Linear Programming Boosting for Uneven Datasets

Jurij Leskovec JURIJ.LESKOVEC@IJS.SI
Jozef Stefan Institute, Ljubljana, Slovenia

John Shawe-Taylor JOHN@CS.RHUL.AC.UK
Royal Holloway University of London, Egham, UK

Abstract

The paper extends the notion of linear programming boosting to handle uneven datasets. Extensive experiments with text classification problem compare the performance of a number of different boosting strategies, concentrating on the problems posed by uneven datasets.

1. Introduction

Boosting is a method of combining so-called weak learners that individually only perform slightly better than a random classifier into a weighted combination that classifies with high accuracy. In general boosting has been shown to exhibit a remarkable resistance to overfitting. An explanation for this phenomenon suggests that this results from boosting optimising the margin of the underlying weighted combination of weak learners [10].

This interpretation has suggested a number of modifications of the underlying boosting strategy through changing the measure of the margin distribution that is optimised [8]. Taking a 1-norm of the slack variables and optimising the 1-norm of the coefficients leads to a linear programme. If this is solved by so-called column generation, the resulting algorithm can be seen as a boosting algorithm [2, 3], where the primal solution gives the weightings of the weak learners and the dual solution the distributions over examples.

The question of how to adapt boosting to handle uneven training sets was considered by Karakoulas and Shawe-Taylor [7]. They introduced the U-boost algorithm that biased the distribution weighting in favour of the examples from the smaller class.

The aim of the current paper is to introduce a version of the linear programming boosting that is tuned for uneven datasets. This is a natural approach to handling uneven datasets as the algorithm optimises a cost criterion that can be adapted to reflect the unevenness of the dataset.

At the same time we present extensive experiments comparing a number of different boosting strategies and methods of weak learner generation. Our main test bed for the experimental work is the Reuters document collection.

2. Boosting algorithms

In this section we introduce the boosting algorithms considered in our experiments, including the adaptation of the linear programming algorithm for uneven datasets.

Boosting is a method to find a highly accurate classification rule by combining many *weak* or *base hypotheses*. Each weak hypothesis may be only moderately accurate. *Weak learners* are trained sequentially. Each weak learner is trained on examples which the preceding weak learners found most difficult to classify.

2.1. AdaBoost

Let $S = (x_1, y_1), \ldots, (x_m, y_m)$ be a set of training examples. Each instance x_i belongs to a domain $\mathcal{X}$ and has assigned a single class y_i. Each class y_i belongs to a finite label space $\mathcal{Y}$. In this paper we focus on binary classification problems in which $\mathcal{Y} = \{-1, +1\}$. We call examples having $y_i = +1$ positive and those having $y_i = -1$ negative examples.

A *weak* learning algorithm accepts as input a sequence of training examples S and a distribution D_t (where $D_t(i)$ could be interpreted as the misclassification cost of the i-th training example). Based on this input the weak learner outputs a weak hypothesis h. We interpret the sign of $h(x)$ as the predicted label and the magnitude $|h(x)|$ as the confidence in that prediction. The parameter α_t is chosen to measure the importance

```
Given $\beta$ and a training set: $S = (x_1, y_1), \ldots, (x_m, y_m)$
where $x_i \in \mathcal{X}, y_i \in \{-1, +1\}$
Initialise: $D_1(i) = \begin{cases} w^+ & \text{if } y_i = +1 \\ w^- & \text{else} \end{cases}$
          where $\frac{w^+}{w^-} = \beta$ and $\sum_{i=1}^m D_1(i) = 1$
For $t = 1, \ldots, T$ :
    Pass distribution $D_t$ to weak learner
    Get weak hypothesis $h_t : \mathcal{X} \to \mathbb{R}$
    Choose $\alpha_t \in \mathbb{R}$
    Update: $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t \beta_i y_i h_t(x_i))}{Z_t}$
    where $\beta_i = \frac{1}{\beta}$ if $y_i = +1$ and 1 if otherwise,
    and $Z_t$ is a normalization factor
Output the final hypothesis: $f(x) = \sum_{t=1}^T \alpha_t h_t(x)$
```

Figure 1. The algorithm AdaUBoost (AdaBoost, if $\beta = 1$)

of the weak hypothesis h_t in the final linear combination of weak hypotheses.

At each round of boosting AdaBoost [11] increases the weights of wrongly classified training examples (i.e. if the signs of h_t and y_i differ) and decreases the weights of correctly classified examples.

