# The Pairwise Piecewise-Linear Embedding for Efficient Non-Linear Classification

**Ofir Pele, Ben Taskar** OFIRPELE,TASKAR@CIS.UPENN.EDU
University of Pennsylvania, Department of Computer and Information Science, Philadelphia, PA 19104 USA

**Amir Globerson, Michael Werman** GAMIR,WERMAN@CS.HUJI.AC.IL
The Hebrew University of Jerusalem, School of Computer Science, Jerusalem 91904 Israel

## Abstract

Linear classifiers are much faster to learn and test than non-linear ones. On the other hand, non-linear kernels offer improved performance, albeit at the increased cost of training kernel classifiers. To use non-linear mappings with efficient linear learning algorithms, explicit embeddings that approximate popular kernels have recently been proposed. However, the embedding process is often costly and the results are usually less accurate than kernel methods. In this work we propose a non-linear feature map that is both very efficient, but at the same time highly expressive. The method is based on discretization and interpolation of individual features values and feature pairs. The discretization allows us to model different regions of the feature space separately, while the interpolation preserves the original continuous values. Using this embedding is strictly more general than a linear model and as efficient as the second-order polynomial explicit feature map. An extensive empirical evaluation shows that our method consistently outperforms other methods, including a wide range of kernels. This is in contrast to other proposed embeddings that were faster than kernel methods, but with lower accuracy.

## 1. Introduction

Several recent works have proposed algorithms for learning linear classifiers in nearly linear time (Joachims, 2006; Shalev-Shwartz et al., 2011; Tsang et al., 2005). These have allowed learning to scale to very large training sets. However, when using kernels such efficient implementations are no longer possible. One approach which can be used to overcome this is to "linearize" kernels, by representing them explicitly in their feature space. Although in some cases (*e.g.* RBF kernels) the feature space is infinite dimensional, approximations are possible. Indeed, in recent works several embeddings that approximate known kernels or give their exact form have been proposed (Chang et al., 2010; Maji et al., 2012; Perronnin et al., 2010; Rahimi & Recht, 2007; Vedaldi & Zisserman, 2012). Most of these methods approximate specific kernels such as the RBF. Maji et al. (2012) is an exception where an approximation of any additive kernel is proposed using single feature discretization and linear interpolation (although the motivation was to approximate the histogram intersection kernel). This enables learning an approximation of any additive model with a time complexity that is linear in the dimension of the input.

Following Maji et al. (2012), we suggest an embedding of an input vector $\vec{x}$ into a high dimensional but sparse vector $\vec{\alpha}(\vec{x})$ and then learning a linear classifier in this representation. We suggest a simple and effective improvement over Maji et al. (2012), namely the addition of second order terms that approximate cross-feature relationships. The resulting linear classifier is the sum of piecewise linear functions in the individual $\vec{x}[n]$ and in pairs $\vec{x}[n], \vec{x}[l]$ (thus highly non-linear in the original representation). Our main contribution is an extensive experimental study that shows that this simple method achieves excellent performance.

## 2. Problem Setup

For simplicity we focus on a binary classification task. The generalization to multi-class or regression tasks is straightforward. Let $\{\vec{x}_i, y_i\}_{i=1}^{M}$ denote a set of $M$ ex-

*Proceedings of the 30$^{th}$ International Conference on Machine Learning*, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. 

amples with inputs $\vec{x}_i \in \mathbb{R}^N$ and labels $y_i \in \{-1, +1\}$. We also use $\vec{x}_i[j]$ to denote the value of the $j^{\text{th}}$ feature of the $i^{\text{th}}$ input.

In standard linear classification, prediction is performed by taking the sign of a linear function of the input $\vec{x}$. A much more general approach, which we follow here, is to use a *piecewise linear* dependence. Specifically, we assume that the prediction is given by $\text{sign}(F(\vec{x}; \vec{w}))$ where $\vec{w}$ are parameters (to be learned) and:

$$F(\vec{x}; \vec{w}) = \sum_{n=1}^{N} f_n(\vec{x}[n]; \vec{w}) + \sum_{n=1}^{N} \sum_{l=n+1}^{N} f_{n,l}(\vec{x}[n], \vec{x}[l]; \vec{w}) \tag{1}$$

where $f_n(\vec{x}[n]; \vec{w})$ are piecewise linear functions in the feature $\vec{x}[n]$ and $f_{n,l}(\vec{x}[n], \vec{x}[l]; \vec{w})$ are piecewise linear functions in the pairs of features $\vec{x}[n], \vec{x}[l]$.

The piecewise linear functions are constructed as follows. We begin by describing the single variable case. For each variable $\vec{x}[n]$, we define a mapping $\vec{\alpha}_n : \mathbb{R} \to \mathbb{R}^D$ (see Section 3) and the function $f_n(\vec{x}[n]; \vec{w})$ that is linear in $\vec{\alpha}_n$, namely:

$$f_n(\vec{x}[n]; \vec{w}) = \vec{w}_n \cdot \vec{\alpha}_n(\vec{x}[n]) \tag{2}$$

where $\vec{w}_n \in \mathbb{R}^D$ is a sub vector of $\vec{w}$.

Similarly, for the pairwise functions we define a mapping $\vec{\alpha}_{n,l} : \mathbb{R}^2 \to \mathbb{R}^{D^2}$ and the function:

$$f_{n,l}(\vec{x}[n], \vec{x}[l]; \vec{w}) = \vec{w}_{n,l} \cdot \vec{\alpha}_{n,l}(\vec{x}[n], \vec{x}[l]) \tag{3}$$

where $\vec{w}_{n,l} \in \mathbb{R}^{D^2}$ is a sub vector of $\vec{w}$.

