UNIVERSITÀ DI PISA

A Critical Reassessment of the Saerens-Latinne-Decaestecker (SLD) Algorithm for Posterior Probability Adjustment

Andrea Esuli, Alessio Molinari and Fabrizio Sebastiani
ISTI, Consiglio Nazionale delle Ricerche

Alessio Molinari

The SLD Algorithm

- SLD is an instance of Expectation-Maximization, an algorithm for finding maximum likelihood estimates of parameters for models that depend on unobserved labels;

- We would like to leverage SLD in order to readjust our *a priori* and *a posteriori* probabilities (i.e. *priors* and *posteriors*) to new data;

- The algorithm key idea is to iteratively update the priors with the posteriors and vice versa.

A bit of Whys

- Why am I working on this daunting and, some could say, boring topic?
  Let’s say it is a consequence of my master thesis

- Why using the SLD algorithm?
  It looked like a good algorithm to use in my specific context, i.e.
  “transductive” dataset!

- Why did we decide to reassess an algorithm from the dinosaur era?
  Because it is one of the standard algorithms for probability
  readjustment!

IN SUPREMÆ DIGNITATIS 1343

Before we start...
what is a *prior* probability?

- Let’s say you wake up with a really bad hangover: you don’t remember what day, month or year it is;
- Someone breaks into your room and asks you which do you think is the probability of rain today;
- You cannot base your prediction on anything else than your *a priori* knowledge, i.e. you know that in Pisa it rains 30% of the days over a year.

Before we start…
what is a *posterior* probability?

- Now you turn on your mobile phone and see it’s the first of january;
- You open the windows and notice it is very cloudy;
- Given these elements, you can now give a more accurate prediction of today probability of rain (eg. 70%).

Some details on the experiments setup

- Since my EAP presentation some things have changed: we dropped the 20NEWSGROUPS dataset and kept RCV1-V2 only;

- We tested SLD against posteriors coming from several machine learning classifiers: Support Vector Machines, Logistic Regression, Random Forests, Multinomial Naive Bayes;

- We evaluate the quality of the prior probabilities with the Normalized Absolute Error (NAE):

$$\mathrm{NAE}(p_U, \hat{p}_U) = \frac{\sum_{j=1}^{|\mathcal{Y}|} |\mathrm{Pr}(y_j) - \hat{\mathrm{Pr}}(y_j)|}{2(1 - \min_{y_j \in \mathcal{Y}} \mathrm{Pr}(y_j))}$$

Some details on the experiments setup

- We subsample and pre-process the RCV1-v2 dataset in order to allow for testing in different scenarios: in fact, we want to test the algorithm both in the binary and the single-label multi-class scenarios;

- We first extract from RCV1-v2 all and only the single-label documents, i.e. the documents which belong to one and only one class (that is, we keep more than 517k documents, and 37 out of 103 target labels);

- For the binary case, we run 500 experiments with all 37 classes, using one at a time as the positive label, and the remaining 36 as the negative examples.

Some details on the experiments setup

- For the single-label dataset, things get a little bit trickier: fix a *n* number of classes out of the 37 remaining;

- Generate two random arrays of probabilities of length *n*, where each element represent the probability of one of the *n* classes. One array is for the training set, the other for the test set;

- Draw a pre-fixed number of documents (1000 in our case) from the dataset, with the probabilities in the random array. We do this twice, once for the training set and a second time for the test set;

- We can now generate lots of subsamples (500 for each classifier in our case) from the original dataset!

So, how does SLD really work?

For every step *s*, we first update our posteriors as a ratio between priors at this step and the classifier priors, multiplied by the classifier posteriors

$$\hat{\Pr}^{(s)}(c_j|x) \leftarrow \frac{\frac{\hat{\Pr}^{(s)}(c_j)}{\hat{\Pr}_t(c_j)}\hat{\Pr}_t(y|x)}{\sum_{j\in C}\frac{\hat{\Pr}^{(s)}(c_j)}{\hat{\Pr}_t(c_j)}\hat{\Pr}_t(c_j|x)}$$

And then...we recompute the priors for next step as the average of the current step posteriors

$$\hat{\Pr}^{(s+1)}(c_j) \leftarrow \frac{1}{N}\sum_{x\in X}\hat{\Pr}^{(s)}(c|x)$$

So, how does SLD really work?

