On the Learnability and Design of Output Codes for Multiclass Problems

Koby Crammer and **Yoram Singer**
School of Computer Science & Engineering
The Hebrew University, Jerusalem 91904, Israel
{kobics,singer}@cs.huji.ac.il

Abstract

Output coding is a general framework for solving multiclass categorization problems. Previous research on output codes has focused on building multiclass machines given *predefined* output codes. In this paper we discuss for the first time the problem of *designing* output codes for multiclass problems. For the design problem of discrete codes, which have been used extensively in previous works, we present mostly negative results. We then introduce the notion of continuous codes and cast the design problem of continuous codes as a constrained optimization problem. We describe three optimization problems corresponding to three different norms of the code matrix. Interestingly, for the l_2 norm our formalism results in a quadratic program whose dual does *not* depend on the length of the code. A special case of our formalism provides a multiclass scheme for building support vector machines which can be solved efficiently. We give a time and space efficient algorithm for solving the quadratic program. Preliminary experiments we have performed with synthetic data show that our algorithm is often two orders of magnitude faster than standard quadratic programming packages.

1 Introduction

Many applied machine learning problems require assigning labels to instances where the labels are drawn from a finite set of labels. This problem is often referred to as multiclass categorization or classification. Examples for machine learning applications that include a multiclass categorization component include optical character recognition, text classification, phoneme classification for speech synthesis, medical analysis, and more. Some of the well known binary classification learning algorithms can be extended to handle multiclass problem (see for instance [4, 17, 18]). A general approach is to reduce a multiclass problem to a multiple binary classification problem.

Dietterich and Bakiri [8] described a general approach based on error-correcting codes which they termed error-correcting output coding (ECOC), or in short output coding. Output coding for multiclass problems is composed of two stages. In the training stage we need to construct multiple (supposedly) independent binary classifiers each of which is based on a different partition of the set of the labels into two disjoint sets. In the second stage, the classification part, the predictions of the binary classifiers are combined to extend a prediction on the original label of a test instance. Experimental work has shown that output coding can often greatly improve over standard reductions to binary problems [8, 9, 14, 1, 19, 7, 3, 2]. The performance of output coding was also analyzed in statistics and learning theoretic contexts [11, 13, 20, 2].

Most of the previous work on output coding has concentrated on the problem of solving multiclass problems using *predefined* output codes, independently of the specific application and the class of hypotheses used to construct the binary classifiers. Therefore, by predefining the output code we ignore the complexity of the induced binary problems. The output codes used in experiments were typically confined to a specific family of codes. Several family of codes have been suggested and tested so far, such as, comparing each class against the rest, comparing all pairs of classes [11, 2], random codes [8, 19, 2], exhaustive codes [8, 2], and linear error correcting codes [8]. A few heuristics attempting to modify the code so as to improve the multiclass prediction accuracy were suggested (e.g., [1]). However, they did not yield significant improvements and, furthermore, they lack any formal justification.

In this paper we concentrate on the problem of designing a good code for a given multiclass problem. In Sec. 3 we study the problem of finding the first column of a discrete code matrix. Given a binary classifier, we show that finding a good first column can be done in polynomial time. In contrast, when we restrict the hypotheses class from which we choose the binary classifiers, the problem of finding a good first column becomes difficult. This result underscores the difficulty of the code design problem. Furthermore, in Sec. 4 we discuss the general design problem and show that given a set of binary classifiers the problem of finding a good code matrix is NP-complete.

Motivated by the intractability results we introduce in Sec. 5 the notion of continuous codes and cast the design problem of continuous codes as a constrained optimization problem. As in discrete codes, each column of the code matrix divides the set of labels into two subsets which are labeled positive ($+$) and negative ($-$). The sign of each entry in the code matrix determines the subset association ($+$ or $-$) and the magnitude corresponds to the confidence in this association. Given this formalism, we seek an output code with small empirical loss whose matrix norm is small. We describe three optimization problems corresponding to three different norms of the code matrix: l_1, l_2 and l_∞. For l_1 and l_∞ we show that the code design problem can be solved by linear programming (LP). Interestingly, for the l_2 norm our formalism results in a quadratic program (QP) whose dual does *not* depend on the length of the code. Similar to support vector machines, the dual program can be expressed in terms of inner-products between input instances, hence we can employ kernel-based binary classifiers. Our framework yields, as a special case, a direct and efficient method for constructing multiclass support vector machine.