Thus, overall we have: $F(\vec{x}, \vec{w}) = \vec{w} \cdot \vec{\alpha}(\vec{x})$, where $\vec{w}$ and $\vec{\alpha}$ are the concatenation of all $\vec{w}_n, \vec{w}_{n,l}$ and $\vec{\alpha}_n(\vec{x}[n]), \vec{\alpha}_{n,l}(\vec{x}[n], \vec{x}[l])$ vectors respectively. In other words, $F$ is a *linear* function in the non-linear functions $\vec{\alpha}$. Thus, to learn $\vec{w}$ we can use standard training methods used for linear SVMs, only with input features $\vec{\alpha}$ (see Section 3.2).

In the next section we explain the construction of the $\vec{\alpha}$ functions and the structure of the resulting piecewise linear functions. As we shall see, the functions $f_n$ and $f_{n,l}$ separate the single and pairwise feature space into a set of discrete bins. Their value at the edges of the bins can be determined arbitrarily, and between the bins linear interpolation is used. This allows us to learn highly non-linear models.

Our method can exactly model a linear function of the features. Additionally, we can exactly model similarity or dissimilarity of feature pairs (corresponding to symmetries and anti-symmetries in the data). The resulting learned model is continuous in feature space but highly expressive due to its different treatment of different feature value regions.

The project homepage including code is at: `http://www.seas.upenn.edu/~ofirpele/PL2/`.

## 3. The Pairwise Piecewise-Linear Embedding

As mentioned earlier, our method works by embedding single features and pairs of features into a vector space using functions $\vec{\alpha}_n(\vec{x}[n])$ and $\vec{\alpha}_{n,l}(\vec{x}[n], \vec{x}[l])$. In what follows, we explain how this embedding is constructed, and what is the resulting structure of the $f_n$ and $f_{n,l}$ functions.

We begin by explaining the construction for single feature functions $f_n(\vec{x}[n]; \vec{w})$. The key idea is to partition the value of the feature $\vec{x}[n]$ into $D$ bins, and then model $f_n(x; \vec{w})$ as a linear function in each bin, with a continuous interpolation between the bins. Thus, we begin by finding $D$ *representative* values of the features $\vec{x}[n]$, and denote their values by $\vec{v}_n \in \mathbb{R}^D$. In practice, we set $\vec{v}_n[1]$ to the minimum value of $\vec{x}[n]$ (observed in the data), $\vec{v}_n[D]$ to the maximum,[1] and the other values to the $k$-means centers (with $k = D - 2$) of the observed values of $\vec{x}[n]$. We assume $\vec{v}_n$ is sorted.

Our goal is to construct $\vec{\alpha}_n$ such the value of $f_n(\vec{x}[n]; \vec{w})$ at one of the discrete points in $\vec{v}_n$ can be arbitrarily determined using $\vec{w}_n$, and the values at other $\vec{x}[n]$ are linear interpolation. Thus, we would like $f_n(\vec{v}_n[i]; \vec{w}) = \vec{w}_n[i]$ and linear interpolation otherwise. It is easy to construct an embedding $\vec{\alpha}_n(\vec{x}[n])$ that satisfies this. We call this embedding function PL1, and describe it in Alg. 1. Note that $\vec{\alpha}_n(\vec{x}[n])$ will have at most two non-zero values (out of $D$ possible values), and it is thus sparse.

We now turn to the pairwise case, and the construction of $\vec{\alpha}_{n,l}$. We use the $\vec{v}_n$ and $\vec{v}_l$ discretization above, to discretize the pair of features into a 2D grid, with $D^2$ points. Accordingly, the vectors $\vec{\alpha}_{n,l}$ and $\vec{w}_{n,l}$ are in $\mathbb{R}^{D^2}$. As before, we would like the points on the lattice to be mapped to arbitrary values, such that $f_{n,l}(\vec{v}_n[i], \vec{v}_l[j]; \vec{w}) = \vec{w}_{n,l}[k]$ where $k$ is the appropriate index in the weight vector (i.e., $k = (i-1)D + j$). The values at other points $\vec{x}[n], \vec{x}[l]$ should be an interpolation of the discrete points.

[1] At test time, values of $\vec{x}[n]$ that are larger than the maximum or smaller than the minimum are clipped to the maximum or minimum respectively. Other schemes are possible, but we found this to work well in practice

