Low-Dimensional Feature Learning with Kernel Construction

Artem Sokolov
LIMSI-CNRS, Orsay, France
artem@limsi.fr

Tanguy Urvoy
Orange Labs, Lannion, France
tanguy.urvoy@orange.com

Hai-Son Le
LIMSI-CNRS, Orsay, France
lehaison@limsi.fr

Abstract

We propose a practical method of semi-supervised feature learning with constructed kernels from combinations of non-linear weak rankers for classification applications. While in kernel methods one usually avoids working in the implied implicit feature space, we use the outputs of weak rankers as new features, and define the kernel as scalar product in the new feature space. The kernel is then used to map high-dimensional data into a low-dimensional space keeping the mapping informative enough to be used as training data for learning algorithms. We evaluate and compare the proposed method with other approaches on a public dataset released during the recent Semi-Supervised Feature Learning Challenge [1].

1 Introduction

Contemporary technology generates abundant, multi-modal and essentially multi-dimensional data flows and requires its rapid processing. One such typical domain is Web information retrieval, where data sources can be characterized by millions of features (e.g., occurring words, phrases, grammatical tagging), can be described by diverse modalities (particular user's preferences, page history, trustworthiness, expected click rate or page advertisement revenue), and are exceptionally plentiful. Because of data size, their direct storage is infeasible or impermissibly expensive. Even more, storing a fixed subset of relevant (for a particular task) features may not be possible, as not every task to be performed with data can be thought of in advance. Thus there is a need to transform the input space into a lower-dimensional one, such that the new features remain sufficiently informative and expressive to allow learning on them and guarantee satisfactory performance on testing. Learning such representations is complicated by the cost and difficulty of acquisition of the supervision information, so the transformation design should make use of unlabeled data that is cheaply available.

One can view a learned model of data as a reduction itself, as it is possible to use the output of the trained model as a one-dimensional feature. In general, however, one would be most interested in preserving, in the reduced representation, the information sufficient for learning a good-performing model for several tasks (binary or multi-class classification, ranking, regression etc.) and several measures of quality. For sequential learning systems it may also be useful to preserve a "not-too-specialized summary" of the past in order to adapt the learned model to changes in the data stream on the fly. Thus, a useful definition of the semi-supervised (reduced) feature learning task would depend on the learning algorithm used on the transformed data, the task(s) in question and the quantity optimized. Although highly desired, such over-generalized task is obviously difficult to tackle. A reasonably simple, yet interesting, task formulation in a form of competition was proposed during Semi-Supervised Feature Learning (SSFL) Challenge [1]:

Challenge settings: Let I be the set of instances of the challenge dataset. For each instance $i \in I$, a feature vector $\boldsymbol{x}_i = (x_i^1, \ldots, x_i^D) \in \mathcal{X}$ was provided. The $\mathcal{X}$ set was high-dimensional ($\mathcal{X} \subset \mathbb{R}^D$ with $D = 10^6$) and sparse ($\max_{i \in I} |\boldsymbol{x}_i|_0 = 414$).

Most instances were unlabeled except two separate subsets I_{train} and I_{test} used, respectively, for training and testing. For each training instance i, only a noisy label $y_i \in \{-1, +1\}$ was provided. This noise was injected to simulate reality and favor using unlabeled instances. The real labels for train and test were kept secret by the organizers.

The goal was to use both labeled and unlabeled data to construct a dimension reduction mapping $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y} \subseteq \mathbb{R}^d$ with $d = 100 \ll D$. The new feature space had to be rich enough and informative, to allow a classifier to be trained on $f(\mathcal{X})$ with the best possible predictive performance. According to the challenge's rules, the classifier had to be a standard linear Support Vector Machine (C-SVM) trained on $f(\mathcal{X})$ according to training labels with a fixed error-penalty parameter [2]:

$$\boldsymbol{w}_f^* = \begin{cases} \underset{\boldsymbol{w},b,\boldsymbol{\xi}}{\text{argmin}} & \frac{1}{2}\langle \boldsymbol{w}; \boldsymbol{w} \rangle + \sum_{i \in I_{train}} \xi_i \\ \text{subject to} & y_i \cdot (\langle \boldsymbol{w}; f(\boldsymbol{x}_i) \rangle + b) \geq 1 - \xi_i, \;\; \xi_i \geq 0. \end{cases} \tag{1}$$