The number of variables in the dual of the quadratic problem is the product of the number of samples by the number of classes. This value becomes very large even for small datasets. For instance, an English letter recognition problem with $1{,}000$ training examples would require $26{,}000$ variables. In this case, the standard matrix representation of dual quadratic problem would require more than 5 Giga bytes of memory. We therefore describe in Sec. 6.1 a memory efficient algorithm for solving the quadratic program for code design. Our algorithm is reminiscent of Platt's sequential minimal optimization (SMO) [15]. However, unlike SMO, our algorithm optimize on each round a reduced subset of the variables that corresponds to a single example. Informally, our algorithm reduces the optimization problem to a sequence of small problems, where the size of each reduced problem is equal to the number of classes of the original multiclass problem. Each reduced problem can again be solved using a standard QP technique. However, standard approaches would still require large amount of memory when the number of classes is large and a straightforward solution is also time consuming. We therefore further develop the algorithm and provide an analytic solution for the reduced problems and an efficient algorithm for calculating the solution. The run time of the algorithm is poly-logarithmic and the memory requirements are linear in the number of classes. We conclude with simulations results showing that our algorithm is at least two orders of magnitude faster than a standard QP technique, even for small number of classes.

2 Discrete codes

Let $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ be a set of m training examples where each instance x_i belongs to a domain $\mathcal{X}$. We assume without loss of generality that each label y_i is an integer from the set $\mathcal{Y} = \{1, \ldots, k\}$. A multiclass classifier is a function $H : \mathcal{X} \rightarrow \mathcal{Y}$ that maps an instance x into an element y of $\mathcal{Y}$. In this work we focus on a framework that uses *output codes* to build multiclass classifiers from binary classifiers. A discrete output code M is a matrix of size $k \times l$ over $\{-1, +1\}$ where each row of M correspond to a class $y \in \mathcal{Y}$. Each column of M defines a partition of $\mathcal{Y}$ into two disjoint sets. Binary learning algorithms are used to construct classifiers, one for each column t of M. That is, the set of examples induced by column t of M is $(x_1, M_{t,y_1}), \ldots, (x_m, M_{t,y_m})$. This set is fed as training data to a learning algorithm that finds a hypothesis $h_t : \mathcal{X} \rightarrow \{-1, +1\}$. This reduction yields l different binary classifiers $h_1, \ldots, h_l$. We denote the vector of predictions of these classifiers on an instance x as $\bar{h}(x) = (h_1(x), \ldots, h_l(x))$. We denote the rth row of M by $\bar{M}_r$.

Given an example x we predict the label y for which the row $\bar{M}_y$ is the "closest" to $\bar{h}(x)$. We will use a general notion for closeness and define it through an inner-product function $K : \mathbb{R}^l \times \mathbb{R}^l \rightarrow \mathbb{R}$. The higher the value of $K(\bar{h}(x), \bar{M}_r)$ is the more confident we are that r is the correct label of x according to the classifiers $\bar{h}$. An example for a closeness function is $K(\bar{u}, \bar{v}) = \bar{u} \cdot \bar{v}$. It is easy to verify that this choice of K is equivalent to picking the row of M which attains the minimal Hamming distance to $\bar{h}(x)$.

Given a classifier $H(x)$ and an example (x, y), we say that $H(x)$ misclassified the example if $H(x) \neq y$. Let $[\![\pi]\!]$ be 1 if the predicate π holds and 0 otherwise. Our goal is therefore to find a classifier $H(x)$ such that $\frac{1}{m}\sum_{i=1}^{m} [\![H(x_i) \neq y_i]\!]$ is small.

When l is small there might be more then one row of M which attains the maximal value according to the function K. To accommodate such cases we will relax our definition and define a classifier $H(X)$ based on a code M to be the mapping $H(x) : \mathcal{X} \rightarrow 2^{\mathcal{Y}}$ given by $H(x) = \{y \mid K(\bar{h}(x), \bar{M}_y) = \max_r K(\bar{h}(x), \bar{M}_r)\}$. In this case we will pick one of the labels in $H(x)$ uniformly at random, and use the expected error of $H(x)$,

$$\begin{aligned} \epsilon_S(M, \bar{h}) &\stackrel{\text{def}}{=} \frac{1}{m}\sum_{i=1}^{m}\left(1 - \frac{[\![y_i \in H(x_i)]\!]}{|H(x_i)|}\right) \\ &= 1 - \frac{1}{m}\sum_{i=1}^{m}\frac{[\![y_i \in H(x_i)]\!]}{|H(x_i)|} \qquad (1) \end{aligned}$$