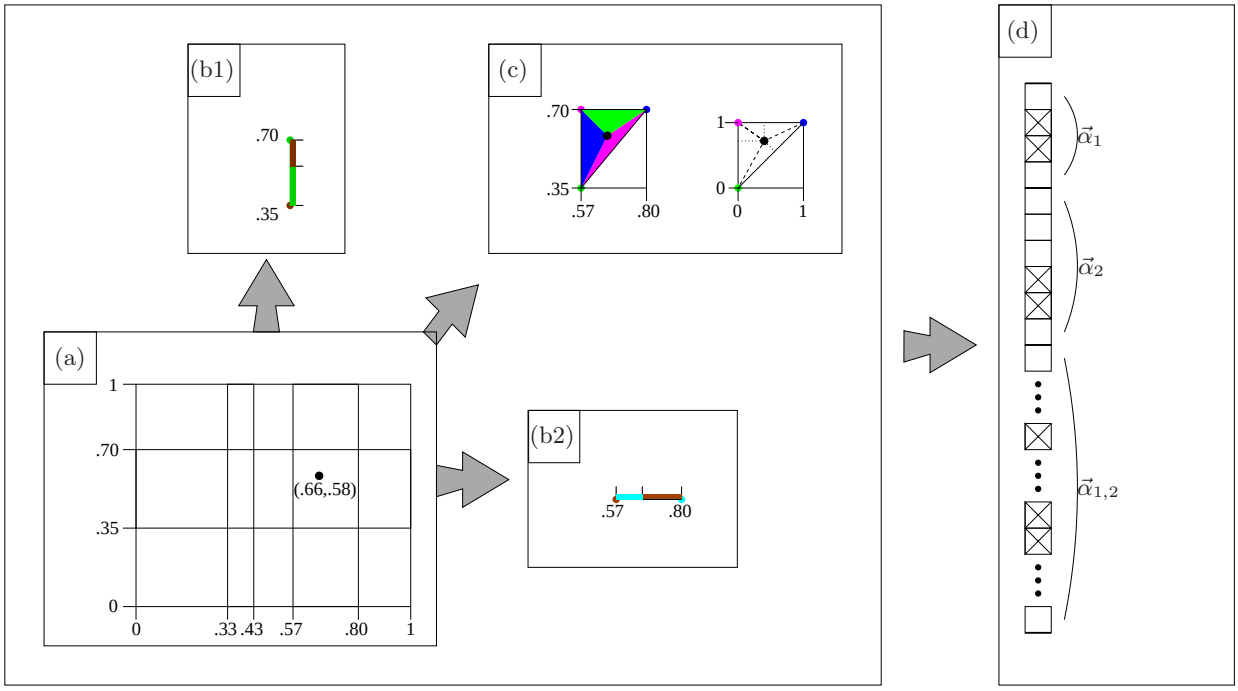


*Figure 1.* A flow diagram of the method for a two-dimensional vector. In (a) we see the grid which is the result of the discretization. We also see a new vector to be embedded: (.66,.58) symbolized by a circle. In (b1) and (b2) we see the PL1 embedding which assigns weights to the two bounding discretization points in inverse linear proportion to the distance from them. The resulting embedding are the sparse vectors $\vec{\alpha}_1$,$\vec{\alpha}_2$ in (d) (x symbolizes a non-zero entry). In (c) we see the illustration of the pairwise embedding. We see the rectangle that bounds the features pair where it is bisected with a diagonal line from bottom left to top right. The barycentric coordinates are the normalized triangle areas. Each triangle corresponds to a features values pair symbolized by a colored circle, where corresponding triangles and circle have the same color. As the normalized triangle areas are invariant to the aspect ratio of the rectangle, we can linearly map the rectangle to a unit rectangle in (c) and get that the normalized triangle areas are equal to the heights of the triangles in (c). The resulting embedding is the sparse vector $\vec{\alpha}_{1,2}$ in (d).

To achieve this, we use an interpolation scheme using three coefficients, given by the PL2 algorithm in Alg. 2 below. The coefficients are barycentric coordinates on triangles. Using this interpolation results in a continuous embedding, which is also differentiable except at the discretization points. This means, for example, that the PL2 embedding is not very sensitive to the number of discretization points chosen. The resulting embedding vector is high dimensional but sparse.

A flow diagram of the approach for a two-dimensional vector is given in Fig. 1 and the algorithm is described in Algs. 1, 2. Alg. 1 starts with trimming the feature value $x$ to be in the range $[\vec{v}[1], \vec{v}[\text{length}(\vec{v})]$. Next it finds the interval $[\vec{v}_{\hat{d}}, \vec{v}_{\hat{d}+1}]$ that contains the feature value $x$. Finally, it linearly maps the value $x$ to the two "buckets" $\vec{\alpha}[\hat{d}]$ and $\vec{\alpha}[\hat{d}+1]$ of the found interval. Alg. 2 starts by applying Alg. 1 on the two features. Then it linearly maps the pairwise value to the four discrete points surrounding it in the 2D grid (Fig. 1(a),(c)).

As we use linear interpolation, we are able to exactly model any linear model of the original feature values. Lines 3, 6 and 10 in Alg. 2 show that we can also model exactly dissimilarity or similarity of features pairs, allowing us to catch symmetries or anti-symmetries in data (see Fig. 2). Other cross-feature relationships that can modeled exactly are the maximum (lines 3,4,8) and minimum (lines 3,5,9). But most importantly, using the discretization we can approximate any single and cross-feature Lipschitz continuous relationships to a desired accuracy using a suf-

**Algorithm 1** $[\hat{d}, \hat{x}, \vec{\alpha}] = \text{PL1}(x, \vec{v})$

1: $\hat{x} = \min(\max(x, \vec{v}[1]), \vec{v}[\text{length}(\vec{v})])$
2: $\hat{d} = \max_{d \in [1 \ldots (\text{length}(\vec{v})-1)]} \text{s.t: } \hat{x} \geq \vec{v}[d]$
3: $\hat{x} = \frac{\hat{x} - \vec{v}[\hat{d}]}{\vec{v}[\hat{d}+1] - \vec{v}[\hat{d}]}$
4: $\vec{\alpha}[\hat{d}] = 1 - \hat{x}$
5: $\vec{\alpha}[\hat{d}+1] = \hat{x}$

