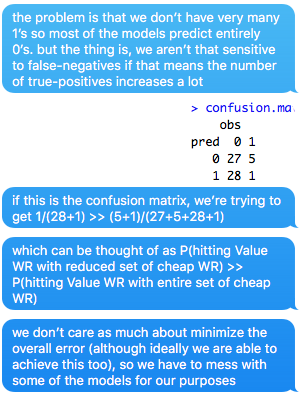
**classifyValueWR.R**



Basically, we want to remove the players that have no chance of hitting.

**1. http://www.win-vector.com/blog/2015/02/does-balancing-classes-improve-classifier-performance/**

Accuracy is simply the fraction of datums classified correctly.

Precision is the fraction of datums classified as positive that really were; equivalently, it’s an estimate of the conditional probability of a datum being in the positive class, given that it was classified as positive.

Recall (also called sensitivity or the true positive rate) is the fraction of positive datums in the population that were correctly identified.

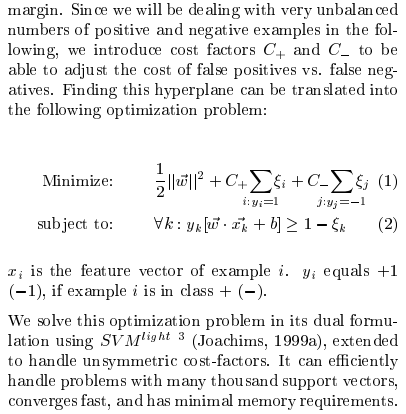
Specificity is the true negative rate, or one minus the false positive rate: the number of negative datums correctly identified as such.

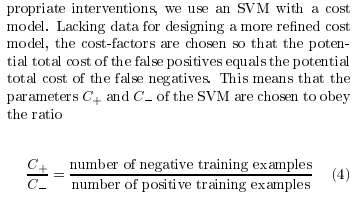
**2. http://stats.stackexchange.com/questions/38412/the-general-approaches-for-improving-a-svm-based-classifier-which-is-low-precisi**

For example, if I'm classifying 100 data points, 95 of which belong to class A, and 5 of which belong to class B, many machine learning algorithms (SVM included) will just classify everything/most things as class A, **yielding great recall but awful precision.**

Used this paper to solve: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1839342/

"...Since we will be dealing with very unbalanced numbers of positive and negative examples, we introduce cost factors C\_+ and C\_- to be able to adjust the cost of false positives vs. false negatives...". (paper: http://www.cs.cornell.edu/People/tj/publications/morik\_etal\_99a.pdf).





**When C\_+ > C\_-, precision is increased.** hmm this should be **C\_+ < C\_-** i think.

**Now, if you want just an out-of-the-box solution, the ratio of C\_+ to C\_- can be passed as the -j parameter to SVMLight: http://svmlight.joachims.org/**

2015 paper customizes the LIBSVM source code:

https://www.csie.ntu.edu.tw/~cjlin/libsvm/

Consider sparse matrix transformations too.

**week.min <- 6**

**salary.threshold <- 5000**

**fpts.threshold <- 18.5**

**1. splinedot SVM**

more on spline kernel: http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/#spline (http://www.svms.org/tutorials/Gunn1998.pdf)

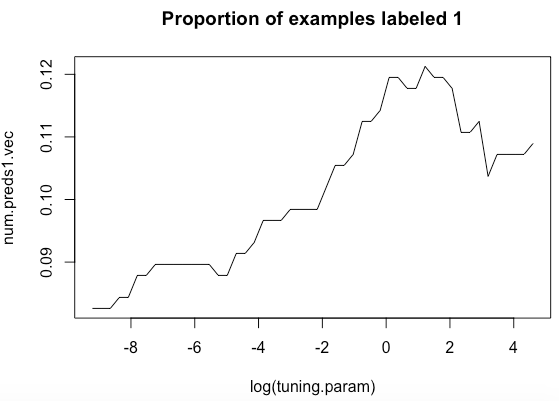
**i. Tuning C by minimizing CV error**

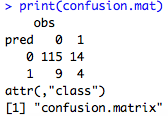
issue is that no 1’s predicted on testing set (terrible precision).

