Win-Vector Blog

The Win-Vector LLC data science blog

Does Balancing Classes Improve Classifier Performance?

February 27, 2015 Nina Zumel data science, Expository Writing, Practical Data Science, Pragmatic Data Science, Pragmatic Machine Learning, Statistics classification, classifier quality, folk theorems, Logistic Regression, R, random forest, SVM

It’s a folk theorem I sometimes hear from colleagues and clients: that you must balance the class prevalence before training a classifier. Certainly, I believe that classification tends to be *easier* when the classes are nearly balanced, especially when the class you are actually interested in is the rarer one. But I have always been skeptical of the claim that artificially balancing the classes (through resampling, for instance) always helps, when the model is to be run on a population with the native class prevalences.

On the other hand, there are situations where balancing the classes, or at least enriching the prevalence of the rarer class, might be necessary, if not desirable. Fraud detection, anomaly detection, or other situations where positive examples are hard to get, can fall into this case. In this situation, I’ve suspected (without proof) that SVM would perform well, since the formulation of hard-margin SVM is pretty much distribution-free. Intuitively speaking, if both classes are far away from the margin, then it shouldn’t matter whether the rare class is 10% or 49% of the population. In the soft-margin case, of course, distribution starts to matter again, but perhaps not as strongly as with other classifiers like logistic regression, which explicitly encodes the distribution of the training data.

So let’s run a small experiment to investigate this question.

Experimental Setup

We used the ISOLET dataset, available at the UCI Machine Learning repository. The task is to recognize spoken letters. The training set consists of 120 speakers, each of whom uttered the letters A-Z twice; 617 features were extracted from the utterances. The test set is another 30 speakers, each of whom also uttered A-Z twice.

Our chosen task was to identify the letter “n”. This target class has a native prevalence of about 3.8% in both test and training, and is to be identified from out of several other distinct co-existing populations. This is similar to a fraud detection situation, where a specific rare event has to be a population of disparate “innocent” events.

We trained our models against a training set where the target was present at its native prevalence; against training sets where the target prevalence was enriched by resampling to twice, five times, and ten times its native prevalence; and against a training set where the target prevalence was enriched to 50%. This replicates some plausible enrichment scenarios: enriching the rare class by a large multiplier, or simply balancing the classes. All training sets were the same size (N=2000). We then ran each model against the same test set (with the target variable at its native prevalence) to evaluate model performance. We used a threshold of 50% to assign class labels (that is, we labeled the data by the most probable label). To get a more stable estimate of how enrichment affected performance, we ran this loop ten times and averaged the results for each model type.

We tried three model types:

- `cv.glmnet` from R package `glmnet`: Regularized logistic regression, with `alpha=0` (L2 regularization, or ridge). `cv.glmnet` chooses the regularization penalty by cross-validation.
- `randomForest` from R package `randomForest`: Random forest with the default settings (500 trees, `nvar/3`, or about 205 variables drawn at each node).
- `ksvm` from R pacakge `kernlab`: Soft-margin SVM with the radial basis kernel and C=1

Since there are many ways to resample the data for enrichment, here’s how I did it. The target variable is assumed to be TRUE/FALSE, with TRUE as the class of interest (the rare one). `dataf` is the data frame of training data, `N` is the desired size of the enriched training set, and `prevalence` is the desired target prevalence.

```
makePrevalence = function(dataf, target,
                          prevalence, N) {
  # indices of T/F
  tset_ix = which(dataf[[target]])
  others_ix = which(!dataf[[target]])

  ntarget = round(N*prevalence)
  heads = sample(tset_ix, size=ntarget,
                 replace=TRUE)
  tails = sample(others_ix, size=(N-ntarget),
                 replace=TRUE)

  dataf[c(heads, tails),]
}
```

Training at the Native Target Prevalence

Before we run the full experiment, let's look at how each of these three modeling approaches does when we fit models the obvious way — where the training and test sets have the same distribution:

```
## [1] "Metrics on training data"
## accuracy precision   recall specificity         label
##   0.9985 1.0000000 0.961039     1.00000      logistic
##   1.0000 1.0000000 1.000000     1.00000 random forest
##   0.9975 0.9736842 0.961039     0.99896           svm
## [1] "Metrics on test data"
##  accuracy precision    recall specificity         label
## 0.9807569 0.7777778 0.7000000   0.9919947      logistic
## 0.9717768 1.0000000 0.2666667   1.0000000 random forest
## 0.9846055 0.7903226 0.8166667   0.9913276           svm
```

We looked at four metrics. *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.

