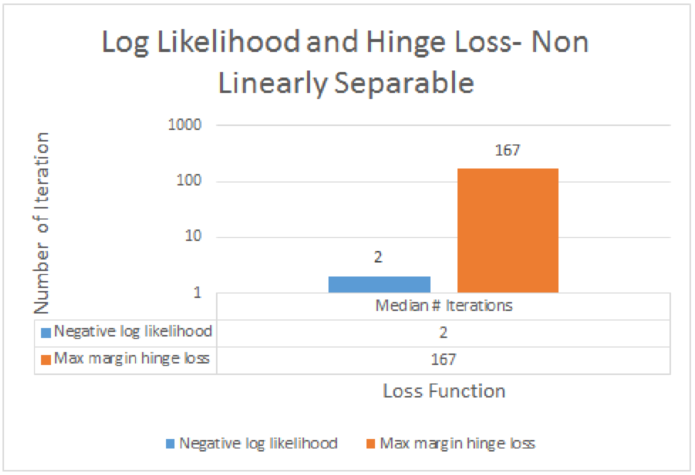
Fig. 4: Loglh Vs hinge: Linearly separable

In the case of non-linearly separable dataset, we noticed that with extremely low values for learning rate and convergence threshold, hinge loss did not even converge after 8000 iterations of SGD. Hence, we lowered the convergence threshold to 0.01 while keeping the learning rate at 10^{-6} to obtain the results in Figures 5a and 5b. Our observations both on the non-linearly separable and the Amazon-GoogleProducts datasets show that hinge loss is inferior to negative log likelihood not only in terms of convergence rate but also F-measure.



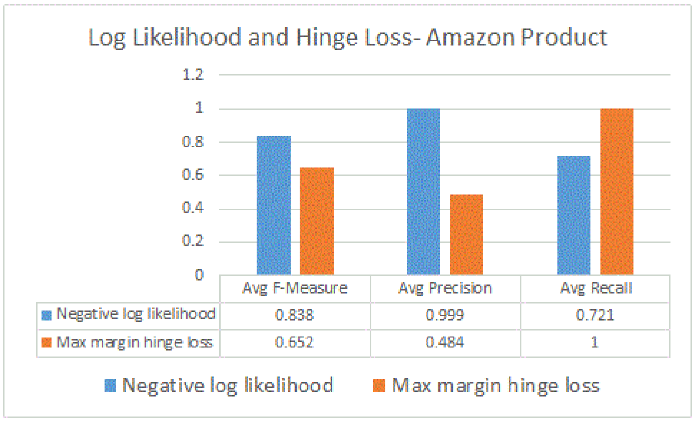
	Avg F-Measure	Avg Precision	Avg Recall
🟦 Negative log likelihood	0.6711	1	0.505
🟧 Max margin hinge loss	0.644	0.89	0.505

(a) Loglh Vs hinge: Non-Linearly separable



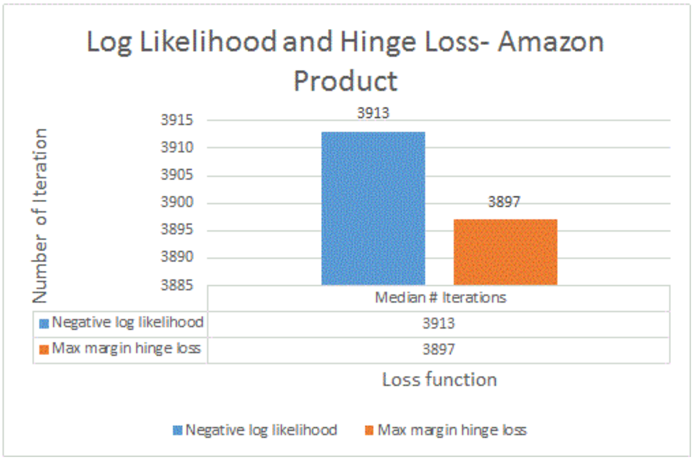
	Median # Iterations
🟦 Negative log likelihood	2
🟧 Max margin hinge loss	167