In the context of output codes, a multiclass mapping $H(x)$ is thus determined by two parameters: the coding matrix M and the set of binary classifiers $\bar{h}(x)$. Assume that the binary classifiers $h_1(x) \ldots h_l(x)$ are chosen from some hypothesis class $\mathcal{H}$. The following natural learning problems arise: **(a)** Given a matrix M, find a set $\bar{h}$ which suffers small empirical loss. **(b)** Given a set of binary classifiers $\bar{h}$, find a matrix M which has small empirical loss. **(c)** Find both a matrix M and a set $\bar{h}$ which have small empirical loss.

Previous work has focused mostly on the first problem. In this paper we mainly concentrate on the code design problem (problem **b**), that is, finding a good matrix M.

3 Finding the first column of an output code

Assume we are given a single binary classifier $h_1(x)$ and we want to find the first column of the matrix M which minimizes the empirical loss $\epsilon_S(M, \bar{h})$. For brevity, let us denote by $\bar{u} = (u_1 \ldots u_k)^T$ the first column of M. We now describe an efficient algorithm that finds $\bar{u}$ given $h_1(x)$. The algorithm's running time is polynomial in the size of the label set $k = |\mathcal{Y}|$ and the sample size m. First, note that in this case

$$y_i \in H(x_i) \Leftrightarrow h_1(x_i) = u_{y_i} \ . \qquad (2)$$

Second, note that the sample can be divided into $2k$ equivalence classes according to their labels and the classification of $h_1(x)$. For $r = 1, \ldots, k$ and $b \in \{-1, +1\}$, define $a_r^b = \frac{1}{m}|\{i : y_i = r, h_1(x_i) = b\}|$ to be the fraction of the examples with label r and classification b (according to $h_1(x)$). For $b \in \{-1,+1\}$, denote by $a^b = \sum_{r=1}^k a_r^b$, and let $w^b = |\{r : u_r = b\}|$ be the number of elements in $\bar{u}$ which are equal to b. (For brevity, we will often use $+$ and $-$ to denote the value of b.) Let

$$\xi_S(M,\bar{h}) \stackrel{\text{def}}{=} \frac{1}{m}\sum_{i=1}^m \frac{[\![y_i \in H(x_i)]\!]}{|H(x_i)|} = 1 - \epsilon_S(M,\bar{h}) \ . \quad (3)$$

We can assume without loss of generality that not all the elements in $\bar{u}$ are the same (otherwise, $\xi_S(M,\bar{h}) = \frac{1}{k}$, which is equivalent to random guessing). Hence, the size of $H(x)$ is :

$$|H(x)| = \begin{cases} w^+ & h(x) = +1 \\ w^- & h(x) = -1 \ . \end{cases} \quad (4)$$

Using Eqs. (2) and (4), we rewrite Eq. (3),

$$\begin{aligned} \xi_S(M,\bar{h}) &= \frac{1}{m}\sum_{i:y_i \in H(x_i)} \frac{1}{|H(x_i)|} \\ &= \frac{1}{m}\sum_{i:h(x_i)=u_{y_i}} \frac{1}{|H(x_i)|} \\ &= \frac{1}{m}\sum_{r=1}^k \sum_{\substack{i:y_i=r \\ h(x_i)=u_r}} \begin{cases} \frac{1}{w^+} & u_r = +1 \\ \frac{1}{w^-} & u_r = -1 \end{cases} \\ &= \sum_{r=1}^k \begin{cases} \frac{a_r^+}{w^+} & u_r = +1 \\ \frac{a_r^-}{w^-} & u_r = -1 \ . \end{cases} \end{aligned} \quad (5)$$

Using Eq. (5) we now can expand $\xi_S(M,\bar{h})$,

$$\begin{aligned} \xi_S(M,\bar{h}) &= \sum_{r=1}^k \left[\frac{1}{2}(1+u_r)\frac{a_r^+}{w^+} + \frac{1}{2}(1-u_r)\frac{a_r^-}{w^-}\right] \\ &= \frac{1}{2}\sum_{r=1}^k \left[u_r\left(\frac{a_r^+}{w^+} - \frac{a_r^-}{w^-}\right)\right] + \frac{1}{2}\sum_{r=1}^k \left(\frac{a_r^+}{w^+} + \frac{a_r^-}{w^-}\right) \\ &= \frac{1}{2}\sum_{r=1}^k \left[u_r\left(\frac{a_r^+}{w^+} - \frac{a_r^-}{w^-}\right)\right] + \frac{1}{2}\left(\frac{a^+}{w^+} + \frac{a^-}{w^-}\right) \ . \end{aligned} \quad (6)$$