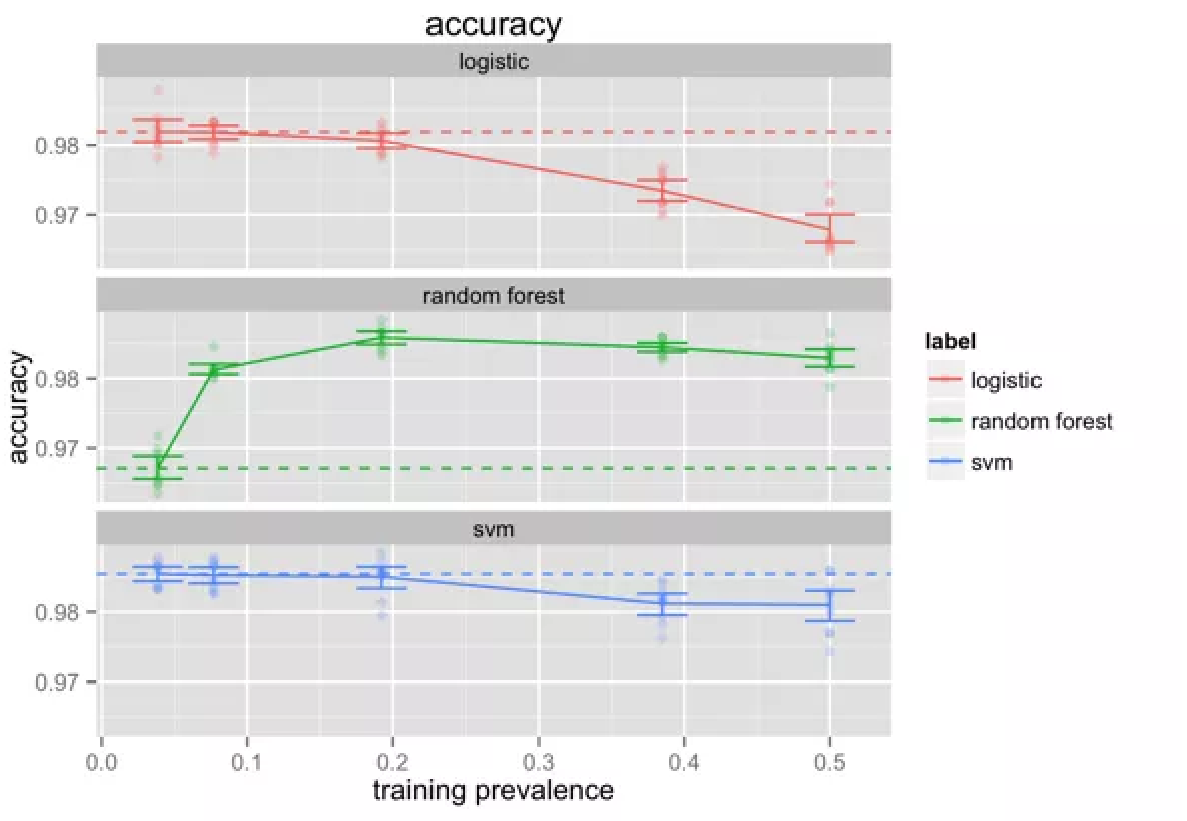
As the table above shows, random forest did perfectly on the training data, and the other two did quite well, too, with nearly perfect precision/specificity and high recall. However, random forest's recall plummeted on the hold-out set, to 27%. The other two models degraded as well (logistic regression more than SVM), but still manage to retain decent recall, along with good precision and specificity. Random forest also has the lowest accuracy on the test set (although 97% still *looks* pretty good — another reason why accuracy is not always a good metric to evaluate classifiers on. In fact, since the target prevalence in the data set is only about 3.8%, a model that always returned FALSE would have an accuracy of 96.2%!).

One could argue that if precision is the goal, then random forest is still in the running. However, remember that the goal here is to identify a rare event. In many such situations (like fraud detection) one would expect that high recall is the most important goal, as long as precision/specificity are still reasonable.

Let's see if enriching the target class prevalence during training improves things.

How Enriching the Training Data Changes Model Performance

First, let's look at accuracy.

