Assignment 1 - Binary Linear Learning

--Hao Liu (hl1283)

Problem 1:

(a) $\frac{d\,L(\omega, S_0)}{d\,h_\omega(x)} = -\frac{1}{|S_0|}\sum_{(x,y)\in S_0} 2*(y - h_\omega(x)) = 0 \Leftrightarrow h_w(x) = \frac{1}{|S_0|}\sum_{(x,y)\in S_0} y$

and $\frac{d^2\,L(\omega, S_0)}{d\,(h_\omega(x))^2} = \frac{1}{|S_0|}\sum_{(x,y)\in S_0} 2 > 0$, so $L(\omega, S_0)$ is minimized when $h_w(x) = \frac{1}{|S_0|}\sum_{(x,y)\in S_0} y$

(b) $\frac{d\,L(\omega, S_0)}{d\,h_\omega(x)} = -\frac{1}{|S_0|}\left(\sum_{(x,y)\in S_-} 1 + \sum_{(x,y)\in S_+} -1\right) \Leftrightarrow |S_-| = |S_+|$

and $\frac{d^2\,L(\omega, S_0)}{d\,(h_\omega(x))^2} = 0$

(c) $p = 2$

$\rho(\omega, x, y) = f(\omega, x, y) - f(\omega, x, \bar{y}) = 2*f(\omega, x, y) = 2\,y\cdot\omega^T x > 2$

$\Rightarrow y\cdot\omega^T x > 1\,,\ \forall\,(x, y)\in S$

$\Rightarrow L(\omega, S) = \frac{1}{|S|}\sum_{(x,y)\in S}\max\left\{0,\ 1 - y\cdot\omega^T x\right\} = 0$

(d) $\Pr[y_1, y_2, .., y_{|S|} \mid \omega, x_1, .., x_{|S|}] = \prod_{(x_i,y_i)\in S}\Pr[y_i \mid \omega, x_i]$

$\Rightarrow -\log[\Pr[y_1, y_2, .., y_{|S|} \mid \omega, x_1, .., x_{|S|}]] = -\sum_{(x_i,y_i)\in S}\log[\Pr[y_i \mid \omega, x_i]]$

$= \sum_{(x_i,y_i)\in S}\log[1 + \exp(-f(\omega, x, y))] = |S|\,L(w, s)$

$\Rightarrow$ minimizing $L(\omega, S)$ will maximize $\Pr[y_1, y_2, .., y_{|S|} \mid \omega, x_1, .., x_{|S|}]$

Problem 2:

(a)

The Hessian for the regularized quadratic loss is

$H = \frac{2}{|S|}\sum_{(x,y)\in S} x\,x^T + 2\,\lambda\,I$, where I is identity matrix.

(b)

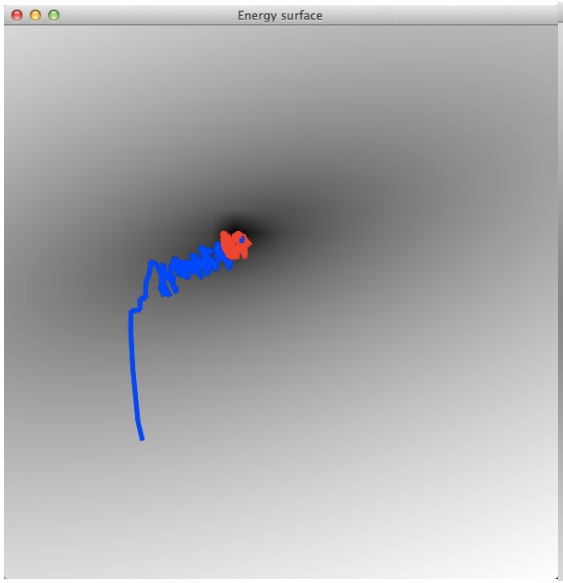
From the experiments, we can know the pre-conditioned learning rate really improve the algorithm.
The problem which the pre-condition tries to solve is that the possible scale difference between features will dominate gradient decent process. Actually the gradient decent will first concertrate on minimizing in the large scale direction (like in the experiments $x_2$direction), and then minimizing anothor direction. If we use pre-condition learning rate, then we will take care more about the small scale direction.