2.2. AdaUBoost

A common scenario when learning on imbalanced data sets is that the trained classifier classifies all examples to the major class. In text classification we often encounter this problem when we are learning a binary classifier to separate a small topic from the rest of the documents. In order to avoid this problem, we would like to emphasise the importance of the smaller class. We assume that the smaller class is the positive class while the dominant class is the negative one.

AdaUBoost [7] utilises most of the ideas of AdaBoost [11], but introduces an unequal loss function and modified weight updating rule. The choice of a parameter β will force uneven misclassification costs for training examples (figure 1).

Examples from the smaller, positive, class will initially get assigned β times larger weights than negative examples. The parameter β will also guide the weight updating rule to increase the weight of false negatives more aggressively than of false positives. On the other hand it will decrease the weights of true positives more conservatively than of true negatives. Under this updating rule the weights of positive examples typically maintain higher values. Each weak hypothesis therefore tends to correctly classify more positive examples, since they maintain higher weights. The final hypothesis, a linear combination of the weak hypotheses, will also correctly classify more positive examples.

2.3. LPBoost

LPBoost [5] is a linear program (LP) approach to boosting. The paper [5] shows that taking a 1-norm of the slack variables and optimising the 1-norm of the coefficients leads to a linear programme. If this is solved by so-called column generation, the resulting algorithm can be seen as a boosting algorithm [2, 3], where the primal solutions give the weightings of the weak learners and the dual solutions the distributions over examples.

LPBoost iteratively optimises dual misclassification costs and dynamically generates weak hypotheses to make new LP columns. In contrast to gradient boosting algorithms, which may only converge in the limit, LPBoost converges in a finite number of iterations to a globally optimal solution satisfying well-defined optimality conditions.

One would expect LPBoost to be computationally expensive. We found, however, that an iteration of LPBoost is slightly more expensive than an iteration of AdaBoost, but on the other hand LPBoost needs far fewer iterations than AdaBoost to converge.

2.4. LPUBoost

The LPBoost algorithm [5] is motivated from a generalisation analysis that bounds the test error in terms of the 1-norm of the vector of coefficients, margin achieved and the slack variables. The minimisation of the bound leads to a Linear Programme that gives the formulation above when converted to the dual.

We now give a similar motivation for the uneven version of the LPBoost algorithm, we have named LPUBoost. In this case we assume that the cost of misclassifying a positive example (assuming that the positive class is the less populated) is larger than that of misclassifying a negative example. Hence, the loss $\mathcal{L}_f$ associated with a classification function f is given by

$$\begin{aligned} \mathcal{L}_f(x, y) &= \beta P(f(x) \leq 0 \text{ and } y = 1) \\ &\quad + P(f(x) \geq 0 \text{ and } y = -1), \end{aligned}$$

where as with AdaUBoost $\beta > 1$ gives the higher cost of misclassification of positive examples. We can upper bound this loss as follows

$$\mathcal{L}_f(x, y) \leq \frac{\beta(1+y)}{2\rho}(\rho - f(x))_+ + \frac{1-y}{2\rho}(f(x) + \rho)_+, \tag{1}$$

where $\rho > 0$ and $(\cdot)_+$ denotes the function that is the identity if its argument is greater than 0 and 0 otherwise. We can now apply the Rademacher technique [1]

```
Given $\beta$ and a training set: $S = (x_1, y_1), \ldots, (x_m, y_m)$
where $x_i \in \mathcal{X}, y_i \in \{-1, +1\}$
Initialise: $\alpha = 0$, $n = 0$, $\beta_{\rm LP} = 0$
        $u_i = \begin{cases} w^+ & \text{if } y_i = +1 \\ w^- & \text{otherwise} \end{cases}$
        where $\frac{w^+}{w^-} = \beta$ and $\sum_{i=1}^m u_i = 1$
Repeat
  $n = n + 1$
  Pass distribution $D_t$ to weak learner
  Get weak hypothesis $h_n : \mathcal{X} \to \mathbb{R}$
  Check for optimal solution:
    if $\sum_{i=1}^m u_i y_i h_n(x_i) \leq \beta_{\rm LP}$ then $n = n - 1$, break
  Solve restricted master for new costs:
        $\text{argmin}_{(u, \beta_{\rm LP})}\ \beta_{\rm LP}$
          subject to $\sum_{i=1}^m u_i y_i h_j(x_i) \leq \beta_{\rm LP}$,
           $j = 1, \ldots, n$ and $\sum_{i=1}^m u_i = 1$
          and $D'^+ \leq u_i \leq D^+$, if $y_i = +1$
             $D'^- \leq u_i \leq D^-$, otherwise,
             $i = 1, \ldots, m$
End
$\alpha$ = Lagrangian multipliers from last LP
Output the final hypothesis: $f(x) = \sum_{j=1}^n \alpha_j h_j(x)$
```