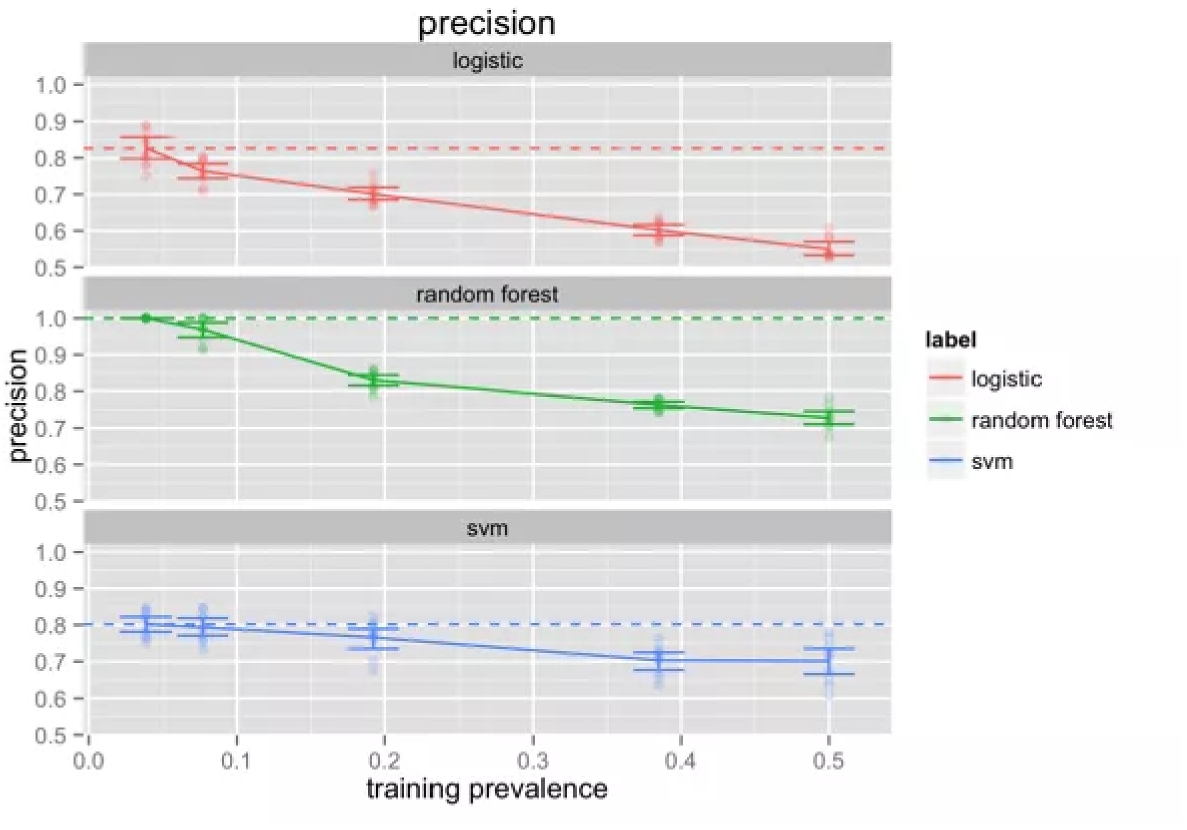


The x-axis is the prevalence of the target in the training data; the y-axis gives the accuracy of the model on the test set (with the target at its native prevalence), averaged over ten draws of the training set. The error bars are the bootstrap estimate of the 98% confidence interval around the mean, and the values for the individual runs appear as transparent dots at each value. The dashed horizontal represents the accuracy of a model trained at the target class's true prevalence, which we'll call the model's *baseline performance*. Logistic regression degraded the most dramatically of the three models as target prevalence increased. SVM degraded only slightly. Random forest improved, although its best performance (when training at about 19% prevalence, or five times native prevalence) is only slightly better than SVM's baseline performance, and its performance at 50% prevalence is worse than the baseline performance of the other two classifiers.

Logistic regression's degradation should be no surprise. Logistic regression optimizes deviance, which is strongly distributional; in fact, logistic regression (without regularization) preserves the marginal probabilities of the training data. Since logistic regression is so well calibrated to the training distribution, changes in the distribution will naturally affect model performance.

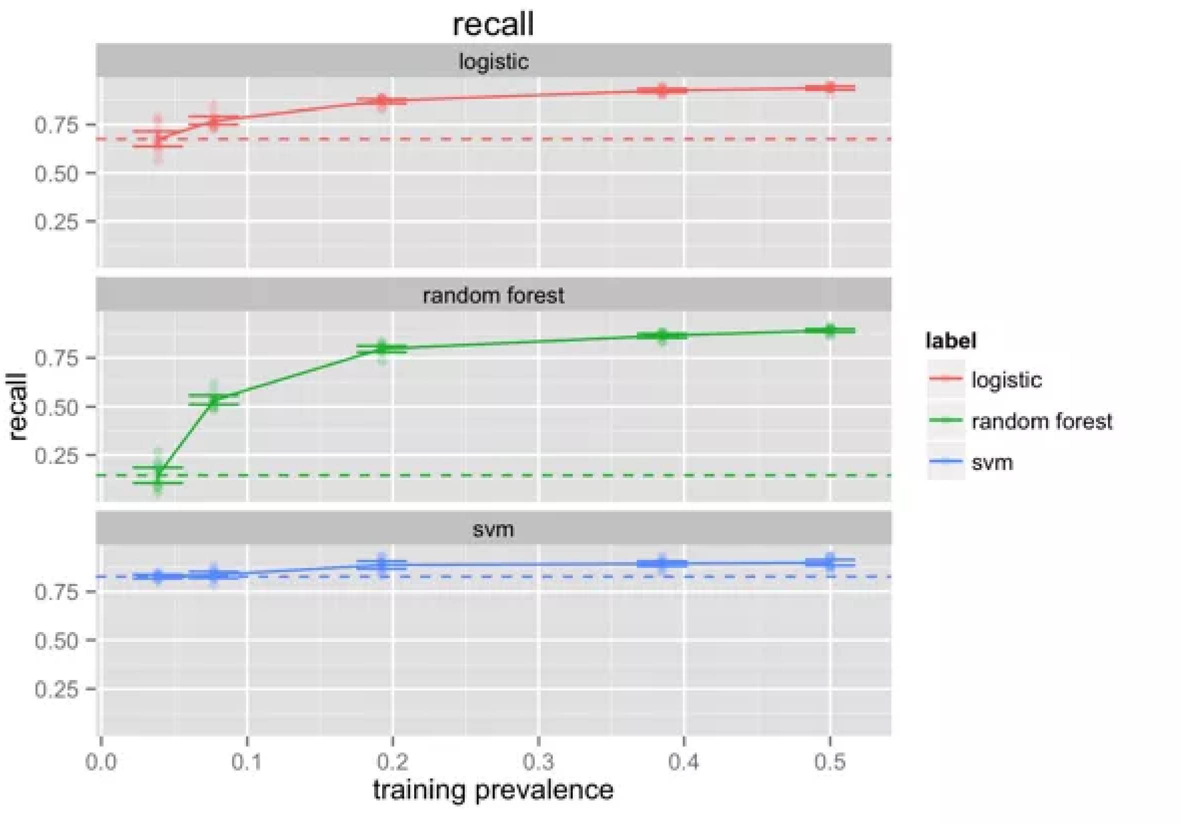
The observation that SVM's accuracy stayed very stable is consistent with my surmise that SVM's training procedure is not strongly dependent on the class distributions.