Two big improvements are:

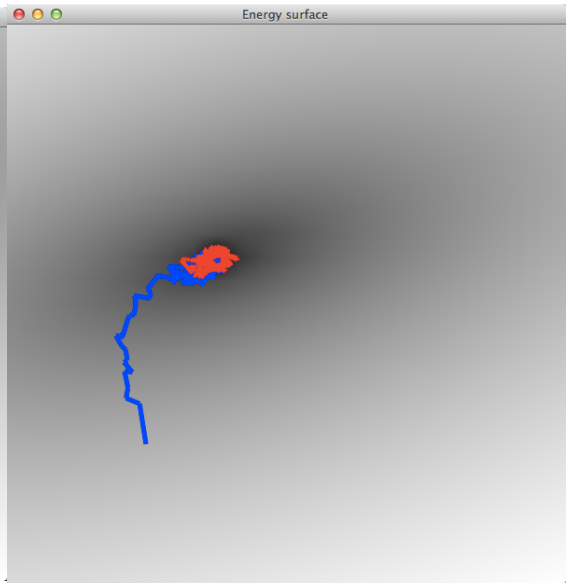
(1) when minimizing the big scale direction, the small scale direction will be changed more that non-precondition.

(2)after the large scale direction is *finished*, we can have a bigger learning rate on small scale diereciton.

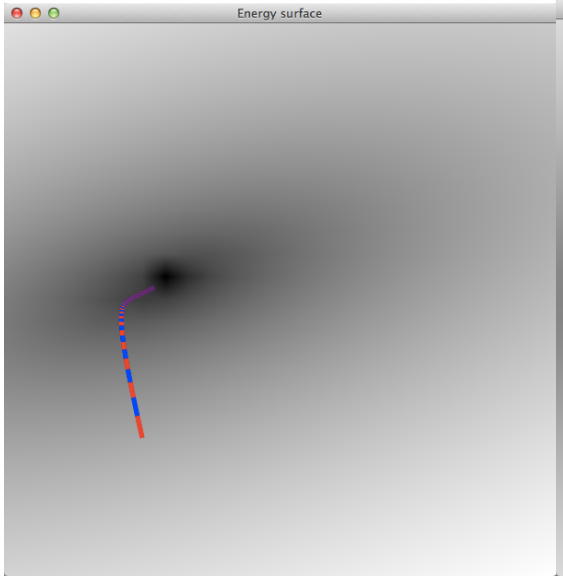
The experiments results are:



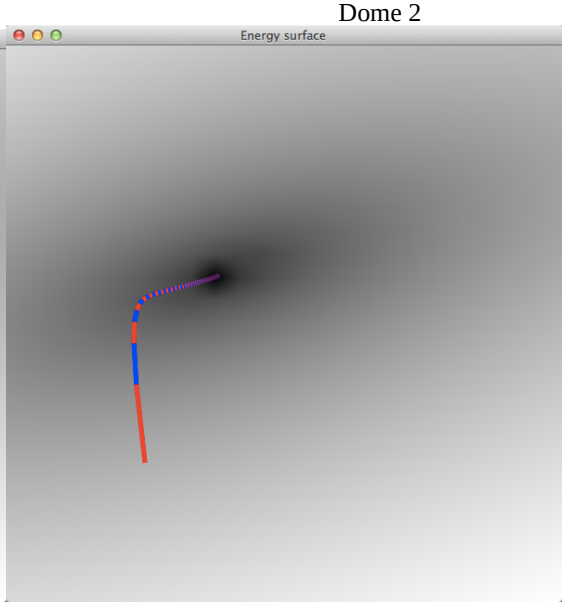
Demo 1



Dome 2



Demo 3



Dome 4

(c)

The results are : (from the experiments on λ = 0.005, 0.05, 0.5, 1, 2; learningRate η = 0.0001, 0.001, 0.01, 0.1, 1; Epoches = 20, 30; batch_size = 1, 2, 4, 8, 16, 64, 96, 100, 200)