Figure 2. The algorithm LPUBoost (LPBoost, if $\beta = 1$)

to bound the loss in terms of its empirical value and the Rademacher complexity of the function class. The Rademacher complexity of a class does not increase if we move to its convex hull and so the bound involves the 1-norm of the slack variables scaled by the inverse margin plus the Rademacher complexity of the weak learners again scaled by the inverse margin. We omit the details because of space constraints. The resulting bound is optimised by the following Linear Programme:

$$\begin{aligned} \min_{\alpha, \rho, \xi^+, \xi^-} & -\rho + D\left(\beta \sum_{i=1}^m \xi_i^+ + \sum_{i=1}^m \xi_i^-\right) \\ & \text{subject to } y_i \sum_{j=1}^n \alpha_j h_j(x_i) \geq \rho - \xi_i^{y_i}, \\ & \xi_i^+, \xi_i^- \geq 0, i = 1, \ldots, m \\ & \alpha_j \geq 0, j = 1, \ldots, n \text{ and } \sum_{i=1}^m \alpha_i = 1. \end{aligned}$$

The parameter D controls the trade off between maximising the margin and controlling the slack variables. Those associated with positive examples receive β times the weight of those from negative examples in line with the bound (1), while the condition $\sum_{i=1}^m \alpha_i = 1$ ensures the resulting function lies in the convex hull of the set of weak learners. The dual Linear Programme of the above is solved by the algorithm LPUBoost of Figure 2.4 with $D^- = D$ and $D^+ = \beta D$.

For LPUBoost we take the same LP formulation of boosting as in LPBoost, but we introduce a new parameter β having the same role as in AdaUBoost. Positive examples have a β times higher bound on their misclassification costs u_i ($\frac{D'^+}{D'^-} = \frac{D^+}{D^-} = \beta$). The choice of parameter D_{LB} controls the lower bound on the misclassification cost u_i. We could set it to 0, but we rather set $D^- = \frac{1}{m\nu}$, where $\nu \in (0, 1)$, and $\frac{D^-}{D'^-} = \frac{D^+}{D'^+} = D_{LB}$. The problem we observed is the sensitivity of LP and LPU boost to the value of parameter ν, which has to be tuned with some care to obtain a good convergence rate.

LPUBoost combines AdaUBoost having an uneven loss function and LPBoost having a well-defined stopping criterion with a mechanism to prevent overfitting. This makes LPUBoost a good algorithm for learning on uneven training sets: maintaining higher weights on positive examples while using an LP to obtain an optimal combination of weak hypotheses with respect to a well-motivated optimisation criterion.

3. Weak learning algorithms

A weak learning algorithm is a procedure for computing weak hypotheses. Boosting finds a set of weak hypotheses by repeatedly calling a weak learning algorithm. The weak hypotheses are linearly combined into a single rule. The input of the weak learning algorithm is a distribution or vector of weights (misclassification costs), and a training data set. Weak learning algorithms use the weights to find a weak hypothesis which has a moderately low error with respect to the weights.

Due to the weight updating rule examples which are hard to classify will get incrementally higher weights while the examples easy to classify get lower weights. The effect is to force the subsequent weak learners to concentrate on hard-to-classify examples.

3.1. AdaBoost.MH class of weak hypotheses

Boosting is a general purpose method that can be combined with any weak learner. In practice it has been combined with a wide variety of classes including decision trees and neural networks.

In this paper we focus on boosting using very simple classifiers. All classifiers considered in this section have the form of a one level decision tree (IF-THEN rule). The test is to check for the presence of a word in a given document. Based on the presence of the word the weak hypothesis outputs a prediction. We interpret the sign of the prediction as a predicted class (recall we are dealing with binary problems) and the magnitude of the output as the confidence in that prediction.

For example: if we try to predict which documents belong to the `Sports` category, we will train a classifier to make a distinction between `Sports` documents (positive class) and the rest of the documents (negative class). Then our weak hypothesis could be a rule: "if the word *football* occurs in a document, then we are highly confident the document belongs to `Sports` category. On the other hand, if *football* does not occur in a document then we predict the document does not belong to the `Sports` category with low confidence."

Formally, we write $w \in x$ to mean a term w occurs in document x. So we define a weak hypotheses h which makes predictions:

$$h(x) = \begin{cases} c_+ & \text{if } w \in x \\ c_- & \text{if } w \notin x \end{cases} \tag{2}$$

where c_+ and c_- are real numbers.