**Algorithm 2** $[\vec{\alpha}] = \text{PL2}(x_1, x_2, \vec{v}_1, \vec{v}_2)$

**Define:** I(i,j) = (i-1)D+j

1: $[\hat{d}_1, \hat{x}_1, \vec{\alpha}_1] = \text{PL1}(x_1, \vec{v}_1)$
2: $[\hat{d}_2, \hat{x}_2, \vec{\alpha}_2] = \text{PL1}(x_2, \vec{v}_2)$
3: **if** $\hat{x}_1 < \hat{x}_2$ **then**
4: $\quad \vec{\alpha}_{1,2}[I(\hat{d}_1, \hat{d}_2)] = 1 - \hat{x}_2$
5: $\quad \vec{\alpha}_{1,2}[I(\hat{d}_1 + 1, \hat{d}_2 + 1)] = \hat{x}_1$
6: $\quad \vec{\alpha}_{1,2}[I(\hat{d}_1, \hat{d}_2 + 1)] = \hat{x}_2 - \hat{x}_1$
7: **else**
8: $\quad \vec{\alpha}_{1,2}[I(\hat{d}_1, \hat{d}_2)] = 1 - \hat{x}_1$
9: $\quad \vec{\alpha}_{1,2}[I(\hat{d}_1 + 1, \hat{d}_2 + 1)] = \hat{x}_2$
10: $\quad \vec{\alpha}_{1,2}[I(\hat{d}_1 + 1, \hat{d}_2)] = \hat{x}_1 - \hat{x}_2$
11: **end if**
12: $\vec{\alpha} = [\vec{\alpha}_1, \vec{\alpha}_2, \vec{\alpha}_{1,2}]$

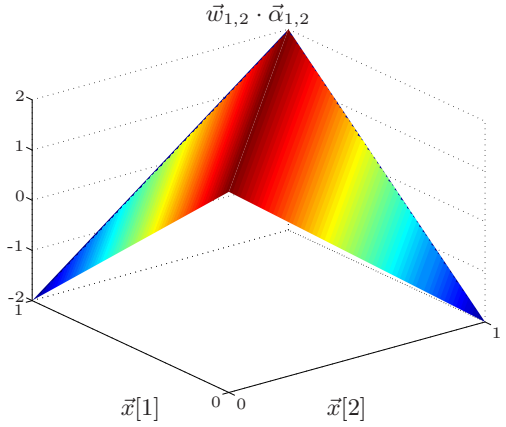


*Figure 2.* Visualization of a function $f_{1,2}(\vec{x}[1], \vec{x}[2]); \vec{w})$ for the $D = 2$ case, and discretization $\vec{v}_1, \vec{v}_2 = \{0, 1\}$. Here $\vec{w}_{1,2} \in \mathbb{R}^4$ has a value 2 for the grid points $[0, 0]$ and $[1, 1]$ and $-2$ for $[0, 1]$ and $[1, 0]$.

ficient number of bins.

The $N$-dimensional algorithm applies Alg. 1 on each of the dimensions and cache the results. Then it applies lines 3-11 in Alg. 2 for all pairs of features and finally concatenates all the $\vec{\alpha}_n$ and $\vec{\alpha}_{n,l}$ vectors.[2]

### 3.1. Time Complexity

We now describe the time complexity for calculating the $\vec{\alpha}$ embeddings. The PL1 embedding has time complexity $O(N \log D)$, where the log is due to the search for upper bounds in line 1 of Alg. 1. The PL2 embedding has time time complexity $O(N^2)$, since we can cache the results of the PL1 part.

In total, the time complexity of the embedding is $O(N \log D + N^2)$. That is, although we increase the number of learned parameters by a factor of $O(DN)$ the time complexity of the embedding compared to using a linear kernel increases by only a factor of $O(N + \log D)$. Since typically $\log D \ll N$, it follows that the PL2 embedding is as efficient as explicitly calculating a second degree polynomial embedding. Finally, we note that in some cases we can decrease the time complexity to $O(N \log D + NK)$ by modeling relationships only between neighboring features, where $K$ is the average number of features neighbors.

If we have prior knowledge that zero values should not contribute to the decision function we can easily change the algorithm to be sparsity preserving. Then, the time complexity will depend only on the average number of non-zero entries in the vectors. This is an interesting direction for future work.

[2]The maximum number of $\vec{\alpha}_{n,l}$ vectors is $\binom{N}{2}$, but as mentioned in Section 3.1 this may be reduced to $NK$ in some cases.

### 3.2. Learning

For learning we use the binary or multi-class (Crammer & Singer, 2002) support vector machine (SVM) with $l_2$ regularization. We note that our embedding is high dimensional but very sparse. Stochastic subgradient methods such as Pegasos (Shalev-Shwartz et al., 2011) are a natural choice for efficient sparse embedding as the embedding can be done on the fly and the time complexity depends on the number of non-zeros entries and not on the dimension. Embedding on the fly also means that our memory usage is extremely small as only one embedded example is held in memory for each subgradient step. An extensive discussion on memory consumption of embeddings that contains pairwise terms is given in Chang et al. (2010). In order to isolate the effect of the kernel / embedding used, when we compared to kernels methods such as RBF we used the LIBSVM solver (Chang & Lin, 2011).

## 4. Related Work

In this section we describe other works that have suggested embedding input vectors into a higher dimension and then learning a linear classifier in this representation. Additionally, we discuss a new metric learning method that uses the interpolation described here.