Let $c(\boldsymbol{x}) = \langle \boldsymbol{w}_f^*; f(\boldsymbol{x}) \rangle$ be the classifier output function obtained by combination of the dimension reduction mapping f and its optimized SVM model $\boldsymbol{w}_f^*$. The performance of this classifier was measured by its Area Under ROC Curve (AUC). For $i, j \in I_{test}$ the AUC of c is given by the Wilcoxon-Mann-Whitney statistic:

$$\text{AUC}_c = \frac{1}{\sum_i [\![y_i = -1]\!] \cdot \sum_j [\![y_j = +1]\!]} \sum_{i: y_i = -1} \sum_{j: y_j = +1} [\![c(\boldsymbol{x}_i) \geq c(\boldsymbol{x}_j)]\!], \tag{2}$$

and is equal to the probability that the value of c on a randomly chosen negative example is lower than that on a randomly chosen positive example [3]. Training of the SVM was performed by the entrants on the public train set, and the leaderboard AUC was computed by the organizers on the test set. One may notice a side-effect of this evaluation process: any multi-dimensional reduction mapping can be replaced by its combination with its associated optimal SVM model, and since the two models are evaluated on the same set of labels, they will result in exactly in the same AUC score. In other words: *constructing 100 features in order to train a prediction model is a more general (harder) problem than building directly a prediction model.* Some possible ways to avoid this in future challenges are discussed in conclusion.

Contribution In this paper we propose a practical method of semi-supervised dimensionality reduction and report on its and other (supervised, semi-supervised, and unsupervised) methods' performance in the SSFL Challenge.

Our method is trifold. First, a kernel function K is learned on labeled instances I_{train} that leverages the label information to tune itself for similarity between instances. To learn K we propose two alternatives – a RankBoost algorithm and a neural network. RankBoost algorithm [4] linearly combines weak learners, each depending on one or two input features, to minimize a pairwise ranking loss function. We consider the found weak non-linear learners to be mappings of the corresponding original feature(s) to coordinates in a new high-dimension feature space. Making data to be linearly separable in the new space is what tries to achieve RankBoost by minimizing the pairwise ranking loss. The actual kernel K is then simply defined as an inner product in the new feature space.

An alternative approach to obtain kernel K is to model it with a multi-layer neural network, trained to minimize an approximation of the kernel alignment measure of similarity [5] on a data sample between K and the perfect kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = y_i y_j$.

Secondly, having our kernel constructed, we take advantage of the unlabeled data. Let I_{sample} be a random subset of I of size r. The data-points $\boldsymbol{x}_k$ such that $k \in I_{sample}$ are used as pivot points for kernel K's evaluations to embed training and testing points into a intermediate feature space $\mathcal{K}$:

$$\boldsymbol{x} \mapsto (K(\boldsymbol{x}_{k_1}, \boldsymbol{x}), K(\boldsymbol{x}_{k_2}, \boldsymbol{x}), \ldots, K(\boldsymbol{x}_{k_r}, \boldsymbol{x}))\,, \; k_i \in I_{sample}. \tag{3}$$

Finally, vectors in $\mathcal{K}$ are randomly projected in an oblivious manner onto a low-dimensional space $\mathcal{Y}$ using a binary variant of Johnson-Lindenstrauss lemma [6]. If the learned kernel K manages to correctly classify training data with a non-zero margin γ the final two steps are guaranteed to secure separability in $\mathcal{K}$ and further in $\mathcal{Y}$ as shown in [7].

The paper is organized as follows. First, in section 2 we review related work that we make use of to construct our reduction. In section 3 we describe the details of our approach to kernel learn-

ing. Finally, in section 5, we compare the proposed approach with several other supervised, semi-supervised and unsupervised learning methods that we tested during the SSFL Challenge [1].

2 Related Work

2.1 Space embeddings

Switching host space or space embedding [8] can greatly simplify distance calculations and/or speed up nearest neighbor queries [9]. Efficient dimensionality reduction and sketching techniques, based on random projections, have been already developed for Euclidean [10, 6] and general ℓ_p distances [11] as well as for cosine similarity [12]. For example, the classic Johnson-Lindenstrauss lemma [10] tells that an Euclidian space ℓ_2 can be randomly projected with a Gaussian matrix R onto ℓ_2 of dimension $O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta})$ such that with probability at least $1-\delta$ the distance between any pair of point in the new space is within a factor of $1\pm\varepsilon$ of their original distance:

$$(1-\varepsilon)\left\|\boldsymbol{x}_1-\boldsymbol{x}_2\right\| \leq \left\|R\boldsymbol{x}_1-R\boldsymbol{x}_2\right\| \leq (1+\varepsilon)\left\|\boldsymbol{x}_1-\boldsymbol{x}_2\right\|. \tag{4}$$

Same guarantees also hold for a uniform $\{-1,+1\}$-valued random matrix R [6].