lambda	learning rate	opt method	loss function	update	epoches	batch size	train_error	train_loss	test_error	test_l
0.05	0.1	sgd	hinge	minibatch	30	64	0.104	0.3018762209	0.07	0.25872
0.5	0.1	sgd	logis	minibatch	20	96	0.104	0.7336468077	0.077	0.67220
0.005	0.1	sgd	quant	minibatch	20	1	0.1346666667	0.5963804043	0.103	0.58842
0.5	0.1	sgd	mse	minibatch	30	16	0.1303333333	0.9292741042	0.113	0.94725
lambda	learning rate	opt method	loss function	update	epoches	batch size	train_error	train_loss	test_error	test_l
0.005	0.001	sgd	logis	stochastic	20	1	0.0806666667	0.5239487211	0.07	0.48642
0.005	0.01	sgd	hinge	stochastic	30	1	0.0773333333	0.2498171644	0.07	0.23766
1	0.0001	sgd	mse	stochastic	20	1	0.0946666667	0.5705446205	0.101	0.59782
1	0.0001	sgd	quant	stochastic	20	1	0.1096666667	0.702175932	0.106	0.71138
lambda	learning rate	opt method	loss function	update	epoches	batch size	train_error	train_loss	test_error	test_l
1	0.001	lbfgs	hinge	minibatch	30	2	0.097	0.362220316	0.085	0.34771
0.5	0.01	lbfgs	logis	minibatch	20	2	0.107	0.7337109791	0.092	0.68758
2	0.01	lbfgs	mse	minibatch	20	8	0.1066666667	0.677565922	0.097	0.72457
2	0.001	lbfgs	quant	minibatch	20	1	0.1396666667	0.5944390088	0.133	0.5948
lambda	learning rate	opt method	loss function	update	epoches	batch size	train_error	train_loss	test_error	test_l
0.05	1	lbfgs	hinge	batch	30	1	0.0663333333	0.1952524569	0.052	0.17310
0.05	0.1	lbfgs	logis	batch	30	1	0.066	0.3965978207	0.058	0.40585
0.05	0.01	lbfgs	mse	batch	30	1	0.1013333333	0.4114685624	0.096	0.51610
0.005	0.01	lbfgs	quant	batch	20	1	0.148	0.4563072822	0.127	0.43317
lambda	learning rate	opt method	loss function	update	epoches	batch size	train_error	train_loss	test_error	test_l
0.005	0.001	bfgs	hinge	stochastic	30	1	0.0793333333	0.23686668	0.072	0.23293
0.005	0.001	bfgs	logis	stochastic	20	1	0.0766666667	0.512420865	0.073	0.50474
2	0.0001	bfgs	mse	stochastic	30	1	0.0996666667	0.6525600832	0.091	0.67772
2	0.0001	bfgs	quant	stochastic	20	1	0.1043333333	0.8017792704	0.101	0.81289
lambda	learning rate	opt method	loss function	update	epoches	batch size	train_error	train_loss	test_error	test_l
0.5	0.1	bfgs	logis	minibatch	30	100	0.102	0.6963402351	0.073	0.6378
0.5	1	bfgs	hinge	minibatch	20	2	0.0933333333	0.4319311871	0.082	0.41707
0.5	0.01	bfgs	mse	minibatch	30	8	0.1193333333	0.5209315395	0.112	0.58914
0.5	0.1	bfgs	quant	minibatch	20	2	0.1223333333	0.5994554909	0.114	0.62849
lambda	learning rate	opt method	loss function	update	epoches	batch size	train_error	train_loss	test_error	test_l
0.05	0.001	cg	logis	batch	30	1	0.066	0.4044715311	0.058	0.42492
0.005	0.1	cg	hinge	batch	20	1	0.073	0.2013545242	0.061	0.20747
0.05	0.0001	cg	mse	batch	30	1	0.11	0.4177899314	0.091	0.40990
2	1	cg	quant	batch	20	1	0.1493333333	0.4578497039	0.128	0.4249

And for each method we choose good loss function as following:

lambda	learning rate	opt method	loss function	update	epoches	batch size	train_error	train_loss	test_error	test_l
0.05	1	lbfgs	hinge	batch	30	1	0.0663333333	0.1952524569	0.052	0.17310
0.05	0.001	cg	logis	batch	30	1	0.066	0.4044715311	0.058	0.42492
0.005	0.001	sgd	logis	stochastic	20	1	0.0806666667	0.5239487211	0.07	0.48642
0.05	0.1	sgd	hinge	minibatch	30	64	0.104	0.3018762209	0.07	0.25872
0.005	0.001	bfgs	hinge	stochastic	30	1	0.0793333333	0.23686668	0.072	0.23293
0.5	0.1	bfgs	logis	minibatch	30	100	0.102	0.6963402351	0.073	0.6378
1	0.001	lbfgs	hinge	minibatch	30	2	0.097	0.362220316	0.085	0.34771

All results are store in p2c_result.xls

Problem 3:Binary linear learning in Vowpal Wabbit

bی و

1ی و The recall means the rate of real malicious urls will be detected. High recall means more malicious urls will be detected.