For every step s, we first update our posteriors as a ratio between priors at this step and the classifier priors, multiplied by the classifier posteriors

$$\hat{\Pr}^{(s)}(c_j|x) \leftarrow \frac{\frac{\hat{\Pr}^{(s)}(c_j)}{\hat{\Pr}_t(c_j)} \hat{\Pr}_t(y}{\sum_{j \in C} \frac{\hat{\Pr}^{(s)}(c_j)}{\hat{\Pr}_t(c_j)} \hat{\Pr}_t}$$

And then...we recompute the priors for next step a
step posteriors

$$\hat{\Pr}^{(s+1)}(c_j) \leftarrow \frac{1}{N} \sum_{x \in X} \hat{\Pr}^{(s)}(c|x)$$

Or better, why should it work?

Probability distributions can often change between training and test sets!

Example

- Let's say that due to a huge catastrophe, it's now always winter and it truly rains a lot;
- But you still don't know and have your priors from before, 30% chance of rain. This does not represent the reality anymore!
- If you have posteriors based on observations during this never-ending winter, you could push your previous priors towards a more realistic value (eg. 70%).
- Theoretically, this could work in the other direction as well, for posteriors!

Or better, why should it work?

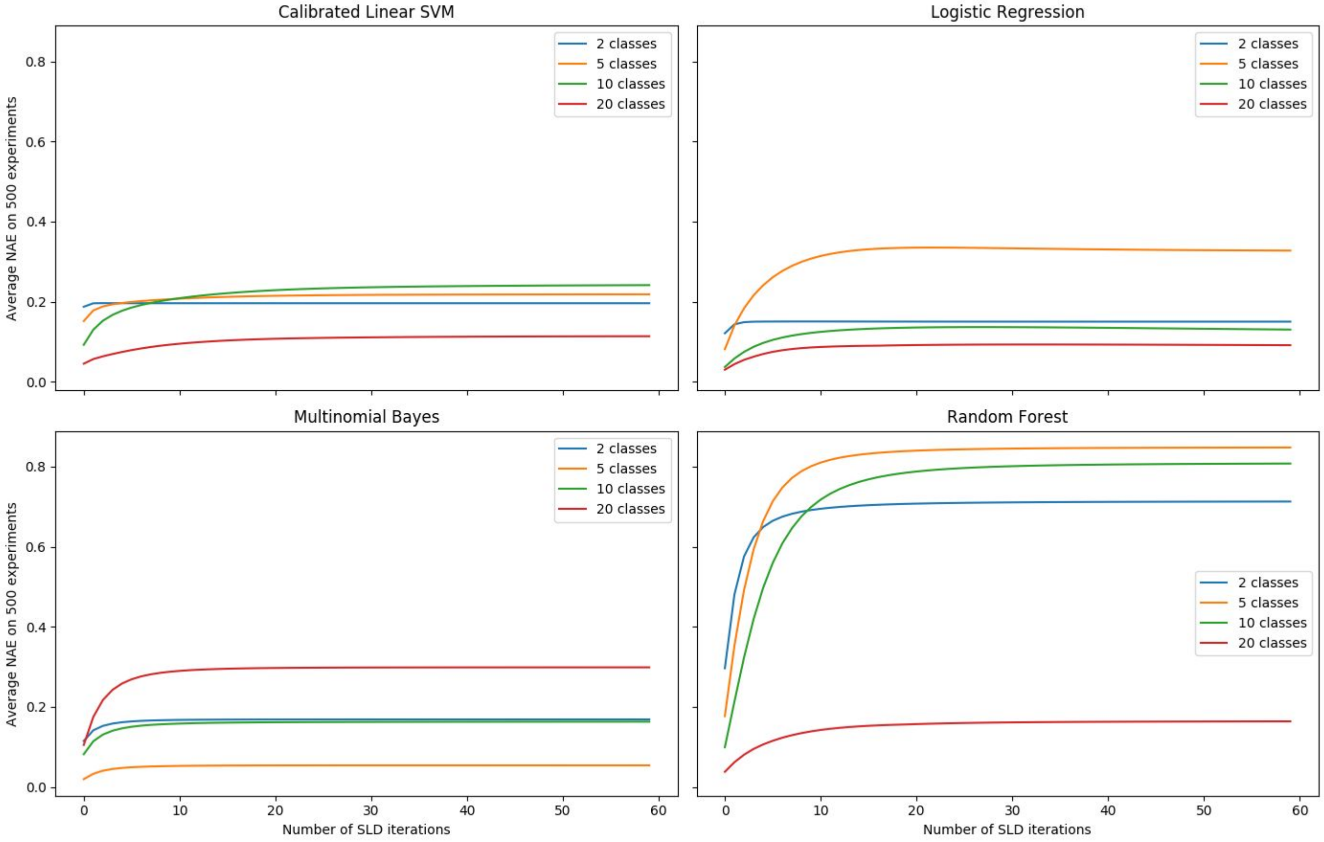
SLD could work really well on a “transductive” dataset, i.e.:

- With “transductive”, in this context, we mean that the dataset is fixed;
- That is, we do not expect to have new documents coming in the future. Yay, overfitting!
- Beware, we do not know the true labels of the set of documents we need to classify (of course), we just know that the dataset have and will keep having a fixed size

Great! But *does* it work?

Not really

Results from 500 experiments on 20ng



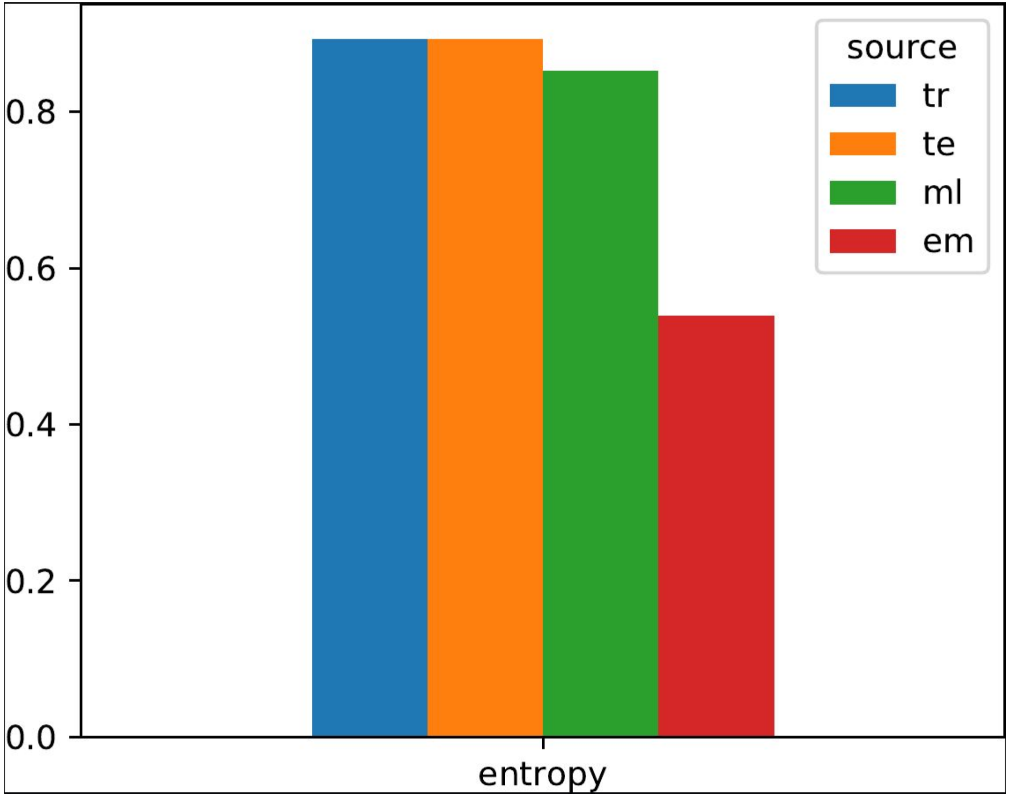
Entropy might be the answer

- In information theory, Entropy can be interpreted as the average level of “information”, “surprise” or “uncertainty” inherent in a random variable’s possible outcomes;

- In our case, this means that if our probabilities are always 1 (i.e. 100% chance of rain), then we are not really surprised to see the rain on any given day. The entropy would be 0;

- Conversely, if our probabilities are always 0.5, we never know what to expect and the entropy would be 1.

Entropy might be the answer



- Average entropy on 500 samples (4 classes);
- SLD (em) completely disrupts the entropy of the distribution;
- The algorithm is basically increasing the probability of one class, reducing probability of the others.

This is still a work in progress