This kind of dimensionality reduction is a naïve approach to the feature learning – although it reduces the dimension, it is unconscious of the learning task behind. However, we will use it as an ingredient for the feature construction method of [7] (section 2.2), and as one of the baselines in section 4.

2.2 Kernels for Feature Learning and Kernel Alignment

Unlike to the traditional dimensionality reduction algorithms, kernel methods [2] non-linearly map data into higher dimensionality in order to, with more degrees of freedom, find a separating hyperplane with no or fewer errors than in the original space. On the other hand, similarly to the dimensionality reduction methods, successful application of kernel methods hinges on the ability of the non-linear implicit mapping to capture the inherent similarity of the data vectors. So, as selecting an appropriate kernel is crucial to the performance of the final classifier, it is important to have a kernel (pre)selection procedure without the need to test the whole family of kernels.

For a data sample $I_{sample} \in I$ and kernel product $K_1 \cdot K_2 = \sum_{i,j\in I_{sample}} K_1(\boldsymbol{x}_i,\boldsymbol{x}_j)K_2(\boldsymbol{x}_i,\boldsymbol{x}_j)$, kernel alignment was introduced in [5] as cosine between kernel matrices unfolded into a vector:

$$A(K_1,K_2) = K_1\cdot K_2/\sqrt{(K_1\cdot K_1)(K_2\cdot K_2)}. \tag{5}$$

In [5] it was shown that a kernel K that maximizes its alignment with perfect kernel $K(\boldsymbol{x}_i,\boldsymbol{x}_j) = y_i y_j$ has good generalization properties.

If, for an appropriately selected kernel, the implicit space $\mathcal{F}$ has such good properties, it is tempting to apply Johnson-Lindenstrauss lemma to map $\mathcal{F}$ into a space of a more practical dimension. It is indeed possible to do in a two-stage process as shown in [7]. First, it can be shown that for a kernel with non-zero margin, sufficiently large random sample I_{sample} from a unlabeled data distribution and the mapping (3) to space $\mathcal{K}$, there exists an approximate linear separator vector $\boldsymbol{w'}$, that linearly separates the distribution under mapping (3) with a small error in $\mathcal{K}$.

Secondly, decompose further $K(\boldsymbol{x}_i,\boldsymbol{x}_j)_{ij\in I_{sample}}$ into a Cholesky decomposition $U^{\mathrm{T}}U$, where U is an upper-triangular matrix. In $\mathcal{K}$, optionally make an orthogonal projection onto the span of the vectors from the random sample with U (which we call "whitening"), followed by random projection with binary Johnson-Lindenstrauss lemma [6], we obtain with probability at least $1-\delta$ a $(1\pm\varepsilon)$-embedding (4) to a low-dimensional space of dimension $O(\frac{1}{\gamma^2}\log\frac{1}{\varepsilon\delta})$, where data remains linearly separable with margin $\gamma/4$ [7]. Here, the final random oblivious projection improves the guarantees on the margin size, compared to direct embedding to a low-dimensional space with the mapping (3).

3 Learning Kernels

In this section we describe two methods of constructing a kernel: one based on an explicit non-linear mapping obtained by applying RankBoost ranking algorithm and another based on a direct approximation of the optimal kernel with a neural network. After the kernel is constructed, we directly

apply the "black-box" semi-supervised method of [7] to build an informative and low-dimensional feature representation, as described in section 2.2.

3.1 RankBoost Kernel

The first of the proposed methods of building a kernel is to use the explicit non-linear feature mapping produced by the pair-wise ranking algorithm RankBoost [4]. For a given number of training steps T the RankBoost algorithm learns a scoring function H, which is a linear combination of "simple" functions h_t called weak learners:

$$H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}), \tag{6}$$

where each α_t is the weight assigned to the weak function h_t at step t of the learning process. RankBoost learns H by minimizing a convex approximation of weighted pair-wise loss $L_P(H)$:

$$L_P(H) = \sum_{\substack{i,j \in I_{train} \\ y_i < y_j}} P(i,j)[\![H(\boldsymbol{x}_i) \geq H(\boldsymbol{x}_j)]\!] \leq \sum_{\substack{i,j \in I_{train} \\ y_i < y_j}} P(i,j) e^{H(\boldsymbol{x}_i) - H(\boldsymbol{x}_j)}. \tag{7}$$

We used two families of weak learners – the simplest decision stumps that depend on one feature:

$$h(\boldsymbol{x}; \theta, k) = [\![x^k > \theta]\!], \tag{8}$$

where k is the selected feature index and θ is a learned threshold, and decision grids, that combine values of two features to trigger a non-zero output value:

$$h(\boldsymbol{x}; \theta_1, k_1, \theta_2, k_2) = [\![x^{k_1} > \theta_1]\!] \cdot [\![x^{k_2} > \theta_2]\!].$$

The stump learners h were trained using the approximate "3-rd method" described in [4], the grid learners – by a straight-forward generalization of the same method.