The precision means the rate of real malicious urls among the detacted malicious urls. High precision means less good urls predicted as malicious.

2ی و The recall will increase when we raise the weight of +1 and the precision will decrease. Because when we reaise the weight of +1, then the loss on +1 example will increase which will force the parameters to make less error on +1 examples and in the other hand will make more errors on -1 examples.
Acoording to the definition of recall

$$\text{Recall} = \frac{\text{Number of } +1 \text{ examples predicated as } +1}{\text{Number of } +1 \text{ examples}}$$

we know the numerator will increase when the denominator does not change.
The precision will decrease.
And according to the definition of precision

$$\text{Precision} = \frac{\text{Number of } +1 \text{ examples predicated as } +1}{\text{Number of examples predicated as } +1} = \frac{1}{1+\frac{\text{Number of } -1 \text{ examples predicated as } +1}{\text{Number of } +1 \text{ examples predicated as } +1}}$$

So the change of precision will depend on the ratio

$$\frac{\text{Number of } -1 \text{ examples predicated as } +1}{\text{Number of } +1 \text{ examples predicated as } +1}$$

First if we assume the recall of +1 is already very high, then the change of ‘Number of +1 examples predicated as +1‘ won’t be big. On the other hand there are many -1 examples can be added to the set “ examples predicated as +1”. Secondly, because of the weight difference, the loss will decrease when we make mistakes on several -1 exampls to make a correct on +1 example.

(c)

Quantile

	Error	Recall	Precision
0.2	0.0273512	0.925041	0.99196
0.25	0.023565	0.938401	0.989784
0.333333	0.0248454	0.932865	0.991352
0.5	0.0194656	0.954372	0.986281
1.	0.0240337	0.936332	0.990814
2.	0.0147841	0.972326	0.982711
3.	0.0152117	0.981266	0.973282
4.	0.0148778	0.974979	0.979758
5.	0.0166251	0.987994	0.962654

Squared

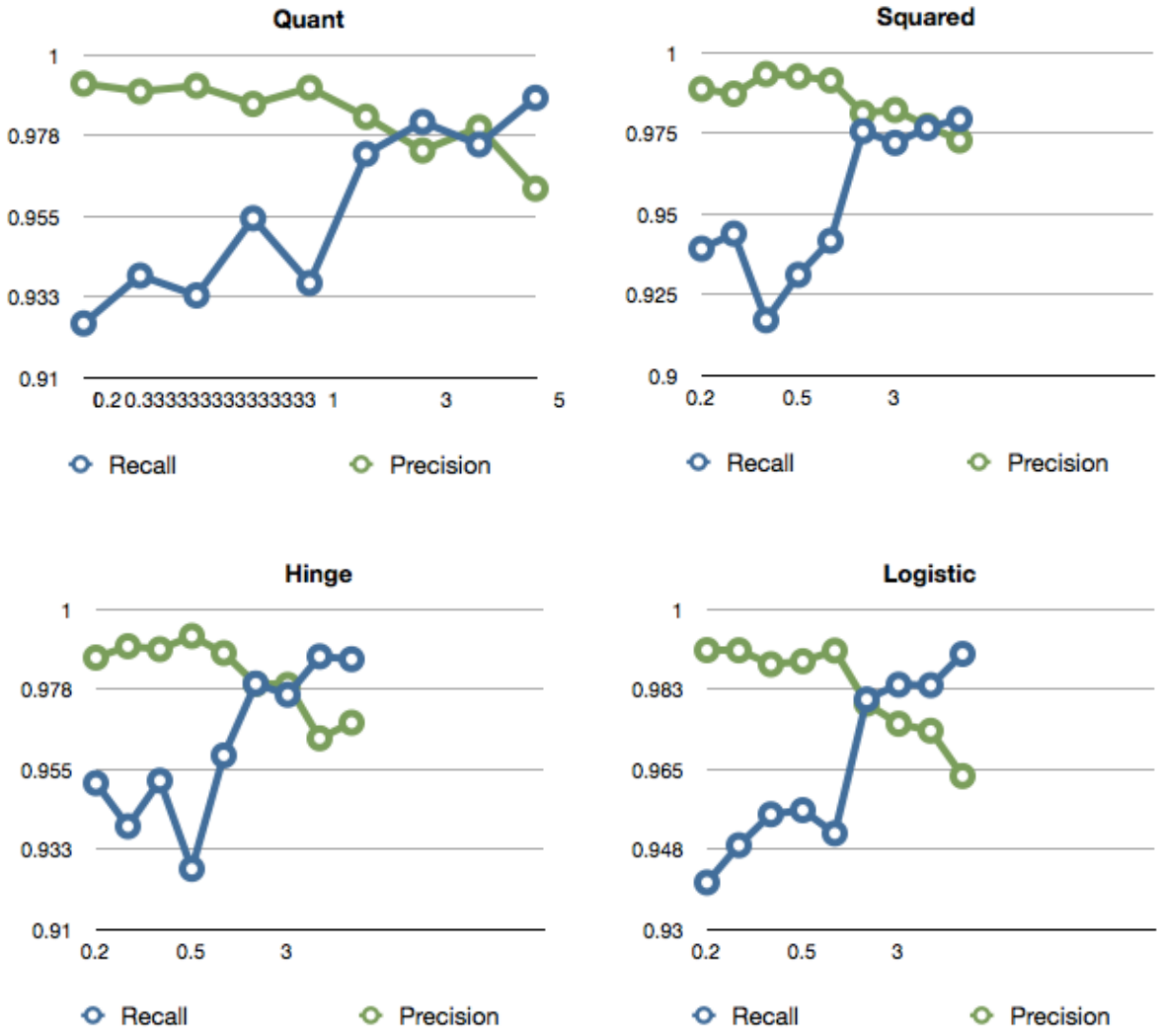
	Error	Recall	Precision
0.2	0.0237563	0.939156	0.988657
0.25	0.0226299	0.943831	0.987149
0.333333	0.0294388	0.917018	0.993276
0.5	0.0250768	0.931059	0.9926
1.	0.0221349	0.941621	0.991311
2.	0.0142402	0.975583	0.981178
3.	0.0152284	0.97195	0.982144
4.	0.0152201	0.976575	0.977207
5.	0.0159071	0.979313	0.972711