The $\gamma$-homogeneous $\chi^2$ embedding was proposed by

Vedaldi & Zisserman (2012). The $\chi^2$ kernel between two scalars $x, y \in \mathbb{R}$ is: $K(x,y) = 2\frac{xy}{x+y}$. A kernel is $\gamma$-homogeneous for two scalars if: $\forall c \geq 0 : k(cx, cy) = c^\gamma k(x,y)$. The $\chi^2$ kernel is 1-homogeneous. It is possible to generalize it to $\gamma$-homogeneous using a scalar function that fully characterizes the kernel (Vedaldi & Zisserman, 2012). For vectors, these kernels are the sum of scalar kernels of each of the dimensions.

An example of second order terms modeling is the second order polynomial kernel. Chang et al. (2010) conducted an extensive study on the usage of the second order polynomial explicit feature map. Their study shows that for some dataset, training and testing using the second order polynomial *explicit* feature map is much faster than using the kernel trick. However, accuracy is slightly lower than RBF. Our method not only significantly outperformed state-of-the-art embeddings but also the RBF kernel on several datasets while being much faster in training and testing. A major problem with RBF is that it is not robust to irrelevant features. Additive kernels are robust to irrelevant or noisy features but cannot capture feature dependencies which might be important for classification. Our method can approximate some feature dependencies while being robust to irrelevant features.

Another work concurrent with ours that approximates second order features relationships was done independently by Bernal et al. (2012). They use second order terms in a Conditional Random Field (CRF) model (with a somewhat different interpolation scheme) and show excellent results for a bioinformatics task.

Finally, the interpolation scheme described here was used for a non-linear, non-Mahalanobis metric learning method in Pele (2011, chap. 5). However, for classification the approach described in this paper outperformed metric learning.

## 5. Results

We present results on datasets of various sizes and types, comparing our proposed pairwise piecewise-linear (PL2) embedding to state-of-the-art kernels and embeddings. The properties of all the datasets used in this paper are summarized in Table 1.

In the experiments the following algorithms are compared: our approach (PL2), our approach with only single feature embedding (PL1), Vedaldi and Zisserman's $\gamma$-homogeneous $\chi^2$ embedding (Vedaldi & Zisserman, 2012) (CHI-H), second order kernel with explicit mapping (POLY-2ND), a $d$ degree polynomial which uses the following non-linear single variable features: $[\vec{x}[1]^1, \ldots, \vec{x}[1]^d, \ldots, \vec{x}[N]^1, \ldots, \vec{x}[N]^d]$ (POLY-BTB), a polynomial kernel classifier (POLY), and an RBF kernel classifier (RBF).

### 5.1. Large Datasets

We evaluated our approach on three large datasets downloaded from UCI (A. Asuncion, 2007). For Covtype and Miniboo, experimental results are averaged over 8 runs of randomly generated stratified 70/30 splits of the data. The Protein data set has a predefined training/testing split.

Training was performed as described in Section 3.2. We implemented our own version of the Pegasos stochastic sub-gradient descent algorithm (Shalev-Shwartz et al., 2011). The code is available at: `http://www.seas.upenn.edu/~ofirpele/PL2/`. It is worth noting that using multi-class SVM and embeddings on the fly is very efficient as embedding times are amortized on the number of classes, and memory consumption is very low. For a further discussion on whether to use embedding offline or online we refer the reader to Chang et al. (2010).

For all methods, features were scaled between zero and one at training and the same scaling factors were used for the test set. Parameters of all methods were set using 30 percent of the training data as a validation set and then retraining on the whole training data with the best parameters. The regularization parameter of all SVMs, $C$, was cross-validated over the set $\{2^{-5}, 2^{-3}, \ldots, 2^{15}\}$. For PL1 (similar to Maji et al. (2012)) and our PL2 embedding methods, the maximum number of discrete points for each feature, $D$ was cross validated over the set $\{2, 4, \ldots, 20\}$. For CHI-H, the homogeneous degree $\gamma$ was cross validated over the set $\{0.1, 0.2, \ldots, 1\}$. We also compared to second degree polynomial as it can be linearly embedded (Chang et al., 2010). Finally, for POLY-BTB, $d$ was cross-validated over $\{1, 2, \ldots, 10\}$.

The results are presented in Fig. 3 (a). Our PL2 method outperformed all other methods on the Covtype and Miniboo datasets. It also outperformed all other methods on Protein, but with a small difference. PL2 consistently outperformed both the one-dimensional PL1 (similar to Maji et al. (2012)) which shows the importance of second order modeling. Finally, PL2 also consistently outperformed the second order polynomial (Chang et al., 2010) which shows that multiplicative modeling of second order features relationships is not a sufficiently accurate model for real world data. A visualization of learned parameters for the Covtype dataset is given in Fig. 4.

| | Covtype | Miniboo | Protein | Corel | Iris | Wine | Ion | Libras | Sonar |
|---|---|---|---|---|---|---|---|---|---|
| #dims | 12 | 50 | 357 | 384 | 4 | 13 | 34 | 90 | 60 |
| #examples | 581012 | 130064 | 24387 | 773 | 150 | 178 | 351 | 360 | 208 |
| #classes | 7 | 2 | 2 | 10 | 3 | 3 | 2 | 15 | 2 |

| | Cardio3 | Cardio10 | Transf | Breast | Steel | Glass | Spectf | Landsat | Segment |
|---|---|---|---|---|---|---|---|---|---|
| #dims | 21 | 21 | 4 | 9 | 27 | 9 | 44 | 36 | 19 |
| #examples | 2126 | 2126 | 748 | 106 | 1941 | 214 | 267 | 6435 | 2310 |
| #classes | 3 | 10 | 2 | 6 | 7 | 6 | 2 | 6 | 7 |

*Table 1.* Properties of the datasets.