Now let's look at precision:



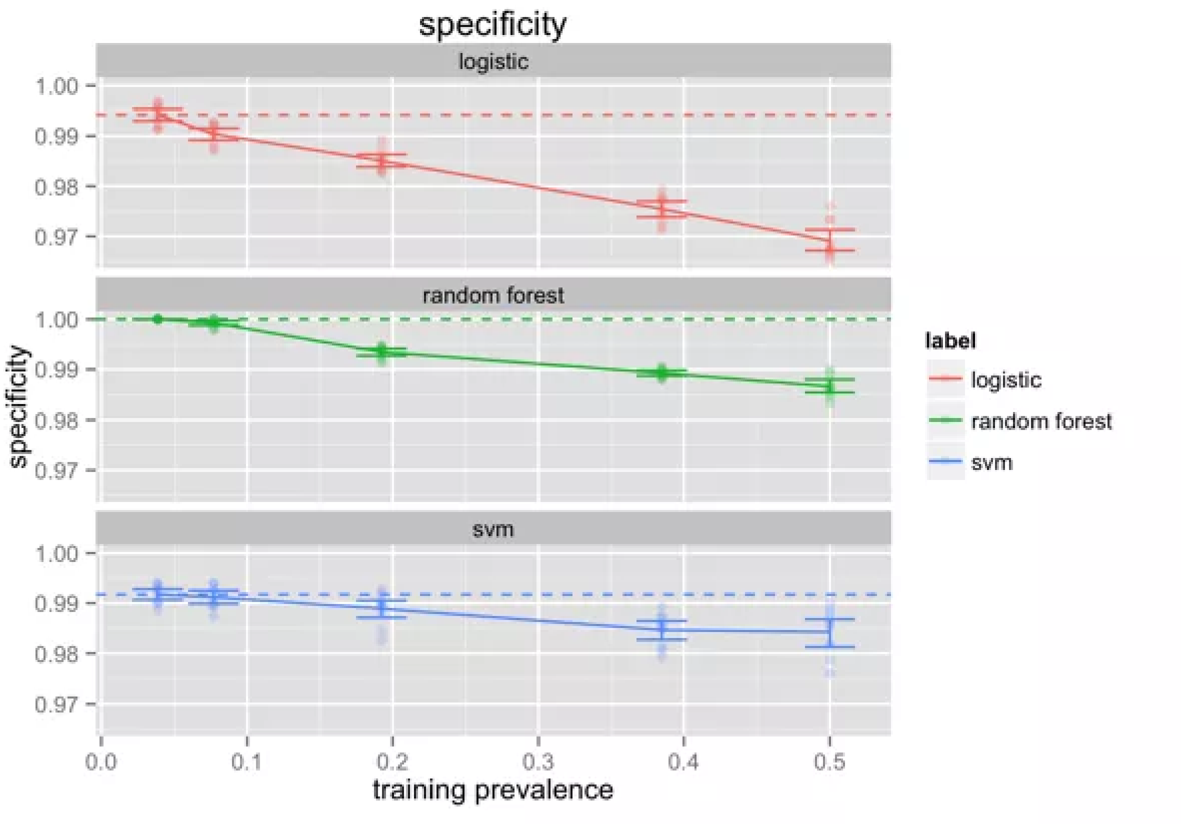
All of the models degraded on precision, random forest the most dramatically (since it started at a higher baseline), SVM the least. SVM and logistic regression were comparable at baseline.

Let's look at recall:



Enrichment improved the recall of all the classifiers, random forest most dramatically, although its best performance, at 50% enrichment, is not really any better than SVM's baseline recall. Again, SVM's recall moved the least.