(b) Loglh Vs hinge: Non-Linearly separable



	Avg F-Measure	Avg Precision	Avg Recall
🟦 Negative log likelihood	0.838	0.999	0.721
🟧 Max margin hinge loss	0.652	0.484	1

(a) Loglh Vs hinge: Amazon-GoogleProducts

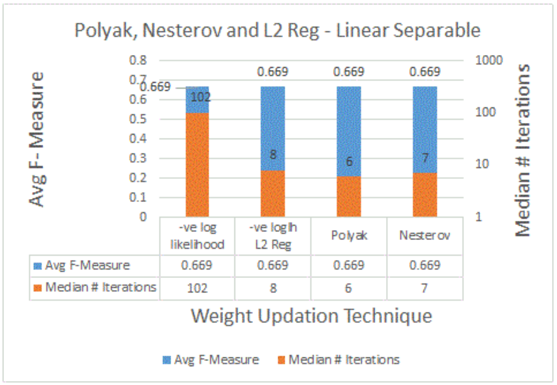


	Median # Iterations
🟦 Negative log likelihood	3913
🟧 Max margin hinge loss	3897

(b) Loglh Vs hinge: Amazon-GoogleProducts

7.3 Polyak Vs Nesterov Vs L2-regularization

We clearly see from the results for Polyak, Nesterov and L2-regularization that L2-regularization performs better than not using L2-regularization in SGD. It not only outperforms the baseline flavor of negative log likelihood upon F-measure but also in the convergence rate. It is important to note that the experiments were done on both hinge loss and log likelihood but here we representatively provide the results for the momentum variants implemented on log likelihood as the relative performances among the momentum algorithms and the L2-regularization stay the same upon hinge loss as well.



	-ve log likelihood	-ve logh L2 Reg	Polyak	Nesterov
🟦 Avg F-Measure	0.669	0.669	0.669	0.669
🟧 Median # Iterations	102	8	6	7

Fig. 7: Polyak Vs Nesterov Vs L2-reg: Linearly Separable

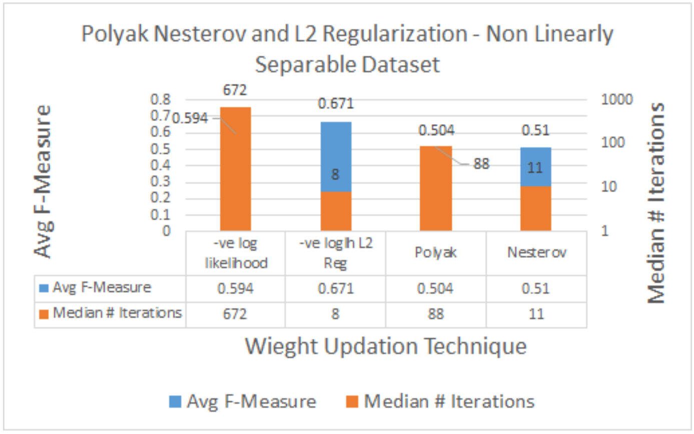
7.4 Effect of Momentum Coefficient

When we varied the momentum coefficient, we expected the iterations always to increase and slow down the convergence rate. However, this was not the case. Rather, the behavior of the momentum coefficient was more like a saddle where the iterations went on a rise and then fell down. The reason for this to happen is that we always take a weighted combination of the velocity or the gradient both from the previous iteration and the current iteration. Hence, the mixture may indeed produce an ideal behavior only at certain values of μ and not necessarily for all of them. In fact for a value of 1.0 for μ, we obtained ideal behavior of minimum iterations.