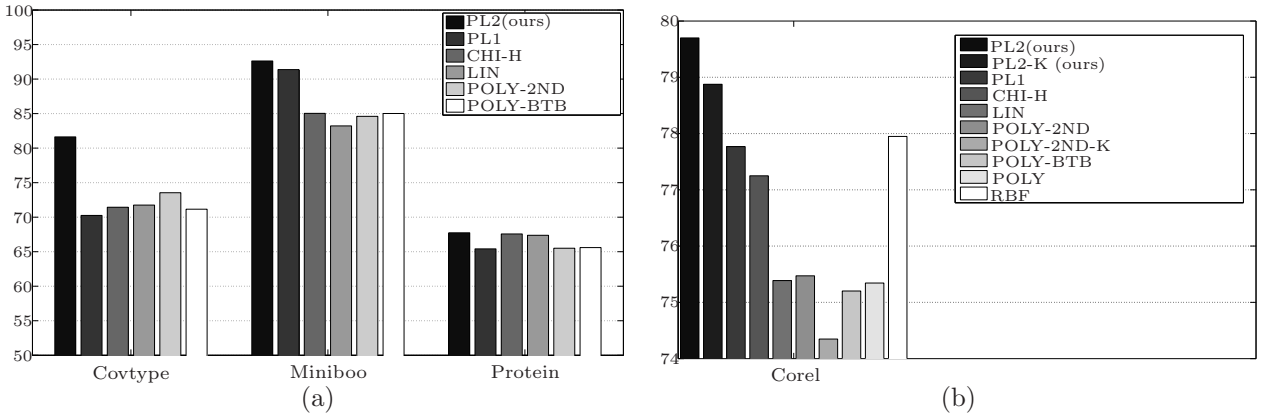


*Figure 3.* Accuracy of SVM classification using several embeddings and kernels on large datasets in (a) and the Corel image retrieval dataset in (b). Our proposed PL2 embedding outperformed all other methods on all of the datasets. We also conducted one sided tailed t-test with p-value of 0.01 of the accuracy difference vector between the PL2 method and all other methods for each of the datasets, except Protein that has a predefined training/testing split. On all datasets our PL2 is statistically significantly different from all other methods except our PL2-K (i.e, PL2 with only feature pairs that are neighbors) in the Corel dataset. See Table 2 for a description of the statistical test.

### 5.2. Image Retrieval

We employed a database that contained 773 landscape images from the COREL database that were also used in Wang et al. (2001). The dataset has 10 classes: People in Africa, Beaches, Outdoor, Buildings, Buses, Dinosaurs, Elephants, Flowers, Horses, Mountains and Food. The number of images in each class ranges from 50 to 100. As features we used SIFT-like descriptors computed on a color edge map (Ruzon & Tomasi, 2001). The descriptors were downloaded from: `www.cs.huji.ac.il/~ofirpele/FastEMD/corel_data_and_utils.zip`. See Pele & Werman (2009) for more details.

The experimental setup was similar to Section 5.1, with the following differences. First, we repeated each experiment 40 times with random shuffles, to evaluate statistical significance. Second, we evaluated the RBF and POLY baselines. For POLY, the polynomial degree was 5 fold cross-validated over the set $\{1,\ldots,10\}$. For RBF, the gamma parameter was 5 fold cross-validated over the set $\{2^{-15},2^{-13},\ldots,2^{3}\}$. The regularization parameter was cross validation as in Section 5.1. Finally, we used LIBSVM (Chang & Lin, 2011) to train the SVMs.

As mentioned earlier, we may consider a variant of $PL2$ where only $K$ feature pairs are used (out of all possible $\binom{N}{2}$). Here we tried this variant, where we only consider features that are in neighboring SIFT bins (see Pele & Werman (2009; 2010)). The same can be done for the POLY-2ND method. We denote the resulting methods by PL2-K and POLY-2ND-K.

The results are presented in Fig. 3 (b). Our PL2 and PL2-K methods statistically significantly outperformed all other methods. PL2-K is much more efficient than PL2 (on average, PL2 embedding had 73920 non-zero entries, while PL2-K had only 1989).