Finally, let's look at specificity:



Enrichment degraded all models’ specificity (i.e. they all make more false positives), logistic regression’s the most dramatically, SVM’s the least.

The Verdict

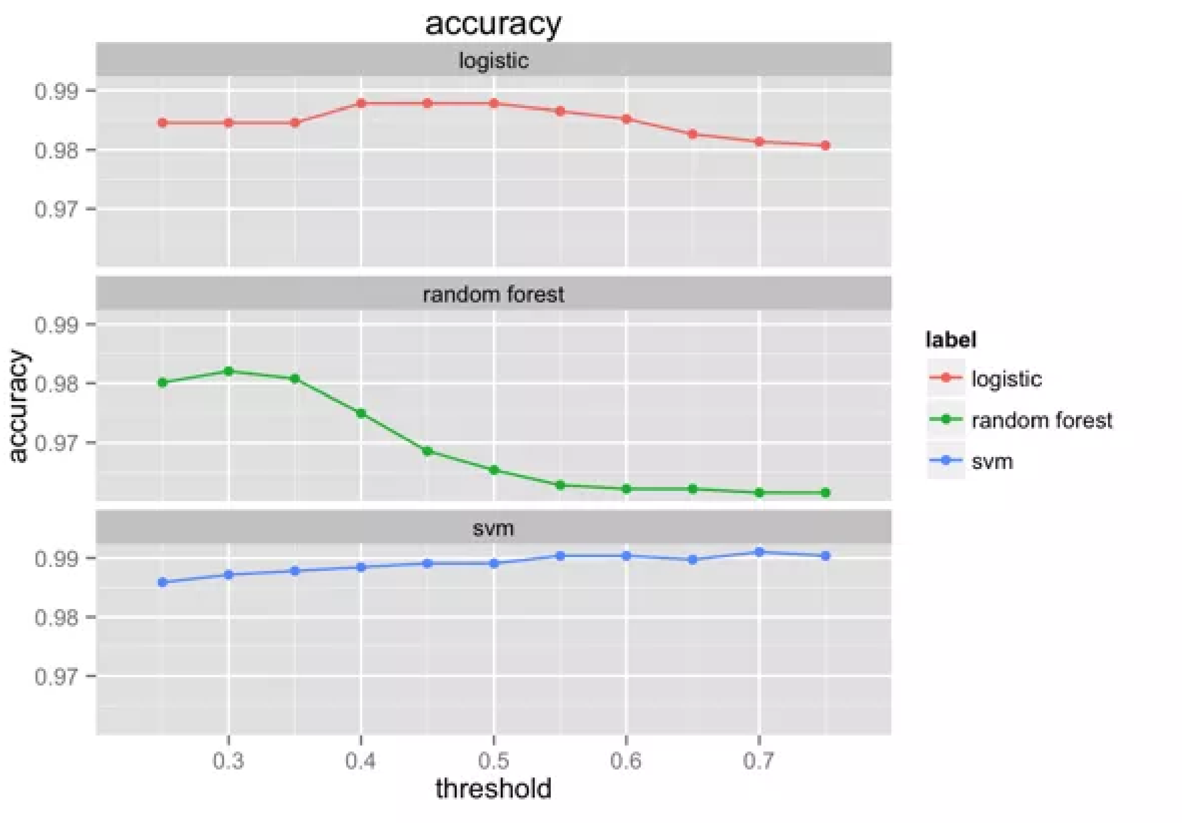
Based on this experiment, I would say that balancing the classes, or enrichment in general, is of limited value if your goal is to apply class labels. It did improve the performance of random forest, but mostly because random forest was a rather poor choice for this problem in the first place (It would be interesting to do a more comprehensive study of the effect of target prevalence on random forest. Does it often perform poorly with rare classes?).

Enrichment is not a good idea for logistic regression models. If you must do some enrichment, then these results suggest that SVM is the safest classifier to use, and even then you probably want to limit the amount of enrichment to less than five times the target class’s native prevalence — certainly a far cry from balancing the classes, if the target class is very rare.

The Inevitable Caveats

The first caveat is that we only looked at one data set, only three modeling algorithms, and only one specific implementation of each of these algorithms. A more thorough study of this question would consider far more datasets, and more modeling algorithms and implementations thereof.

The second caveat is that we were specifically supplying class labels, using a threshold. I didn't show it here, but one of the notable issues with the random forest model when it was applied to hold-out was that it no longer scored the datums along the full range of 0-1 (which it did, on the training data); it generally maxed out at around 0.6 or 0.7. This possibly makes using 0.5 as the threshold suboptimal. The following graph was produced with a model trained with the target class at native prevalence, and evaluated on our test set.