The positively-valued preference matrix P in (7) encodes the orderings observed in the training set: the higher the value of $P(i,j)$ is, the more important it is to preserve the relative ordering of he two instances i, j and in the learning process of the values of P are updated to allow concentrating on examples that have not been correctly classifier so far [4]. If P is chosen uniform, the loss becomes the Kendall τ and minimizing it is equivalent to maximizing AUC for the so called bipartite ranking problem – when labels are binary [13, 14]. As AUC was the evaluation measure for the SSFL Challenge and labels were binary we naturally chose $P(i,j) = \frac{y_j - y_i}{Z_P}$, where $Z_P = \sum_{i,j \in I_{train} \,:\, y_i < y_j} (y_j - y_i)$ is a global normalization factor.

Consider the set of weak learners that were selected and that participate in the final scoring function (6). From this set we can build an explicit mapping $\boldsymbol{\Phi} : \mathcal{X} \to \mathcal{F} = [0,1]^T$ by setting:

$$\boldsymbol{\Phi}(\boldsymbol{x}) = (\alpha_1 h_1(\boldsymbol{x}), \ldots, \alpha_T h_T(\boldsymbol{x}))\,. \tag{9}$$

For the $P(i,j)$ defined above, the loss (7) is the number of wrongly ordered pairs in a linear classification task for a fixed hyperplane vector $\boldsymbol{w} = (1, 1, \ldots, 1) \in [0,1]^T$. Indeed, for such vector $\boldsymbol{w}$, we have $H(\boldsymbol{x}) = \langle \boldsymbol{w}; \boldsymbol{\Phi}(\boldsymbol{x}) \rangle$ and by setting $y = sgn(\langle \boldsymbol{w}; \boldsymbol{\Phi}(\boldsymbol{x}) \rangle)$ the RankBoost algorithm would chose weak learners in a way to force the image of training data under $\boldsymbol{\Phi}$-mappings to be separable for hyperplane vector $\boldsymbol{w}$ and maximize AUC. Mapping (9) can be shaped into an alternative form for decision stumps (8), where it can be viewed as non-linear distortion of the coordinate axes of $\mathcal{X}$ with the same aim of making data separable. For this rewrite H as: $H(\boldsymbol{x}) = \sum_{d=1}^{D} \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}) [\![\kappa(h_t) = d]\!]$, where $\kappa(h_t)$ gives the acting feature's index k for the learner h_t, and define the coordinate mappings Φ'_k of $\boldsymbol{x}$ to weighted sums of the outputs of the corresponding learners: $\Phi'_k : \boldsymbol{x} \to \sum_{t \,:\, \kappa(h_t) = k} \alpha_t h_t(\boldsymbol{x})$. After establishing this relation to linear classification, it is natural to define a kernel K as inner product on vectors $\boldsymbol{\Phi}(\boldsymbol{x}) \in \mathcal{F}$:

$$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \langle \boldsymbol{\Phi}(\boldsymbol{x}_1); \boldsymbol{\Phi}(\boldsymbol{x}_2) \rangle.$$

A logic question to ask would be about the relation of the performance of RankBoost to the performance of a SVM with the obtained kernel. To provide some intuition, we mention hinge rank loss

(a close ranking analog of the hinge loss function, implied by SVM optimization task (1)), which was introduced in [15], and where its minimization was proven to maximize AUC asymptotically. Experimentally, though, it was already known that classic C-SVM with hinge loss are excellent AUC optimizers ([16, 17] and [15] with references therein). On the other hand, maximizing mean AUC (averaged over tasks with a fixed classification error) was shown to minimize the classification error [13] for the bipartite ranking problem, and even that for threshold weak rankers the mean AUC equals the observed AUC [13]. Hence, at least with empirical support, one may expect that the $\mathbf{\Phi}$ feature representation found by optimizing RankBoost loss (7) (and AUC) would be also advantageous for finding a large margin separator hyperplane $\boldsymbol{w}$ under hinge loss.