Let us also define: given a current distribution D_t and a term w: let X_+ be a set of documents having the term w, $X_+ = \{x : w \in x\}$, $X_- = \{x : w \notin x\}$ and $b, l \in \{+, -\}$ then we calculate $W_{class}^{has-word}$:

$$W_l^b = \sum_{x_i \in X_b \wedge y_i = l} D_t(i) \tag{3}$$

The weak learners presented in this subsection all have the same form of weak hypotheses, but they impose different restrictions on the values c_+, c_- and α_t and use different criterion to choose a weak hypothesis at each round of learning.

At each round of learning our weak learners search all possible terms. For each term, the values c_+ and c_- and a score are assigned to that particular weak learner. After all terms have been searched, a learner with best score is chosen. Different weak learners use different scores.

For AdaBoost.MH a bound on empirical loss (fraction of misclassified examples) has been proven by Schapire and Singer [11]. They showed that the Hamming loss (MH stands for minimum Hamming loss) of the boosted function obtained using AdaBoost.MH is at most $\prod_{t=1}^{T} Z_t$, where Z_t is a normalization factor at round t. This upper bound can be a used as a guideline for choosing α_t and the design of the weak learning algorithm.

3.1.1. Real AdaBoost.MH

The first algorithm is called AdaBoost.MH (*Real.MH*) with real-valued predictions [12]. We permit c_+ and c_- to be unrestricted real valued predictions.

It was shown in [11] that Z_t is minimised by choosing

$$c_b = \frac{1}{2} \ln\left(\frac{W_+^b}{W_-^b}\right) \tag{4}$$

and setting $\alpha_t = 1$ implies that:

$$Z_t = 2\left(\sqrt{W_+^+ W_-^+} + \sqrt{W_+^- W_-^-}\right) \tag{5}$$

Thus we choose the term w for which Z_t has the minimal value. As suggested in Schapire and Singer [12] we smooth the values of c_b to limit the magnitudes of predictions

$$c_b = \frac{1}{2} \ln\left(\frac{W_+^b + \epsilon}{W_-^b + \epsilon}\right) \tag{6}$$

We use $\epsilon = \frac{1}{m}$. Since W_+^b and $W_-^b \in [0, 1]$, this bounds $|c_b|$ by roughly $\frac{1}{2} \ln(1/\epsilon)$.

3.1.2. Disc AdaBoost.MH

AdaBoost.MH with discrete predictions (*Disc.MH*) forces the predictions c_b of the weak hypothesis to be either $+1$ or -1. This is a more traditional setting where predictions do not carry confidences, and α_t is a measure of confidence in the weak hypothesis. We still minimise Z_t for a given term w. Using the same notation as introduced in the previous section, we set:

$$c_b = sign(W_+^b - W_-^b) \tag{7}$$

We can interpret the choice of c_b as a weighted majority vote over training examples. Let $r_t = |W_+^+ - W_-^+| + |W_+^- - W_-^-|$, then it can be shown [11], that in order to minimise Z_t we should choose:

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 + r_t}{1 - r_t}\right) \tag{8}$$

giving $Z_t = \sqrt{1 - r_t^2}$. So we choose a weak hypothesis (a term w) which has the smallest Z_t.

3.1.3. Disc and Real AdaBoost.LP

The LPBoost algorithm [5] (see Section 2.3) suggests that at each round of boosting a weak hypothesis h with maximal sum of misclassification costs D_t multiplied by class value $y_i \in \{-1, +1\}$ and prediction h:

$$DtSum = \sum_{i=1}^{m} D_t(i) h(x_i) y_i \tag{9}$$

should be chosen. Since this is a different criterion for choosing the best weak-hypothesis in AdaBoost.MH, we obtain two new weak learners. We call them *Real.LP* and *Disc.LP*. They differ from AdaBoost.MH in the way weak hypothesis is chosen, instead of minimising Z_t, $DtSum$ is maximised.

3.1.4. Disc and Real AdaBoost.U

Karakoulas and Shawe-Taylor in their paper on boosting imbalanced training sets [7] proposed a new method for calculating Z_t and choosing α_t.