Hinge

	Error	Recall	Precision
0.2	0.0207362	0.951049	0.986338
0.25	0.0236652	0.938919	0.989539
0.333333	0.0196065	0.951765	0.988689
0.5	0.0265725	0.926938	0.99242
1.	0.0178698	0.958832	0.987616
2.	0.0141608	0.979029	0.978943
3.	0.015295	0.975901	0.978692
4.	0.0169311	0.986672	0.963648
5.	0.0156609	0.985897	0.968048

Logistic

	Error	Recall	Precision
0.2	0.0227524	0.940198	0.991012
0.25	0.0200584	0.948339	0.990974
0.333333	0.0188027	0.955109	0.987915
0.5	0.0183292	0.956016	0.988587
1.	0.0192929	0.950942	0.990902
2.	0.0135834	0.980258	0.979303
3.	0.0141996	0.983467	0.974909
4.	0.0145941	0.983353	0.973377
5.	0.0158988	0.990181	0.963479



Remarks: There are two ways to define the recall and precision in weighted case. One is to use the original definition. The other is to define in a weighted way. The above argument is on the base on original definition.

Weighted definition of recall and precision:

$$\text{Recall}^w = \frac{\text{Cumulate weight of } +1 \text{ examples predicated as } +1}{\text{Cumulate weight of } +1 \text{ examples}} = \text{Recall}$$

$$\text{Precision} = \frac{\text{Cumulate weight of } +1 \text{ examples predicated as } +1}{\text{Cumulate weight of examples predicated as } +1}$$

According to this definition, the recall is the same as original one, but precision will change. And because

$$\text{Precision} = \frac{\text{Cumulate weight of } +1 \text{ examples predicated as } +1}{\text{Cumulate weight of examples predicated as } +1} = \frac{1}{1+\frac{\text{Cumulate weight of } -1 \text{ examples predicated as } +1}{\text{Cumulate weight of } +1 \text{ examples predicated as } +1}}$$

the precision will increase. Because we will have less loss when we make mistakes on several -1 exampls to make a correct on +1 example. Then increased cumulate weight of -1 examples predicated as +1 should be less than increased cumulate weight of +1 examples predicated as +1.