The x-axis corresponds to different thresholds for setting class labels, ranging between 0.25 (more permissive about marking datums as positive) and 0.75 (less permissive about marking datums as classifiers). You can see that the random forest model (which didn't score anything in the test set higher than 0.65) would have better accuracy with a lower threshold (about 0.3). The other two models have fairly close to optimal accuracy at the default threshold of 0.5. So perhaps it's not fair to look at the classifier performance without tuning the thresholds. However, if you're tuning a model that was trained on enriched data, you still have to calibrate the threshold on un-enriched data — in which case, you might as well train on un-enriched data, too. In the case of this random forest model, its best accuracy (at threshold=0.3) is about as good as random forest's accuracy when trained on a balanced data set, again suggesting that balancing the training set doesn't contribute much. Tuning the threshold may be enough.

However, suppose we don't need to assign class labels? Suppose we only need the score to sort the datums, hoping to sort most of the items of interest to the top? This could be the case when prioritizing transactions to be investigated as fraudulent. The exact fraud score of a questionable transaction might not matter — only that it's higher than the score of non-fraudulent events. In this case, would enrichment or class balancing improve? I didn't try it (mostly because I didn't think of it until halfway through writing this), but I suspect not.

Conclusions

- Balancing class prevalence before training a classifier does *not* across-the-board improve classifier performance.
- In fact, it is contraindicated for logistic regression models.
- Balancing classes or enriching target class prevalence may improve random forest classifiers.
- But random forest models may not be the best choice for very unbalanced classes.
- If target class enrichment is necessary (perhaps because of data scarcity issues), SVM may be the safest choice for modeling.

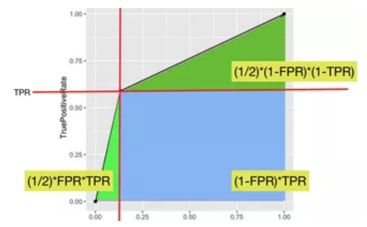
A knitr document of our experiment, along with the accompanying R markdown file, can be downloaded here, along with a copy of the ISOLET data.

SHARE THIS:

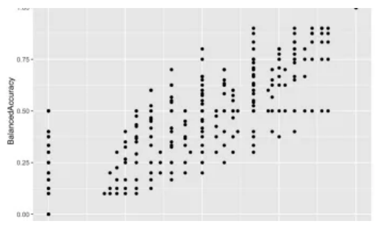
82 125

RELATED

The Geometry of Classifiers
In "Coding"



A bit on the F1 score floor
In "Mathematics"



A budget of classifier evaluation measures

In "Mathematics"

February 27, 2015 Nina Zumel data science, Expository Writing, Practical Data Science, Pragmatic Data Science, Pragmatic Machine Learning, Statistics classification, classifier quality, folk theorems, Logistic Regression, R, random forest, SVM

20 thoughts on “Does Balancing Classes Improve Classifier Performance?”

Michael Benesty

February 27, 2015 at 9:20 am

Just my 2 cents, but I am not surprised RF doesn’t work very well for this problem.

Each tree is built on a sample of the features and on a sample of the observations (to increase the variance of the trees).

If the correct label is rare, building on a sample of the observations is an important issue. The weight of a tree is decreased when its performance is not good, but in this extreme case it may be not enough to rebalance the unbalanced distribution of classes. This explanation matches your other observation being that decreasing the threshold increase the performance (because few well built trees can vote correctly).

It would be super interesting to update this article with gradient boosting algo. This kind of algo will increase its focus on badly learned observations and therefore should be able to catch the characteristics of the rare observations. And it would give to decision tree algo family a better image for this case :-)

Xgboost R package works super well (much more rapid than classical GBM package). If you are interested use Github version instead of Cran (can be installed easily with devtools). If you have any issue, contact me with the email I used for this comment or post an issue on the Github depository.

https://github.com/tqchen/xgboost/tree/master/R-package

There is a subsample parameter… I would not use it for this case.

Kind regards,
Michaël

Nina Zumel

February 27, 2015 at 10:27 am

Thank you for your reply!

Yes, I also suspected that RF wouldn't do well on this problem. Your diagnosis of RF is definitely consistent with the supposition that RF is not the best choice for the very unbalanced class situation, and gives insight as to why target class enrichment might improve RF's performance in that case.

The other reason to enrich a rare class is that it's another way to reweight the data, thereby giving the algorithm a clue as to what is important to you ("I really want you to get these cases right!"). Boosting, as you point out, gets at this, too. I actually planned to include gradient boosting in this study, but gbm crashed horribly while scoring the test set, and I never tracked the problem all the way down. So thank you for pointing me to Xgboost! I will check it out.

Michaël Benesty

February 28, 2015 at 11:35 pm

Btw, I have written two Vignettes for xgboost. They are not yet published (we will update cran version soon) but already available in github deposit.

In the one named discover your data you have the code to execute to prepare the data. In the other vignette all the code to build a nice model.

Kind regards,
Michaël

JimK

February 27, 2015 at 12:35 pm

What happens if you importance weight the negative samples in your logistic regression function ?

Nina Zumel

February 27, 2015 at 2:50 pm

I imagine that it gives results similar to resampling, so if you had data that was not correctly balanced, for whatever reason, you could approximately restore the correct prevalence by reweighting.