Note that in case of an uneven loss function (AdaUBoost, LPUBoost) we have an additional parameter β. Positive examples will get β times higher weight than negative examples. The parameter β will also guide the weight updating rule to increase the weight of false negatives more aggressively than of false positives. We define Z_t as:

$$Z_t(\alpha_t) = \sum_{i=1}^{m} D_t(i) \exp\left(-\alpha_t \beta_i h_t(x_i) y_i\right) \tag{10}$$

where $\beta_i = 1/\beta$ if $y_i = +1$ and 1 if otherwise. To minimise the error we seek to minimise Z_t with respect to α_t. By taking the first derivative of (10) and equating it to zero and introducing notation $W_{c,p} = \sum_{i=1}^{m} D_t(i) | y_i = c \wedge h(x_i) = p$, where $c, p \in \{-1, +1\}$, we get: $-\exp(-\alpha_t/\beta)W_{++}/\beta + \exp(\alpha_t/\beta)W_{-+}/\beta + \exp(\alpha_t)W_{+-} - \exp(-\alpha_t)W_{--} = 0$. Substituting $Y = \exp(\alpha_t)$ we obtain:

$$C_1 Y^{1-1/\beta} + C_2 Y^{1+1/\beta} + C_3 Y^2 + C_4 = 0 \tag{11}$$

where $C_1 = -W_{++}/\beta, C_2 = W_{-+}/\beta, C_3 = W_{+-}, C_4 = -W_{--}$. The root of equation (11) can be found numerically. $Z_t''(\alpha_t) > 0$ implies $Z_t(\alpha_t)$ is convex and has only one minimum.

A weak hypothesis h with minimal Z_t is chosen. This is another way of calculating α_t and two new weak learners can be obtained (*Real.U, Disc.U*). They differ from AdaBoost.MH only in the way Z_t is calculated and for discrete learners we set α_t to be the solution of (11) which minimises Z_t.

4. Experimental evaluation

The following sections describe the experimental setup. We also describe and analyse experiments performed using the four boosting algorithms and six text categorization weak learners that were described in the previous sections.

4.1. Experimental setup

We performed empirical evaluation on the ModApte split of the Reuters-21578 dataset compiled by David Lewis. The split consists of 12,902 documents of which 9,603 are used for training and 3,299 for testing.

The following preprocessing was performed: all words were converted to lower case and punctuation marks were removed. We removed stop words from a list of 523 English words. We used the Porter stemmer [9] and retained only those terms having document frequency larger than 3. After the preprocessing the corpus contained 6,242 distinct features (terms).

We compare some of the results with Support Vector Machines [4]. We used the SVM^{light} [6] implementation of SVM with a linear kernel.

Table 1. Test F1 of Reuters categories (with the number of positive training documents). We omit the weak learner used with a certain boosting algorithm.

Category	Ada	U	LP	LPU	SVM
earn (2877)	0.97	0.97	0.91	0.92	0.98
acq (1650)	0.90	0.94	0.69	0.85	0.93
money-fx (538)	0.69	0.73	0.66	0.67	0.77
interest (347)	0.67	0.72	0.61	0.68	0.69
corn (181)	0.80	0.88	0.88	0.90	0.82
gnp (101)	0.77	0.82	0.78	0.82	0.82
carcass (50)	0.56	0.67	0.63	0.63	0.57
cotton (39)	0.82	0.90	0.95	0.95	0.70
meal-feed (30)	0.50	0.80	0.81	0.81	0.47
pet-chem (20)	0.00	0.47	0.00	0.29	0.15
lead (15)	0.44	0.70	0.67	0.75	0.00
soy-meal (13)	0.35	0.80	0.38	0.71	0.26
groundnut (5)	0.00	0.57	0.86	0.86	0.00
platinum (5)	0.00	0.14	1.00	1.00	0.44
potato (3)	0.50	0.80	0.86	0.86	0.00
naphtha (2)	0.00	0.40	0.89	0.89	0.00
Average	0.50	0.71	0.72	0.79	0.48

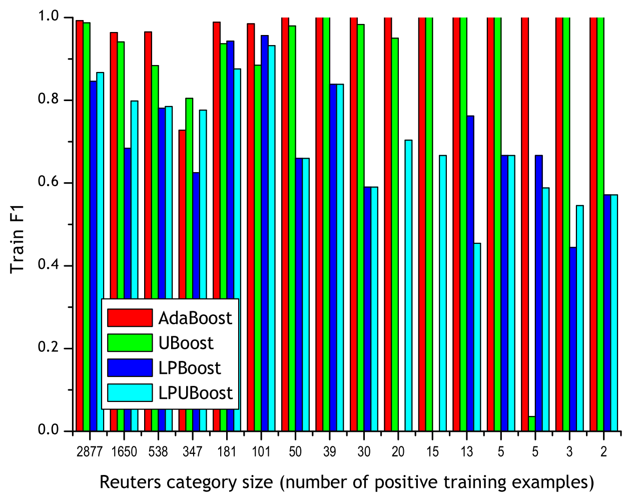
4.2. Performance on Reuters categories

Based on category size we have chosen a set of 16 Reuters-21578 categories. Some of them are large (`earn`, `acq`) and some really small (`potato`, `platinum`) having only a few examples.