8 Discussion

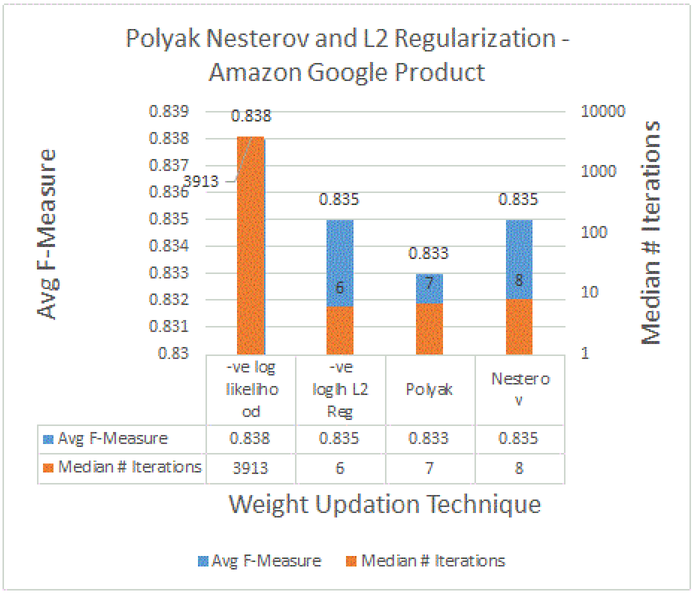
Following are the insights that we obtained from the experiments:

- Hinge loss is prone to more iterations to converge in the non-linearly separable case



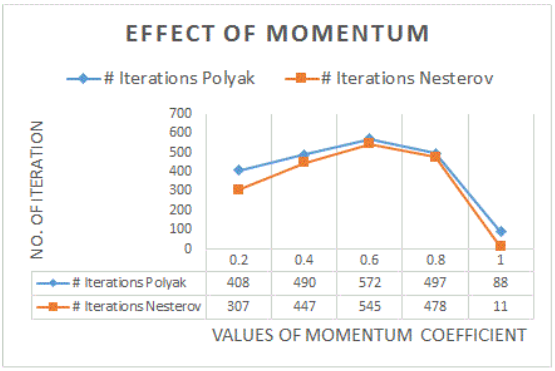
	-ve log likelihood	-ve log lh L2 Reg	Polyak	Nesterov
Avg F-Measure	0.594	0.671	0.504	0.51
Median # Iterations	672	8	88	11

(a) Polyak Vs Nesterov Vs L2-reg: Non-linearly Separable



	-ve log likelihood	-ve loglh L2 Reg	Polyak	Nesterov
Avg F-Measure	0.838	0.835	0.833	0.835
Median # Iterations	3913	6	7	8

(b) Polyak Vs Nesterov Vs L2-reg: Amazon-GoogleProducts



	0.2	0.4	0.6	0.8	1
# Iterations Polyak	408	490	572	497	88
# Iterations Nesterov	307	447	545	478	11

Fig. 9: Effect of Momentum Coefficient μ

- L2-regularization brings stability by reducing the oscillations to the cost function
- Nesterov typically takes lesser iterations to converge because Polyak indeed oscillates more on non-separable dataset
- Polyak takes lesser iterations than Nesterov upon linearly separable data because it is too easy to classify and leads to no oscillations
- Momentum coefficient variation behaves more like a saddle than showing up monotonicity

9 Conclusion and Future Work

We compared two variants of optimization functions - negative log-likelihood and hinge loss for Logistic Regression which drew us to the conclusion that hinge loss takes longer to converge with lesser F-measures than negative log likelihood. Nesterov and Polyak momentum implementation showed that both of them perform competitively well as L2-regression. However, Nesterov converges faster and produces superior F-measures than Polyak in the non-linearly separable synthetic datasets. When the dataset is linearly separable, Polyak performs better than Nesterov. However, in either case, the importance of momentum stands out in a prominent manner. Possible future extensions to this work can include comparison with Hessian based approaches.

References

[1] B. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[2] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," in *Soviet Mathematics Doklady*, vol. 27, no. 2, 1983, pp. 372–376.

[3] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, 2013, pp. 1139–1147. [Online]. Available: http://jmlr.org/proceedings/papers/v28/sutskever13.html

[4] H. Köpcke, A. Thor, and E. Rahm, "Evaluation of entity resolution approaches on real-world match problems," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 484–493, 2010.