I'd still prefer to train and apply at the same prevalence, when possible. And not all algorithms support weights (though perhaps they all should).

ssantic

February 28, 2015 at 12:11 am

I still don't really understand why RF is a poor choice for unbalanced datasets. Kuhn and Johnson tout it very highly in their book "Applied Preditcitve Modeling", and I have had good results in R using the classwt, cutoff, and sampsize parameters for the randomForest function.

jmount

February 28, 2015 at 6:30 am

Random forest is a good method, Nina said as much in her "Geometry of Classifiers" article. It just may not be the best method in some situations. Because random forest trains on sub-samples, and a rare class would show a lot of variance (when re-sampled at the original rate) you might then wonder if this could be a problem situation.

But for all the speculation- if you get good results it is a good method for your problems, if you don't then there may be an issue.

Mahendra

March 1, 2015 at 10:06 am

Folks,
randomForest works very well in case of unbalanced class .

One of my model implemented for big Insurance Firm have class (0=98% ,1=2%)
And it worked like a charm got OOT accuracy at 0.74

It's just about tweaking parameters :)

Thanks

Matt

March 1, 2015 at 2:52 pm

Hello,

I am dealing with a similar problem in a project but instead of resampling to address unbalanced classes, I instead optimized on the kappa statistic instead of accuracy (or precision , recall, etc). Is this something you considered? If not, why? It is my (albeit rudimentary) understanding that the kappa statistic is a good way to do just this… Its also implemented in caret, which is what I'm using, so there's that too.

Thank you for a thoughtful and informative post!

Best regards,

Matt

---

jmount

March 1, 2015 at 3:37 pm

I think we are all circling around the same idea: randomForest is a good method, though in some situations you may want to give a hint as to how your application values sensitivity/specificity, type1/type2 errors, or precision/recall.

You can do this different ways, depending on what parameters a given implementation exposes. I don't think R's randomForest public interface encourages you to set a kappa-criteria, directly change its internal cost function (by changing the objective function), or to specify individual data weights (not an argument on R's randomForest, but common to a lot of other R algorithms). R's randomForest does have a classwt argument to change overall class weights (and therefore attempt to indirectly change the cost structure, but the documentation says it is ignored in regression mode).

In production you would use the easiest and most direct of the previous controls. For trying a few different algorithms quickly you tend prefer something simple like re-sampling (which doesn't have to be re-specialized for different parameters for each machine learning method). In all cases you are in fact tweaking parameters- so you do want to know if such parameter changes actually do anything, and if so what (improve or make things worse). Also, yes: at some point you may want to delegate parameter tuning to something like the R caret package (authored by Max Kuhn and others); which I assume would optimize kappa indirectly for randomForest by sweeping values through the classwt control (and using weights controls for other methods such as glm).

Dale

March 24, 2015 at 6:22 am

caret will optimize kappa directly, for an example see

http://www.r-bloggers.com/predictive-modelling-fun-with-the-caret-package/

Balancing the sample sizes in the R randomForest package is via sampsize – for a binary classification, you need to supply a vector of two integers, one for each classification. Make them equal to get balanced samples, and choose replace = TRUE (for sampling with replacement).

Tanmay

March 1, 2015 at 11:51 pm

Hi,
I have a general question. Have you guys applied ML techniques on time series data? (e.g. predicting up and down movement of stocks). Do you have any remarks comparing it with traditional econometric methods.
Thanks

jmount

March 2, 2015 at 7:58 am

We actually did a couple of times. A long time ago tracking a hypothesized close/morning inefficiency for one client, and another time for another client. The problem is with time series you have to do a lot to account for useless interfering short-term correlations (the autoregressive/ARMA portion) before digging deeper.

And there is a danger you move from what is called quantitative/statistical trading into "technical trading" (chartism).

Tanmay

March 2, 2015 at 9:26 pm

Thanks for the link. I undestand ARMA models, but not sure what you are referring to. Are you saying we first fit an ARMA model, and then analyse the residuals? Woudn't the remaining be just noise? I am probably misunderstanding something, would appreciate if you can clarify.
Thanks

jmount

March 2, 2015 at 9:31 pm

I was just speaking generally- that if you don't apply the ARMA thinking (look out for unit roots and such), you really run into trouble.

Max Ghenis

March 2, 2015 at 3:14 pm

Your caveat toward the end of 0.5 being a poor threshold for the random forest on the test set resonates with me, I've seen the same thing in my models. For this reason I've favored AUC over accuracy/precision/recall in evaluating RFs – did you check this?

jmount

March 2, 2015 at 5:05 pm

I think Nina was just using accuracy as quick and familiar notional measure, I know it is not one of her favorites (see http://www.win-vector.com/blog/2009/11/i-dont-think-that-means-what-you-think-it-means-statistics-to-english-translation-part-1-accuracy-measures/). For scoring systems she tends to use sorting/gain-based metrics, deviance, and AUC.