We trained boosting binary classifiers to make predictions whether a document belongs to a category or not. We assigned all documents having the category a positive class and all other documents a negative class.

We ran a set of 120 experiments for a single Reuters category using combinations of all the described boosting algorithms and weak learners. In all experiments a number of rounds of learning was set to 300. We tested the combinations of the following parameters: ν: 0.1, 0.2; D_{LB}: 0, 10, 50, 100; β: 2, 4, 8.

For each boosting algorithm we display the best experiment using the standard information retrieval F1 score on the test dataset. We realise that choosing the best performance on the test set invalidates their status. But the aim of this experiment was to get the idea about the best possible performance (upper bound) of various algorithms. Table 1 shows results for chosen categories.



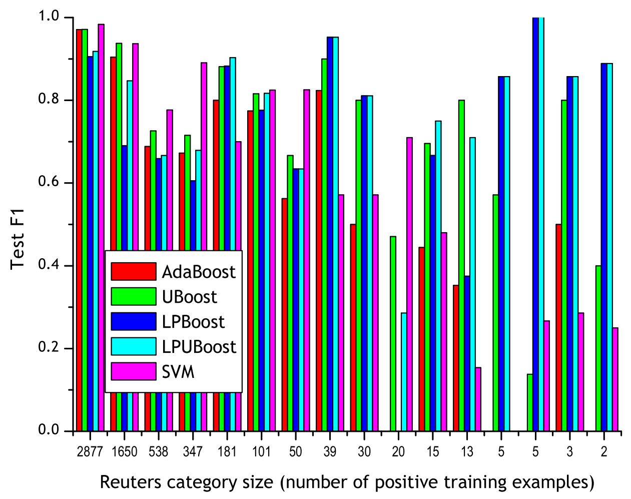


Figure 3. Performance on Reuters categories of different sizes

Table 2. Average test F1 based on 5 fold cross validation.

Category	Ada	U	LP	LPU	SVM
earn (2877)	0.97	0.97	0.97	0.91	0.98
acq (1650)	0.91	0.94	0.88	0.84	0.94
money-fx (538)	0.65	0.70	0.63	0.65	0.76
interest (347)	0.65	0.69	0.59	0.66	0.65
corn (181)	0.81	0.87	0.82	0.83	0.80
gnp (101)	0.78	0.80	0.64	0.66	0.81
carcass (50)	0.49	0.65	0.63	0.63	0.52
cotton (39)	0.68	0.89	0.95	0.95	0.68
meal-feed (30)	0.59	0.77	0.65	0.81	0.45
pet-chem (20)	0.03	0.16	0.03	0.19	0.17
lead (15)	0.20	0.67	0.24	0.45	0.00
soy-meal (13)	0.30	0.73	0.35	0.38	0.21
groundnut (5)	0.00	0.00	0.22	0.75	0.00
platinum (5)	0.00	0.00	0.20	1.00	0.32
potato (3)	0.53	0.53	0.29	0.86	0.15
naphtha (2)	0.00	0.00	0.20	0.89	0.00
Average	0.47	0.59	0.52	0.72	0.46

As we can see from the averages LPUBoost (LPU) is dominant, followed by the LPBoost (LP) and AdaUboost (U). AdaBoost (Ada) and linear SVM are far behind. On large categories Ada, U and SVM are a little better than LP and LPU, but as we move to smaller categories the qualities of LPU (and also LP and U to some extent) seem to appear. AdaUBoost has the feature of uneven loss function which helps and LPBoost has the mechanism to find an optimal combination of weak hypotheses; both these features are combined in LPUBoost.

On small categories Ada and U overfit the training data (figure 3). AdaUBoost is more resistant to overfitting than AdaBoost. On the other hand the performance of LP and LPU on the training set decreases with decreasing category size, but on the test set it remains at about the same level. We can see that LP and LPU are superior to Ada and SVM while U does a little worse than LP and LPU.

Secondly we performed the same set of 120 experiments using stratified 5 fold cross validation. Based on average F1 score over 5 trials, we have chosen best parameter configuration for each algorithm. Table 2 shows the average F1 on test set.

We can see that results obtained by cross validation (table 2) are not far from the optimal (table 1). For algorithms with a small set of parameters (AdaBoost, SVM) the difference between optimal and cross validation performance is small. There is a surprisingly large gap for U and even larger for LP.

