**Part I. Least-Squared Linear Regression**

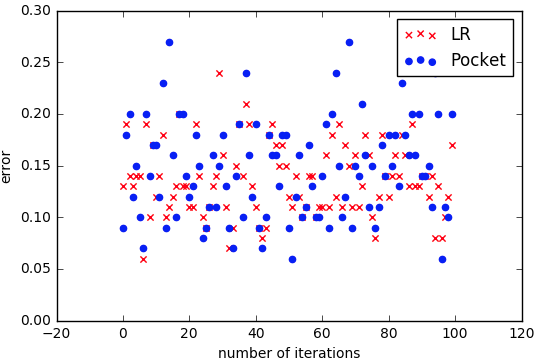
Theory:

We define the line that best fits the data by trying to minimize the distance between each data point and the line we fit.  This is done by minimizing the error function that minimizes the sum of all distances between data points and the fitted line.

The most common error function used is the **least squared optimization** which chooses the parameters in order to minimize the squared difference between the prediction (line) and actual data value summed over all of the data points.

Main Part:

We are given non-separable data points. First, we have to solve classification problem with Pocket Algorithm (almost the same, but we use the best weight for all iterations), where we find a line that separates out the classes so that they can be distinguished. Then we solve Linear Regression problem, where we fit a line to the data. Here we apply pseudo-inverse technique to get the analytical answers.



Mean for E(w\_lin:) 0.1359

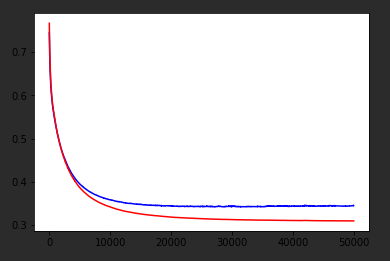
Mean for E(w\_pocket:) 0.1462

So we get here that Linear Regression performs slightly better than Pocket Algorithm. This is may be due to pocket algorithm can 'stuck' in some cases where slight modification gives worse results and large modification is optimal. And this is due to theoretical weakness of Pocket PLA, depite the fact the gradient in random direction guarantees improvement in average as in SGD but gradient in direction of the point which is misclassified can be 'bad'.

**Part 2**

We tried to implement Logistic Regression using numpy vectorization for performance optimization, also in LogisticRegression.py added some extra parameters for SGD as learning rate decay and Monte-Carlo optimization (nothing special, just trying more than one initial weight vector than one).

Following picture illustrates training loss(red) and test\_loss(blue) as function of iterations divided by 10. Thanks for numpy vectorization it takes only 5-10 secs.



As we can see from the plot test error drops with train error but at some point it starts to vibrate. This due to over-fitting in the number of iterations. Since we didn’t use any regularization number of iterations find global minima for train loss, but this weight vector may not be as good as its poor approximation for test loss.

You can see that converging to local minima takes so many iterations(>500,000), because we don’t use weight decay. Algorithm used for model selection needs less iterations and converges faster.

**Cross-validation**

Because of the slicing in python cross validation part is fairly easy to implement. We need also mention that LOOCV is equivalent fold = len(train) cross validation since we get folds equal to number of training examples and thus one sample gets left each time.

We’ve implemented algorithms in OOP style to easier usage in cross validation part and it helped a lot. We didn’t use LOOCV in model selection part since it is really slows down all experiment. So our target algorithms are Logistic Regression and Linear Regression. Metric used is ‘accuracy’ = ‘which portion of labels are correctly classified’. Model selection we use is ‘GridSelection’ which just tries all combinations of hyper-parameters for each algorithm.

Note that if data generated as in exercise 3.2 maximum cross-val score best accuracy possible is 0.9. Experiments show that without regularization best 5-fold val is 0.9 and test result of this algorithm is 0.854, 10-fold best algorithm gives 0.89 val and the test result of this algorithm is 0.855. The best algorithm takes 1000 iterations with weight decay square root of iterations. Linear Regression gives slightly worse result: 0.851 vs. tuned Logistic Regression and similarly in cross-val. This may be due to the fact that Linear Regression is not theoretically built for classification task. We see that cross-val score correlates with test score but due to the low number of training samples (only 100) we have large bias + var.

In this particular problem 5-fold cross val is better due to the low number of samples

Let’s comment hyperparameters:

Iterations:

less than 1000 is usually bad. Best ones are 10000 and more.

Decay:

Setting learning rate to logarithm of iterations decay is usually good. But we can still converge without it with small learning rate.

Optimization:

SGD and batch seem to give similar results with SGD faster.

Regularization:

Regularization gave only slightly better results of 0.856 and 5-fold cross val score 0.9. Best parameters were: 10000 iterations, lr = 0.1, no learning rate decay, optimization = ‘sgd’.

General:

Model performance seems to be non-convex function of hyper-parameters. I hope we will have datasets with larger number of samples for final project since 100 samples really makes learning hard☺

Finally we think that test result 0.856 is good and practical maximum is around 0.87. (if only 100 samples for training)