In the same time, for the special case when RankBoost finds a perfect model achieving zero loss (7) and AUC $= 1$, it will do it in space $\mathcal{F}$ with margin

$$\gamma = \frac{1}{2} \min_{\substack{i,j \in I_{train}: \\ y_i=-1, y_j=+1}} \langle \boldsymbol{w}; \mathbf{\Phi}(\boldsymbol{x}_j) \rangle - \langle \boldsymbol{w}; \mathbf{\Phi}(\boldsymbol{x}_i) \rangle = \frac{1}{2} \min_{\substack{i,j \in I_{train}: \\ y_i=-1, y_j=+1}} H(\boldsymbol{x}_j) - H(\boldsymbol{x}_i),$$

Then, an SVM is bound to find a separating plane with a margin at least as big, and a zero hinge loss.

Finally, one might argue that choosing a strongly related boosting technique AdaBoost [18], designed specifically for classification, might be a better choice, instead of ranking-targeted RankBoost. However, as shown in [14], AdaBoost and RankBoost losses are equal if the AdaBoost's example weighting function P is the same for examples from two classes.

3.2 Learning Aligned Kernels with Neural Networks

The second method we tried was to learn a simple neural network optimizing an approximation to the kernel alignment (5). While the original work on kernel alignments involves resource-heavy optimization of semi-definite programming task [5], we tried to use a multi-layer neural network to learn an alignment similarity function between vectors. Similarly to [19], we were unable to optimize (5) directly because of the normalization factor, and minimized quadratic loss instead:

$$L_e = \sum_{\boldsymbol{x}_i, \boldsymbol{x}_j} (K(\boldsymbol{x}_i, \boldsymbol{x}_j) - y_i y_j)^2. \tag{10}$$

Three network architectures were tested (Figure 1). Two input sparse vectors $\boldsymbol{x}_i, \boldsymbol{x}_j,\ i, j \in I_{train}$ are propagated to the one or two hidden layers and finally to the single output node representing the target similarity value $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. All layers use hyperbolic tangent as non-linear activation function. We learn the models with the stochastic gradient descent with back-propagation. For each example pair $\boldsymbol{x}_i, \boldsymbol{x}_j$, in the forward phase, we compute the output value $K^f(\boldsymbol{x}_i, \boldsymbol{x}_j)$. In the backward phase, the gradient value at the output $2(K^f(\boldsymbol{x}_i, \boldsymbol{x}_j) - y_i y_j)$ is propagate back to the input layer. Finally, all network parameters β are updated like: $\beta_{t+1} = \beta_t + \alpha \frac{\partial L_e}{\partial \beta}$, where α is a learning rate found by 5-fold cross validation. As the approach gives in general a non-symmetric function K, we find the actual similarity function by symmetrization: $K_{sym}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{1}{2}(K(\boldsymbol{x}_i, \boldsymbol{x}_j) + K(\boldsymbol{x}_j, \boldsymbol{x}_i))$. Function K_{sym} is not guaranteed to be positive semi-definite (PSD), however, we did not investigate the practical effects of the PSD property absence because of the tight schedule of the Challenge. A way to enforce the property might be untangling internal layers for the inputs, and forcing the output layer form to be $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{s=1}^{S} \phi_s(\boldsymbol{x}_i)\phi_s(\boldsymbol{x}_j)$, where ϕ_s are S functions to learn.

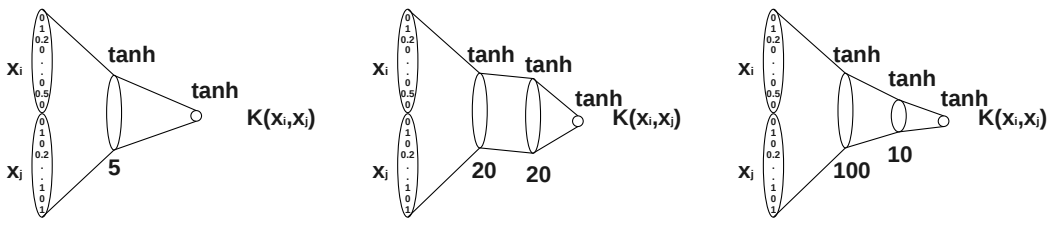

Figure 1: Structure of the kernel alignment optimizing neural network.

4 Baseline and Other Methods

100 k-means The baseline provided by organizers was obtained by k-means algorithm with 1000 mini-batches [20], 100 centroids trained on all instances I and 10000 iterations. Each test example was mapped to 100 features, using the mapping (3), with RBF kernel with kernel parameter 0.01.

1000 k-means and neural dimension reduction The previous baseline was extended to 1000 centroids, and a neural network was used to learn a dimension reduction to 100 in the supervised way. Concretely, the network had 1000 nodes in the input layer, 100 nodes in the hidden layer and 1 node as the output with a sigmoid as activation function to represent the probability of class +1. After training the model, we used the 100 output values of the hidden layer as reduced representation.