SVM performs best of all algorithms on categories with more than 1% of positive training examples, but as we decrease the category size it is no more competitive. LPUBoost performs best and not far from optimal. We also observed that optimal parameter settings are different from those obtained by cross validation. Performance of LPUBoost is quite stable on the whole range of categories of various sizes.

Figure 4 shows typical learning curves of boosting algorithms. In the top row we see the typical performance of AdaBoost on large categories and the oscillations we noticed in AdaUBoost. The bottom row shows LP and LPU. We observe larger jumps in performance than for instance with Ada or U. Typically at the early rounds of learning LP (LPU) is not stable, but when we move forward performance gets more stable, when finally the algorithm converges.

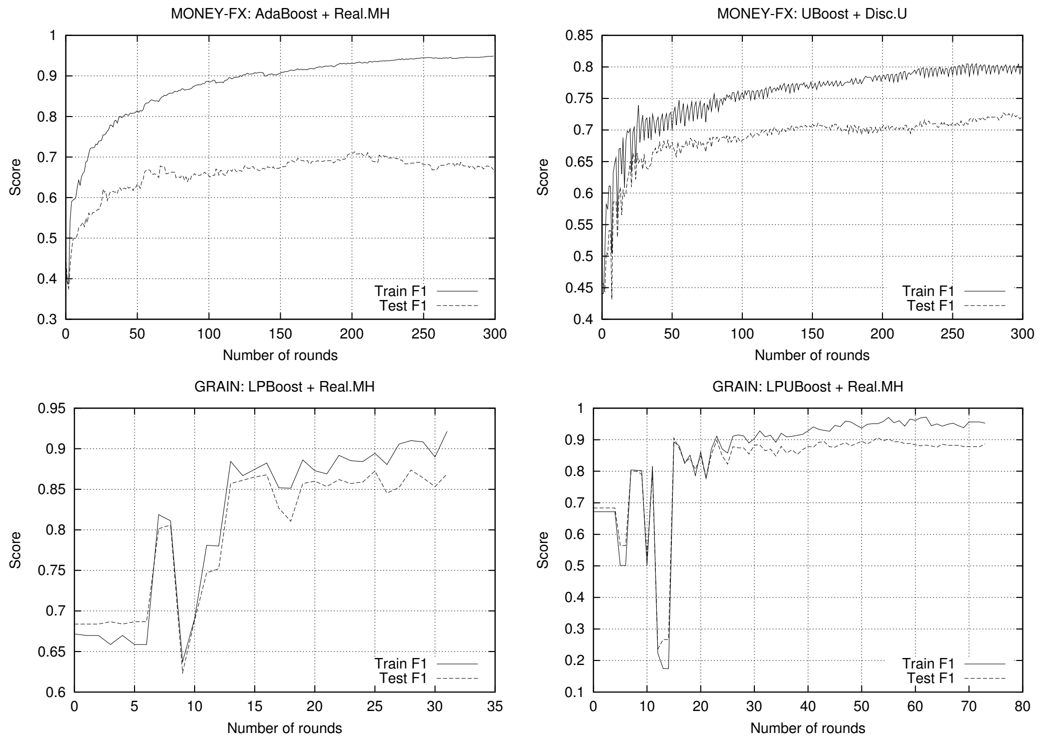


Figure 4. Comparison of various boosting algorithms learning curves

LP and LPU use many fewer weak hypotheses than Ada or U . We trained Ada and U for 300 rounds (300 weak hypotheses were chosen). LP and LPU converged in around 50 rounds for large and around 5 or less rounds of learning for small categories. For some of the smallest categories LPU picked just 1 or 2 weak hypotheses and made no error – for category `platinum` it is sufficient to check for the presence of a word *platinum* in order to make perfect predictions.

Considering chosen weak learners we see Real.MH gives best performance for most of the categories. This is in accordance with the results reported in [12]. Real.MH is followed by Disc.MH and Real.LP. Special weak learners did not improve the performance – not even in combination with the boosting algorithm they were designed to work together.

4.3. Discarding positive training examples

We took 2 largest categories: `earn` and `acq`. Training set consists of all negative and a number of randomly selected positive training examples. Test set is unchanged. By selecting a number of positive training examples we artificially created a small category.