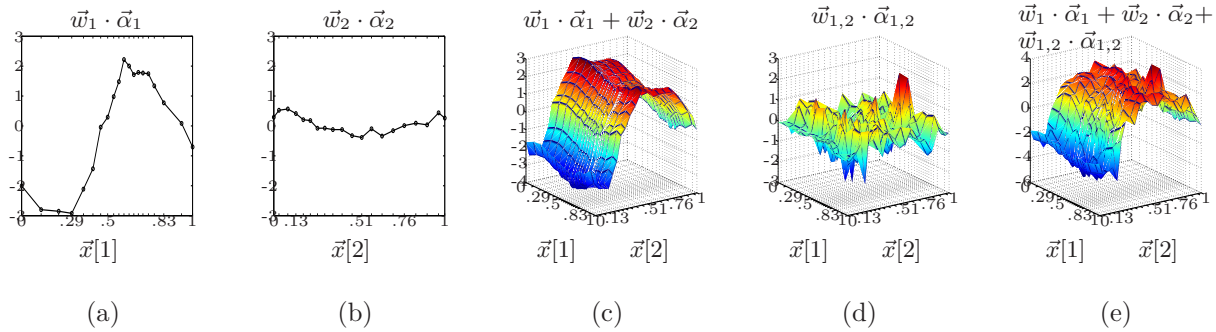


*Figure 4.* Visualization of learned parameters for the Covtype dataset for the first class. $\vec{w}_1$, $\vec{w}_2$ are the learned coefficients of single features 1, 2 and $\vec{w}_{1,2}$ are the learned coefficients of features 1 and 2 pairwise combination. (a),(b) show the functions $f_1, f_2$ learned for the first and second features. In (c) we see the two-dimensional function $f_1 + f_2$ resulting from adding the two one-dimensional functions. In (d) we see the learned function $f_{1,2}$. In (e) we see the sum of all terms containing $\vec{x}[1]$ and $\vec{x}[2]$. We can see that the two-dimensional part in (d) enables learning highly non-linear functions.

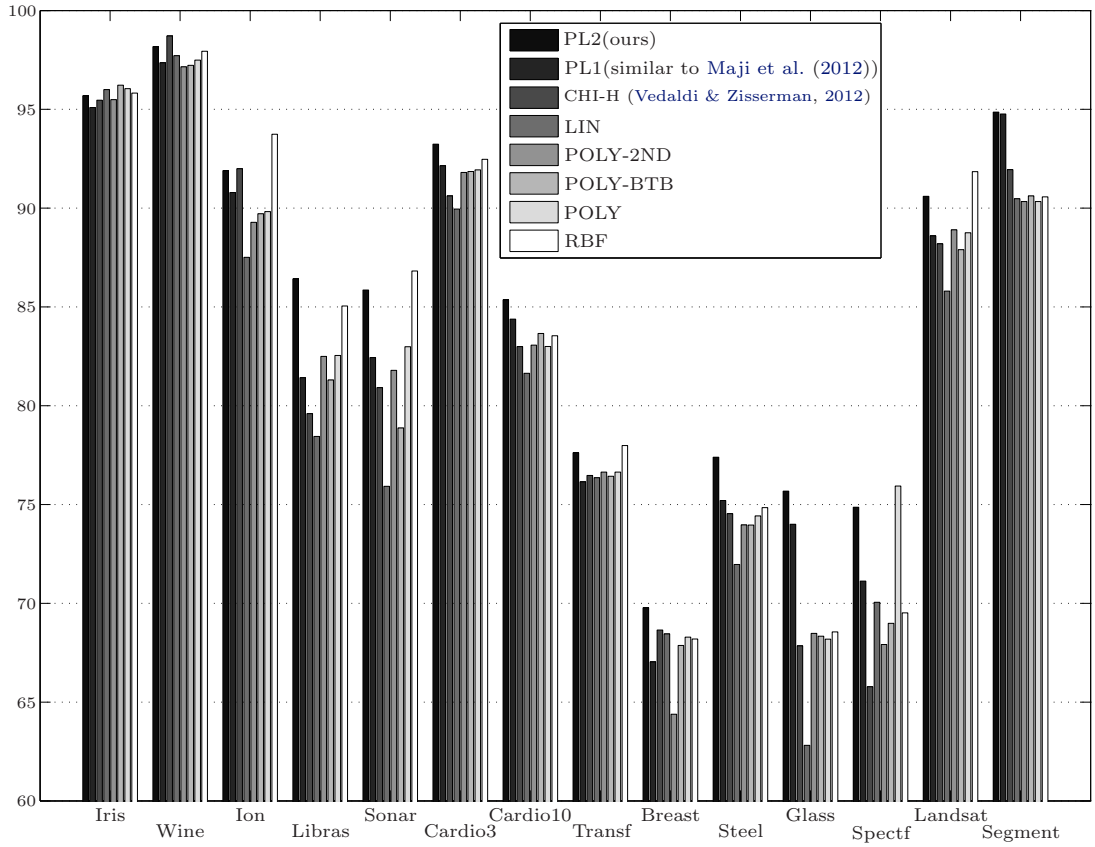


*Figure 5.* Accuracy of SVM classification using several embeddings and kernels. Our PL2 method is a strong competitor on all datasets and outperformed other methods on most of the datasets. See Table 2 for statistical significance analysis.

### 5.3. Small to Medium Datasets

We also evaluated our approach on fourteen small to medium data sets downloaded from UCI (A. Asuncion, 2007). Except for Spectf, Landsat and Segment all experimental results are averaged over 100 runs of randomly generated stratified 70/30 splits of the data. Spectf, Landset and Segment data sets have a predefined training/testing split. Other experimental details are exactly like in Section 5.2.

The results are presented in Fig. 5. There are sev-

| | Iris | Wine | Ion | Libras | Sonar | Cardio3 | Cardio10 | Transf | Breast | Steel | Glass |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Most accurate | POLY-BTB | CHI-H | RBF | **PL2** | RBF | **PL2** | **PL2** | RBF | **PL2** | **PL2** | **PL2** |
| Statistically indistinguishable | POLY,LIN RBF,**PL2** | | | | **PL2** | | | **PL2** | CHI-H,LIN POLY,RBF | | |

*Table 2.* Statistical analysis for the results in Figure 5. For each dataset, we used a one sided tailed t-test with p-value of 0.01, to test if the best method is significantly better than the others. Spectf, Landsat and Segment datasets are not shown, since they used a fixed test set.