1000 k-means and random projection To compare with the previous method we included as baseline a direct application of the random projection (section 2.1) to the 1000 k-means centroids.

Direct application of RankBoost After the challenge, we evaluated the performance of RankBoost scoring function as a single feature. With $T = 5000$ stumps, it obtained slightly better results than the stumps-kernel method. On the other hand, the kernel method is slightly better for grids.

Single-feature sparse logistic regression According to the Challenge setting it was possible – and legal – to build a single-dimension mapping. Surprisingly, a simple sparse (ℓ_1-regularized) logistic regression trained with the `liblinear`[1] obtained a respectable AUC. A slightly worse performance was obtained with the ℓ_2-regularization, what can be explained by that the ℓ_1-regularization tends to select sparser models and is thus more robust to the label noise.

Sparse logistic regression augmented by k-means To test a simple method of semi-supervised classification we concatenated k-means features (learned in a unsupervised manner on the unlabeled data) with the original features. We tested this method for $k = 200$ and $k = 800$ injected features.

Logistic regression by neural dimension reduction Original sparse data was input to a neural network with one linear hidden layer with 100 neurons and a single soft-max output neuron. In this way we obtain a logistic regression model, as in previous two methods, with the advantage of having a reduced intermediate representation on the hidden layer to train on.

Logistic regression and graph kernel smoothing Another method for semi-supervised learning is to regularize the scores $\hat{y}_i$ obtained by an inductive model (sparse logistic regression) using neighborhood in a similarity graph built on the whole dataset. A similar approach was used in [21] to enhance spam detection on a Web graph. The loss to minimize is:

$$Loss(\boldsymbol{z}) = \underbrace{\sum_{i \in I} (z_i - \hat{y}_i)^2}_{\text{consistency}} + \lambda \underbrace{\sum_{i,j \in I} w_{i,j} \cdot (z_i - z_j)^2}_{\text{smoothness}},$$

where $w_{i,j}$ is a similarity weight between instances and λ is a trade-off parameter. Weights $w_{i,j}$ were estimated by sampling with Charikar's cosine fingerprints [12]. Tuning of λ was time consuming; because of the lack of time we could not evaluate thoroughly the validity of this approach. We evaluated also a simple neural network counterpart to such graph kernel smoothing by labeling the unlabeled data with the logistic regression neural network described above using threshold 0.5 and retraining the network on the obtained data set.

5 Experiments

The SSFL dataset [1], introduced in section 1, was crawled for web-classification purposes. It consists of 3 sets: a binary labeled train set I_{train} (50K examples), a test set I_{test} of the same size and a set of 1M unlabeled instances. Each instance is a 1M-dimensional sparse vector, with on average 115 simultaneously active features; about 200K features were active at least once, 88% of which are binary. Some noise was introduced in the training labels to stimulate the use of the unlabeled data. Participants had to come up with a transformation of the input space to a vector

[1] www.csie.ntu.edu.tw/∼cjlin/liblinear/

Table 1: AUC of baseline methods on public, private subsets and on the whole dataset.

baseline method	30% public	70% private	**100% total**
100 k-means, organizers' baseline	0.98050	0.98867	0.98308
1000 k-means, random projection	0.98195	0.98565	0.98455
1000 k-means, neural dimension reduction	0.98408	0.98798	0.98683
RankBoost, stumps, 5000 steps	-	-	0.99613
RankBoost, grids, 2000 steps	-	-	0.99495
sparse logistic regression ($C = 0.22$)	0.99021	0.98841	0.99576
logistic regression + 200 k-means ($C = 0.2$)	**0.99470**	**0.99630**	**0.99631**
logistic regression + 800 k-means ($C = 0.2$)	-	-	0.99620
logistic regression with neural network	0.99192	0.99440	0.99366
log. reg. + graph smoothing ($C = 0.38$, $\lambda = 0.1$)	0.99378	0.99460	0.99493
log. reg. with neural net on labeled unlabeled data	0.98289	0.98554	0.98475

Table 2: AUC of RankBoost and neural kernels on public, private subsets and on the whole dataset.