**ii. Tuning C by maximizing P(hitting Value WR with reduced set of cheap WR)**

issue is that it still doesn’t predict enough 1’s (better precision but not enough)

**iii. Tuning C by maximizing number of 1's predicted (as a % of total examples)**

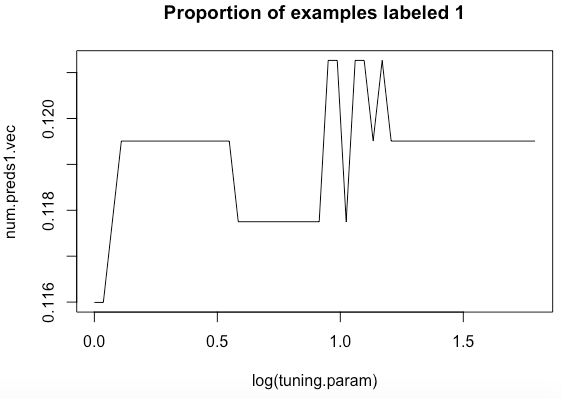


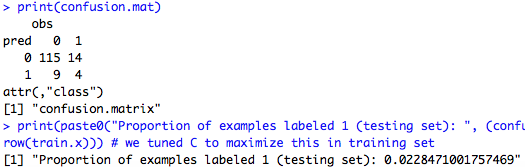




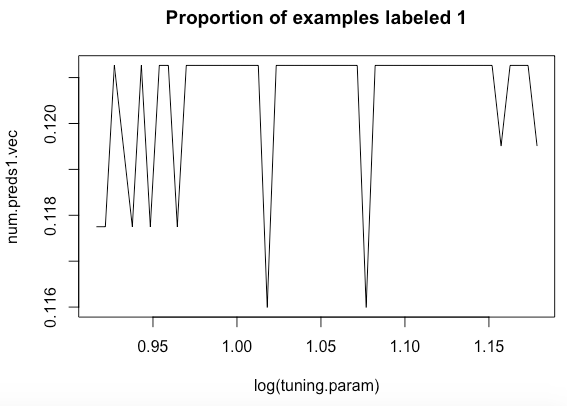
weird that testing set proportion is much lower than training (~12%)

let’s tune again with more iterations around the log(tuning.parameter) = 0 to 2 range, i.e. ln.tuning.param <- seq(log(1.0), log(6.0), length.out=num.C)

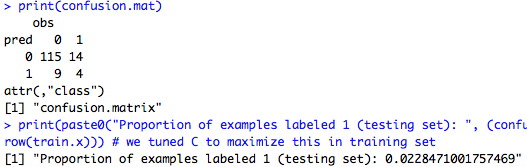




Let’s try fine tuning C a little more. ln.tuning.param <- seq(log(2.50), log(3.25), length.out=num.C)

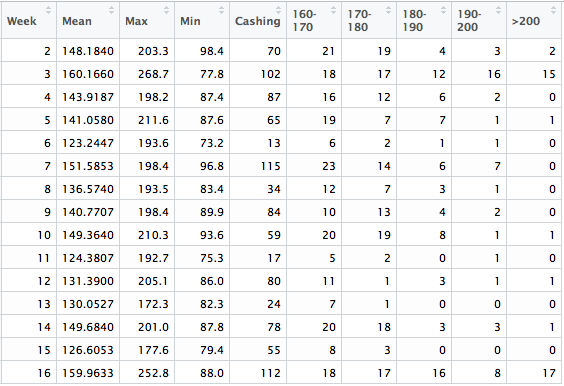


C.optimal (the first maximum in the plot) here is 2.5269

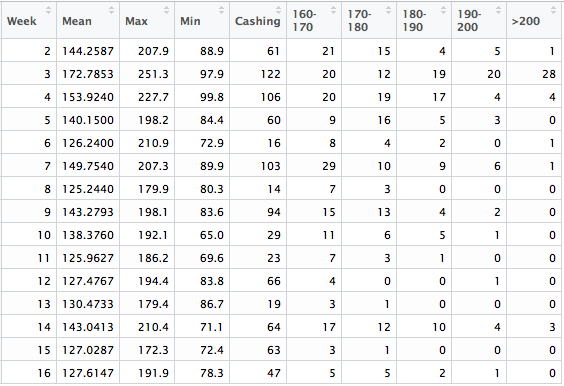


classification on the testing set is still the same, so all this extra tuning doesn’t really achieve anything.

**model1 (ValueWR using baseline, spike projections by 1.5 using linear kernel svmlight, form 14, overlap 4, defexp 0.25, wrexp 0.25, rbexp 0.75, teexp 0.75, qb 0.5 for weeks 7-16; spike projections for baseline ValueWR 1’s by 1.5 for weeks 2-6)**



form 4, overlap 4, exposure 0.4 **(baseline)**



form 14, overlap 4, defexp 0.25, wreck 0.25, rbexp 0.75, teexp 0.75, qb 0.5 for weeks 7-16; spike projections for baseline ValueWR 1’s by 1.5 for weeks 2-6 **(baseline)**