- We are still studying the algorithm and carrying out many analysis. We hope to understand it better and give an answer as to why the algorithm gives such disappointing results;

- However, we are pretty confident SLD is not as good as it was supposed to be;

- It is still often used as one of the baselines in several tasks, eg. quantification.

References

- Saerens, M., Latinne, P., and Decaestecker, C. (2002). Adjusting the outputs of a classifier to new a priori probabilities: A simple procedure. *Neural Computation*, 14(1):21–41.

- Sebastiani, F. (2020). Evaluation measures for quantification: An axiomatic approach. *Information Retrieval Journal.* Forthcoming.

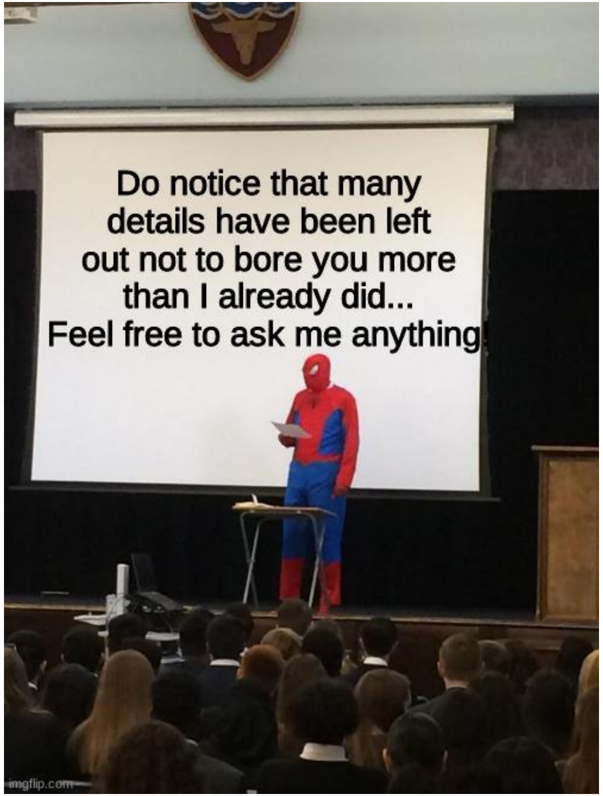
- Zadrozny, B. and Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (KDD 2002)*, pages 694–699, Edmonton, CA.

- Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society,* B, 39(1):1–38.

- DeGroot, M. H. and Fienberg, S. E. (1983). The comparison and evaluation of forecasters. *The Statistician*, 32(1/2):12–22.

- Brier, G. W. (1950). Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78(1):1–3.

Thank you!


Do notice that many details have been left out not to bore you more than I already did...
Feel free to ask me anything!
imgflip.com