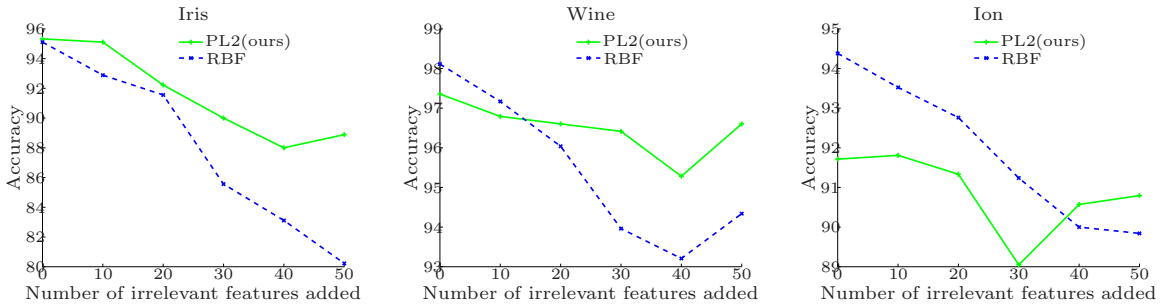


*Figure 6.* Accuracy vs. number of artificial irrelevant features added

eral observations. First, for almost all the databases our method outperformed all other methods or is at least comparable to the best method. Second, our embedding consistently outperformed both the one-dimensional PL1 (similar to Maji et al. (2012)) which shows the importance of second order relationships. Third, our method also consistently outperformed the second order polynomial (Chang et al., 2010) which shows that multiplicative modeling of second order features relationships is not a good model for real world data. Our method also outperformed general polynomial kernels on all but the Spectf dataset in which our performance is comparable. This again shows that multiplicative modeling of features relationships, even using higher order terms, is not a good model for real world data.

Finally, PL2 usually outperformed RBF, sometimes considerably (Libras, Cardio3, Cardio10, Steel, Glass, Spectf and Segment). PL2 is also much faster to train and test on these datasets. Test time in RBF scales with the number of support vectors (which often grows linearly with the dataset) multiplied by the dimension. In contrast, the time complexity of PL2 is quadratic in the dimension of the data, and independent of the training data size. In the datasets used here our method is much more efficient.

### 5.4. Irrelevant Dimensions

One possible reason that our method often outperforms RBF is that it is more robust to irrelevant features, while RBF is quite sensitive. Linear or additive models (such PL1) are also robust to irrelevant features, but they are not rich enough to capture feature dependencies. Similarly, feature selection methods can often detect irrelevant feature but are likely to miss non-linear cross feature interactions. In this section we did experiments where we gradually added irrelevant features to several datasets (features drawn uniformly from $[0, 1]$) and compared our PL2 method to RBF. Results are presented in Fig. 6.

## 6. Conclusions and Future Work

As data sets are becoming larger, non-parametric models that can be efficiently learned will probably outperform parametric methods. Here we proposed a non-parametric model that only imposes a factorization-like assumption (with respect to features) on the prediction function, and is very efficient. Extensive experiments show that the method outperforms state-of-the-art explicit feature embeddings and various kernels. This is in contrast to previous approximate embedding methods that had smaller accuracy compared to kernels methods.

Interesting future work includes applying the PL2 embedding to sparse datasets, modeling more complex features relationships than pairwise and using the embedding as a feature for segmentation.

**Acknowledgments:** This research is partially supported by the ISF Centers of Excellence grant 1789/11.

## References


A. Asuncion, D.J. Newman. UCI machine learning repository, 2007.

Bernal, A., Crammer, K., and Pereira, F. Automated gene-model curation using global discriminative learning. *Bioinformatics*, 2012.

Chang, Chih-Chung and Lin, Chih-Jen. LIBSVM: A library for support vector machines. *IST*, 2011.

Chang, Y.W., Hsieh, C.J., Chang, K.W., Ringgaard, M., and Lin, C.J. Training and testing low-degree polynomial data mappings via linear SVM. *JMLR*, 2010.

Crammer, K. and Singer, Y. On the algorithmic implementation of multiclass kernel-based vector machines. *JMLR*, 2002.

Joachims, T. Training linear SVMs in linear time. In *KDD*, 2006.

Maji, S., Berg, A.C., and J., Malik. Efficient classification for additive kernel SVMs. *PAMI*, 2012.

Pele, O. *Distance Functions: Theory, Algorithms and Applications*. PhD thesis, The Hebrew University of Jerusalem, 2011.

Pele, O. and Werman, M. Fast and robust earth mover's distances. In *ICCV*, 2009.

Pele, O. and Werman, M. The quadratic-chi histogram distance family. In *ECCV*, 2010.

Perronnin, F., Senchez, J., et al. Large-scale image categorization with explicit data embedding. In *CVPR*, 2010.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. *NIPS*, 2007.

Ruzon, M.A. and Tomasi, C. Edge, Junction, and Corner Detection Using Color Distributions. *PAMI*, 2001.

Shalev-Shwartz, S., Singer, Y., Srebro, N., and Cotter, A. Pegasos: primal estimated sub-gradient solver for SVM. *MP*, 2011.

Tsang, I.W., Kwok, J.T., and Cheung, P.M. Very large svm training using core vector machines. In *Proc. AISTATS*, pp. 349–356, 2005.

Vedaldi, A. and Zisserman, A. Efficient additive kernels via explicit feature maps. *PAMI*, 2012.

Wang, J.Z., Li, J., and Wiederhold, G. SIMPLIcity: Semantics-Sensitive Integrated Matching for Picture LIbraries. *PAMI*, 2001.