kernel	T	I_{sample}	whitening	30% public	70% private	**100% total**
stump	5000	1000	no	0.98050	0.98016	0.98026
stump	5000	1000	yes	0.99139	0.99314	0.99262
stump	2270	1000	no	0.99084	0.99295	0.99232
stump	5000	5000	no	0.99264	0.99346	0.99322
stump	5000	5000	yes	0.99098	0.99248	0.99203
stump	2270	5000	no	-	-	0.991664
stump	2270	10000	no	0.99231	0.99346	0.99311
grid	2000	1000	no	0.99419	0.99547	0.99509
grid	2000	1000	yes	0.99254	0.99430	0.99378
grid	1150	1000	no	0.99365	0.99543	0.99490
grid	2000	5000	no	**0.99422**	**0.99597**	**0.99546**
grid	2000	5000	yes	0.99379	0.99561	0.99507
neural (layer 5)		1000	no	0.98901	0.98962	0.98945
neural (layers 20-20)		1000	no	-	-	0.98865
neural (layers 100-10)		1000	no	-	-	0.98718
neural (layer 5)		5000	no	0.99060	0.98991	0.99012
neural (layers 20-20)		5000	no	-	-	0.99279
neural (layers 100-10)		5000	no	-	-	0.99223

space of dimensionality at most 100, such that a binary C-SVM classifier (with C=1.0)[2] is capable of finding a "good" separating plane. The "goodness" of the transformation was measured by AUC (2), preliminary on the public subset (30%) of the test set and, finally, on the private test subset (70%).

The evaluation showed a clear superiority of the more powerfully grid learner over the less flexible simple decision stumps. Increasing the size of the random sample I_{sample} improves performance of the classifier on the final reduced data, which can be attributed to better covering of the unlabeled data and lowering the probability of error δ (section 2.2) and/or the distance estimation error ε in (4).

Selecting the number of iterations T with cross validation (5-fold in our case) was not useful, possibly because of the introduced label noise in the training data. The lower than baseline performance of the kernel trained with the neural network is, most likely, because of the simplicity of the chosen network architecture and ignoring the normalization factors of the kernel alignment (5) in the quadratic loss function (10). It remains unclear if "whitening" improves quality. While the step is justified on theoretical ground [7] (see also [22], where a similar transform is used for kernelized nearest-neighbor queries), more experiments (for different $|I_{sample}|$) are necessary to decide on this, as opposite effects were observed for stumps, while for grids it usually worsened performance.

We also separately evaluated each stage of the proposed trifold method. Let **A** be the RankBoost features (9), **B** – projection onto unlabeled data (3), **C** – random projection (section 2.1) and **D** – the final SVM learning stage. We write **A**$\rightarrow$**D** for SVM training directly on Rankboost features,

[2]Finally, the requirement to use SVM was not enforced.

Table 3: AUC values for the separate evaluation of the method's stages.

kernel	T	I_{sample}	**A➛D**	**A➛B➛D**	**A➛C➛D**	**A➛B➛C➛D**
stump	2270	1000	0.99539	0.99280	0.99517	0.99232
stump	2270	5000	0.99539	0.99296	0.99517	0.99166
grid	2000	1000	0.99076	0.99518	0.99416	0.99378
grid	2000	5000	0.99076	0.99517	0.99416	0.99546

Table 4: Mean AUC values and variances for two stages over different samples of I_{sample}.

kernel	T	I_{sample}	**A➛B➛D**	**A➛B➛C➛D**
stump	2270	100	0.99156±0.00152	0.99350±0.00169
stump	2270	1000	0.99257±0.00029	0.99245±0.00040
stump	2270	5000	0.99267±0.00014	0.99254±0.00040
grid	2000	100	0.99497±0.00047	0.99512±0.00028
grid	2000	1000	0.99522±0.00038	0.99504±0.00026
grid	2000	5000	0.99502±0.00042	0.99260±0.00071
neural (layers 20-20)		1000	0.99565±0.00003	0.98934±0.00066
neural (layers 100-10)		1000	0.99550±0.00004	0.98860±0.00214

A➛B➛D for method application without random projection and so on. Evaluations of several stage chains are given in Table 3 with variances in Table 4 (over different samples of pivots). Although the point of step **C** is to have larger margin [7] and satisfy the output dimension constraint of the Challenge, in practice it reduces AUC (degrading the "signal" quality of the **B** step) and increases variance, compared to the **A➛B➛D** chain for large I_{sample}. Using simply $|I_{sample}|$ =100 without step **C**, however, can incur large variance, because of too small, hence instable, samples (see stumps in Table 4). Another explanation of the degrading could be that the artificial label noise makes train and test data originate from different distributions, contrary to the assumptions of [7].

6 Conclusion

We presented a practical method for semi-supervised low-dimensional feature learning, by first constructing a kernel using RankBoost non-linear feature transformations and using it to build low-dimensional, yet informative, feature representations. The method uses both labeled and unlabeled data and achieves a higher value of AUC compared to the method optimizing kernel alignment and the various supervised and semi-supervised baseline methods on the same data. The only baseline method that performed better was a one-dimensional sparse logistic regression, that is not suited for learning rich feature representations, what was the main goal aim of the proposed method.