Abhishek Shrivastava

April 14, 2015 at 5:02 am

Hello Nina,

I came across your post on kdnuggets. You have an interesting analysis. However, as you also noted it is limited in (at least) selection of datasets to generalize the conclusions. Since this problem is of interest to me in my research, I though I'll share a few things I know.

There is more work on this problem, also called *imbalanced classification* in literature. Following are some key approaches and some references (there are many more good references):
* *Resampling for balancing class representation in training set* – This is one of the older approaches, similar to that presented in your blog post, and seems to have limited success in several datasets. (ref: 'KNN approach to unbalanced data distributions: A case study involving information extraction' by Zhang and Mani in ICML 2003)
* *Cost-sensitive learning* – Another old approach that works by modifying the loss function, penalizing mistakes in predicting rarer class more heavily (see – 'The foundations of cost-sensitive learning' by Elkan in IJCAI 2001)
* *Synthetic sampling* – A more recent sampling based approach. It generates artificial rare class samples. It works well (better than the two above) in many cases. (ref: 'SMOTE: Synthetic minority over-sampling technique' by Chawla, et al., in JAIR)
* *Decomposition-based approach* – Another recent approach. It works by decomposing the majority class, and then training multiple classifiers separating each majority class cluster with rare class. Seems to work very well too. (ref: 'Combating sub-clusters effect in imbalanced classification' by Zhao and Shrivastava in ICDM 2013).

Best,
Abhishek

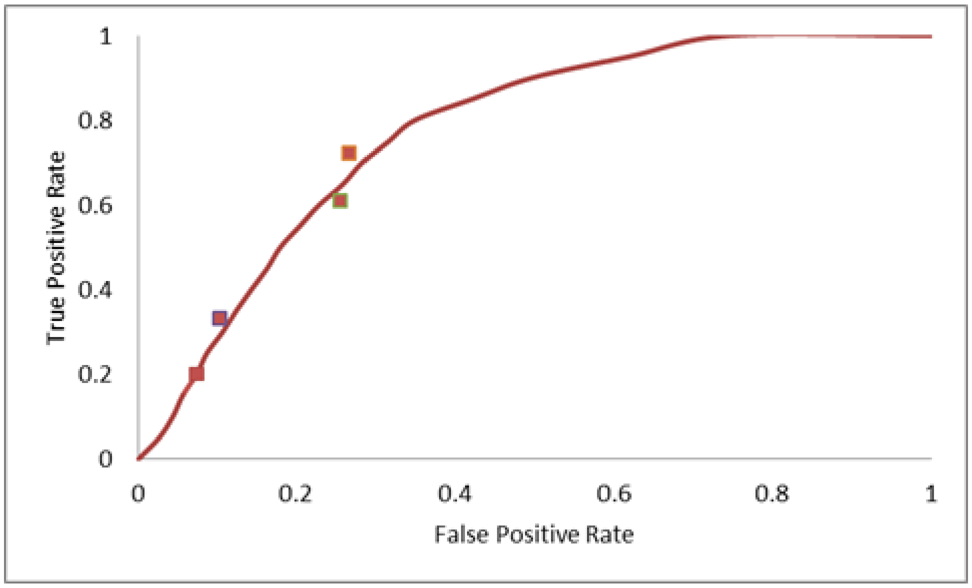
Rouzbeh Razavi

May 8, 2015 at 8:48 am

I found the article to be very interesting. This is especially considering that classification of rare events where you have significant class misbalanced is becoming more and more important.

Here is what I think on the subject. To me, class balancing will only help us to move to different regions of the ROC curve, while it is not changing the underlying *characteristics of the ROC curve*. In other words, it's only a mean for trading between Type I and Type II errors.

I am totally agreeing with you on the fact that modifications of class distributions may not be an effective way for changing the behaviour of the SVM (as it is not distributional in nature), but is more effective for logistic regression (and perhaps for likes of Naïve Bayesian), but again we are only moving across the ROC curve (which will be the objective in many applications as the cost of a false negative might be way higher than a false positive). To clarify this, I did a small experiment where the ROC curve for Logistic Regression classifier was experimentally derived (averaged over 100 runs) and then I tried different oversampling rates and the results are shown as square points. As you can see, experiments are well aligned with the ROC curve, confirming that by oversampling we are not changing the characteristics of the ROC curve. The minority class in this experiment originally accounted for less 2% of the population.



So in terms of conclusion, we might say "target class enrichment" my not be an efficient way to achieve the error type trade of for SVM, hence other methods such as adapting the classification threshold (post-process or by setting class weights when calling the classifier) may be a more appropriate solution, while this more effective for Logistic Regression.

Nina Zumel

May 8, 2015 at 8:54 am

This is a really great followup study, Rouzbeh! And it's consistent with the observation I made that changing the classification threshold works about as well as class rebalancing (since that is also moving up and down the ROC curve, as well). That emphasizes the point: there are things you can do to improve classification on unbalanced classes, but we need to check if merely re-balancing the training data is enough to achieve much more than moving along the ROC curve.

Thanks for sharing this.

Comments are closed.