Q u a n t i l e

Squared

	Error	Recall	Precision
0.2	0.00744618	0.925041	0.961055
0.25	0.00783342	0.938401	0.960352
0.333333	0.0106183	0.932865	0.974497
0.5	0.0116594	0.954372	0.972934
1.	0.0240337	0.936332	0.990814
2.	0.0111146	0.972326	0.99128
3.	0.00912991	0.981266	0.990932
4.	0.00748289	0.974979	0.994861
5.	0.00716241	0.987994	0.992301

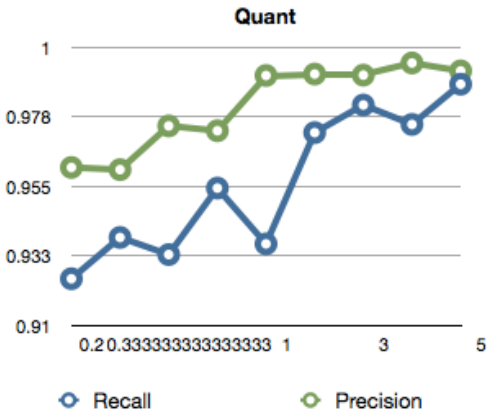
	Error	Recall	Precision
0.2	0.00646749	0.939156	0.945748
0.25	0.00752258	0.943831	0.950505
0.333333	0.0125814	0.917018	0.980096
0.5	0.0150203	0.931059	0.985309
1.	0.0221349	0.941621	0.991311
2.	0.0107057	0.975583	0.990499
3.	0.00913991	0.97195	0.993976
4.	0.00765501	0.976575	0.994203
5.	0.00685311	0.979313	0.99442

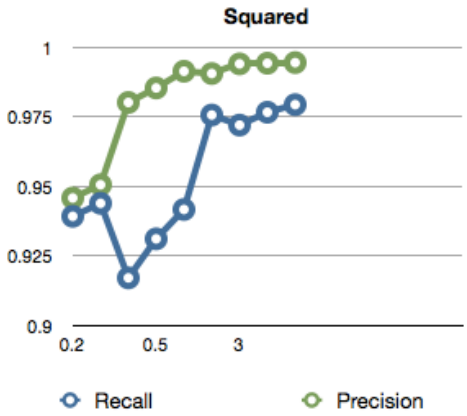
Hinge

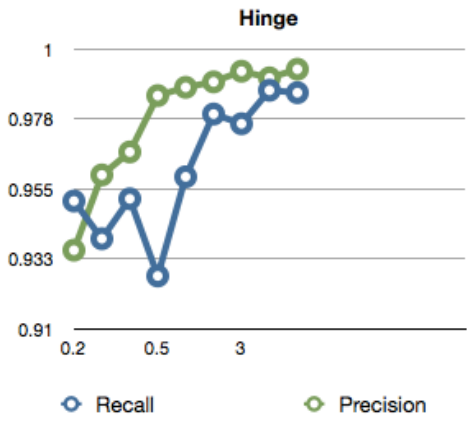
	Error	Recall	Precision
0.2	0.0056453	0.951049	0.935229
0.25	0.00786673	0.938919	0.959431
0.333333	0.00837933	0.951765	0.966818
0.5	0.0159162	0.926938	0.984954
1.	0.0178698	0.958832	0.987616
2.	0.010646	0.979029	0.989359
3.	0.00917991	0.975901	0.992795
4.	0.0085156	0.986672	0.990657
5.	0.00674701	0.985897	0.993442

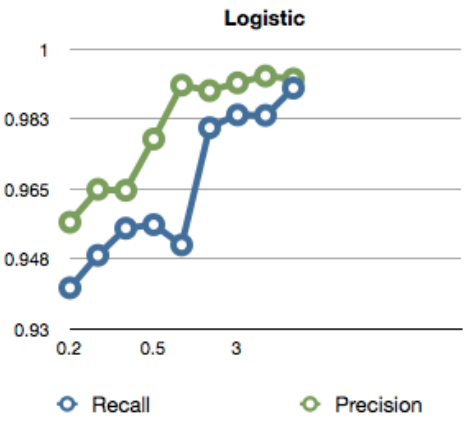
Logistic

	Error	Recall	Precision
0.2	0.00619418	0.940198	0.956621
0.25	0.00666778	0.948339	0.964849
0.333333	0.00803582	0.955109	0.964602
0.5	0.0109787	0.956016	0.977431
1.	0.0192929	0.950942	0.990902
2.	0.0102119	0.980258	0.989543
3.	0.00852241	0.983467	0.991494
4.	0.00734016	0.983353	0.993209
5.	0.00684951	0.990181	0.992476






Hinge
1
0.978
0.955
0.933
0.91
0.2
0.5
3
Recall
Precision


Logistic
1
0.983
0.965
0.948
0.93
0.2
0.5
3
Recall
Precision