While the proposed method uses theoretically motivated random unlabeled samples as pivot points in the intermediate embedding, it would be interesting to empirically verify the potential of using non-random (e.g., learned) pivots. The k-means baseline method, that is also based on (3) and uses learned pivots (centroids), performed competitively and indicates that this might have sense.

As mentioned in the introduction, a more realistic competition setting should evaluate reduced representations on several tasks, like multi-class classification (with different data slicing according to modalities or facets), class-dependant penalties, ranking and regression tasks etc. Training data was largely sufficient for a powerful supervised method to outperform semi-supervised methods, and although label noise was attempted to make the labeled data less informative, providing less instances could be another way to emphasize semi-supervised learning as opposed to the fully supervised one. A way to favor real feature learning submissions might be to cross-validate them by retraining SVM on the transformed private dataset. One may expect the "multi-purpose" reductions to be more robust to label drifts than specialized ones; this can also be thought of as "stacked learning" [23].

Acknowledgments

We thank organizers of the SSFL Challenge for the great opportunity to evaluate several feature learning methods and three anonymous reviewers for their helpful remarks. This work has been partially funded by OSEO under the Quaero program.

References

[1] D. Sculley. Results from a deep learning and semi-supervised feature learning competition. In *NIPS 2011 Workshop on Deep Learning and Unsupervised Feature Learning*, Granada, Spain, 2011.

[2] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. 1 edition, 2000.

[3] Tom Fawcett. An introduction to ROC analysis. *Pattern Recogn. Lett.*, 27:861–874, 2006.

[4] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, 2003.

[5] Nello Cristianini, Jaz Kandola, Andre Elisseeff, and John Shawe-Taylor. On kernel-target alignment. In *Advances in Neural Information Processing Systems 14*, pages 367–373, 2002.

[6] Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66:671–687, 2003.

[7] Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. Kernels as features: On kernels, margins, and low-dimensional mappings. In *Proc. of the Int. Conf. on Alg. Learning Theory*, pages 79–94, 2004.

[8] Piotr Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Proc. of the IEEE symposium on Foundations of Computer Science*, pages 10–, DC, USA, 2001.

[9] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. of the ACM symposium on Theory of computing*, pages 604–613, 1998.

[10] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Struct. Algorithms*, 22:60–65, 2003.

[11] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. of the Symposium on Computational geometry*, pages 253–262, NY, USA, 2004.

[12] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. of the Annual ACM symposium on Theory of computing*, pages 380–388, NY, USA, 2002.

[13] Corinna Cortes and Mehryar Mohri. AUC optimization vs. error rate minimization. In *Proc. of NIPS*, 2004.

[14] Cynthia Rudin and Robert E. Schapire. Margin-based ranking and an equivalence between AdaBoost and RankBoost. *J. Mach. Learn. Res.*, 10:2193–2232, 2009.

[15] Harald Steck. Hinge rank loss and the area under the roc curve. In *Proc. of the European Conf. on Machine Learning*, pages 347–358, Berlin, Heidelberg, 2007.

[16] Ulf Brefeld and Tobias Scheffer. AUC maximizing support vector learning. In *Proc. ICML workshop on ROC Analysis in Machine Learning*, 2005.

[17] Thorsten Joachims. A support vector method for multivariate performance measures. In *Proc. of the Int. Conf. on Machine Learning*, pages 377–384, 2005.

[18] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proc. of the European Conf. on Computational Learning Theory*, pages 23–37, London, UK, 1995.

[19] Koby Crammer, Joseph Keshet, and Yoram Singer. Kernel design using boosting. In *Advances in Neural Information Processing Systems 15*, pages 537–544, 2003.

[20] D. Sculley. Web-scale k-means clustering. In *Proc. of the Int. Conf. on WWW*, pages 1177–1178, NY, USA, 2010.

[21] Artem Sokolov, Tanguy Urvoy, Ludovic Denoyer, and Olivier Ricard. MADSPAM consortium at the ECML/PKDD Discovery Challenge 2010. In *Proc. of ECML/PKDD Discovery Challenge Workshop*, Barcelona, Spain, 2010.

[22] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE Int. Conf. on Computer Vision*, 2009.

[23] Kai Ming Ting and Ian H. Witten. Stacked generalization: when does it work? In *Proc. of the Int. Joint Conf. on Artificial Intelligence*, pages 866–871, 1997.