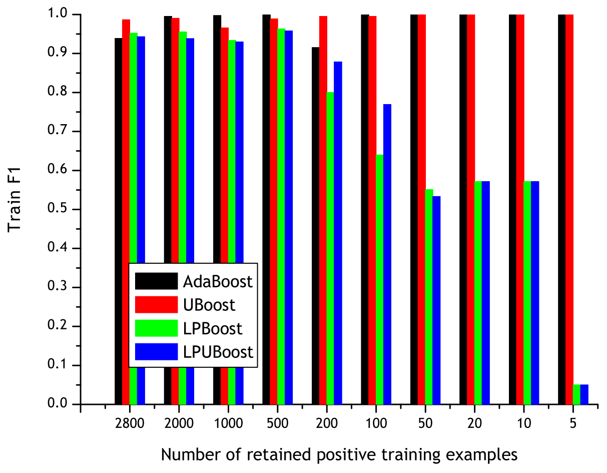
We performed the same set of 120 experiments as in section 4.2. For each boosting algorithm we display the best experiment using the F1 score on the test set. This means we show the upper bound of the algorithm. Figure 5 shows the results which are surprising. Previous experiments showed that LPU is best on uneven training sets, because it does not overfit and picks a sufficiently small number of weak learners. But figure 5 shows a different picture. As the number of positive training examples decreases, the performance of LPU (LP) also dramatically decrease, but the performance of U (Ada) remains almost at the same level.

We observed almost the same things with `acq`: U is still the best, closely followed by LPU. Performance of Ada and LP is poor and after the number of positive training examples drops bellow 50, F1 is less than 0.1.

Since experiment showed that the best possible performance of LP and LPU are far behind from Ada and U, we didn't run stratified cross validation.

LPU (LP) performed very well on naturally uneven datasets. But when we artificially create an small category, the performance of LPU decreased dramatically.

This suggests that there is a fundamental difference between naturally small and artificially small categories. We think that Reuters' editors categorizing documents made `earn` very diverse (broad and not specific), while a small category like `platinum` is very specific. Note that the test set is unchanged so it resembles original category (has the same distribution as original category). To make good predictions using small training



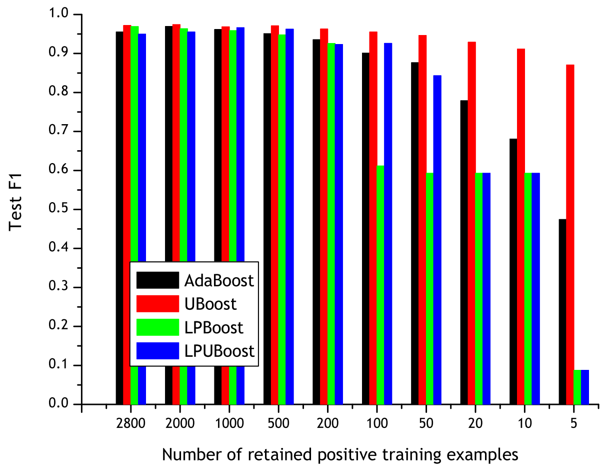


Figure 5. Discarding `earn` positive training examples

(and large diverse testing) `earn` one has to "overfit" by taking all (not necessarily significant) features of the training data. On the other hand we have to be very careful and take only really significant features to make good predictions on `platinum`.

5. Conclusions

This paper introduces LPUBoost boosting algorithm. We provide both theoretical and empirical evidence that LPUBoost is well suited for text categorization for uneven data sets.

LPUBoost has many benefits over gradient-based approaches: finite termination at globally optimal solution, well-defined convergence criteria, unequal loss function and use fewer weak hypotheses.

References

[1] Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2002.

[2] Kristin P. Bennett, Ayhan Demiriz, and John Shawe-Taylor. A column generation algorithm for boosting. In *Machine Learning: Proceedings of the 17th International Conference, ICML'2K*.

[3] Kristin P. Bennett, Ayhan Demiriz, and John Shawe-Taylor. A column generation algorithm for boosting. *Machine Learning*, 46(1):225–254, 2001.

[4] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[5] Ayhan Demiriz, Kristin P. Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1-3):225–254, 2002.

[6] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398, pages 137–142. Springer Verlag, 1998.

[7] Grigoris Karakoulas and John Shawe-Taylor. Optimizing classifiers for imbalanced training sets. In *Proceedings of Neural Information Processing Workshop, NIPS'98*, 1999.

[8] Huma Lodhi, Grigoris Karakoulas, and John Shawe-Taylor. Boosting the margin distribution. In *Proceedings of Intelligent Data Engineering and Automated Learning – IDEAL 2000, Lecture Notes in Computer Science 1983*, 2000.

[9] M. F. Porter. An algorithm for suffix stripping.

[10] R. Schapire, Y. Freund, P. Bartlett, and W. Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, 1998.

[11] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Computational Learing Theory*, pages 80–91, 1998.

[12] Robert E. Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.