For a particular choice of w^+ (and $w^- = k - w^+$) ξ_S is maximized (and ϵ_S is minimized) by setting $u_r = +1$ at the w^+ indices which attain the highest values for $\left(\frac{a_r^+}{w^+} - \frac{a_r^-}{w^-}\right)$, and set the rest w^- of the indices to -1. This can be done efficiently in $k \log k$ time using sorting. Therefore, the best choice of $\bar{u}$ is found by enumerating all the possible values for $w^+ \in \{1,\ldots,k-1\}$ and choosing the value of w^+, w^- which achieves the maximal value for Eq. (6). Since it takes m operations to calculate a_r^+ and a_r^-, the total number of operations needed to find the optimal choice for the first column is $O(m + k^2 \log k)$. We have proven the following theorem.

Theorem 1 *Let $S = \{(x_1,y_1),\ldots,(x_m,y_m)\}$ be a set of m training examples, where each label is an integer from the set $\{1,\ldots,k\}$. Let $\mathcal{H}$ be a binary hypothesis class. Given an hypothesis $h_1(x) \in \mathcal{H}$, the first column of an output code which minimizes the empirical loss defined by Eq. (1) can be found in polynomial time.*

To conclude this section we use a reduction from SAT to demonstrate that if the learning algorithm (and its corresponding class of hypotheses from which h_1 can chosen from) is of a very restricted form then the resulting learning problem can be hard. Let $\Psi(x_1 \ldots x_n)$ be a boolean formula over the variables $x_i \in \{-1,+1\}$ where we interpret $x = -1$ as False and $x = +1$ as True. Define $\mathcal{X} = \{x_i, \bar{x}_i\}_{i=1}^n \cup \{\perp\}$ to be the instance space. Let $S = \{(x_i, i)\}_{i=1}^n \cup \{(\bar{x}_i, i+n)\}_{i=1}^n \cup \{(\perp, 2n+1)\}$ be a sample of size $m = 2n+1$, where the labels are taken from $\mathcal{Y} = \{1,\ldots,2n+1\}$. Define the learning algorithm L_Ψ as follows. The algorithm's input is a binary labeled sample of the form $\{(x_i,y_i),(\bar{x}_i,y_{i+n}),(\perp,y_{2n+1})\}_{i=1}^n$. If $\Psi(x_1,...,x_n) = \texttt{True}$ and for all i $y_i \oplus y_{i+n} = \texttt{True}$, then the algorithm returns an hypothesis which is consistent with the sample (the sample itself). Otherwise, the algorithm returns the constant hypothesis, $h(x) \equiv 1$ or $h(x) \equiv 0$, which agrees with the majority of the sample by choosing $h(x) \equiv \arg\max_{b \in \{-1,+1\}} |\{i : y_i = b\}|$. Note that the learning algorithm is non-trivial in the sense that the hypothesis it returns has an empirical loss of less than $1/2$ on the binary labeled sample.

We now show that a multiclass learning algorithm that minimizes the empirical loss ϵ_S over both the first column $\bar{u}$ and the hypothesis $h_1(x)$ which was returned by the algorithm L_Ψ, can be used to check whether the formula Ψ is satisfiable. We need to consider two cases. When $\Psi(x_1,...,x_n) = \texttt{True}$ and for all i $y_i \oplus y_{i+n} = \texttt{True}$, then using the definition from Eq. (3) we get $\xi_S = \frac{1}{m}\left(n\frac{1}{n} + (n+1)\frac{1}{n+1}\right) = \frac{2}{m}$. If the above conditions do not hold ($h(x)$ is constant), let $v \geq n+1$ be the number of examples which the hypothesis $h_1(x)$ classifies correctly. Then, using Eq. (3) again we obtain $\xi_S = \frac{1}{m}\left(v\frac{1}{v}\right) = \frac{1}{m}$. Thus, the minimum of ϵ_S is achieved if and only if the formula Ψ is satisfiable. Therefore, a learning algorithm for $h_1(x)$ and $\bar{u}$ can also be used as an oracle for the satisfiability of Ψ.

While the setting discussed in this section is somewhat superficial, these results underscore the difficulty of the problem. We next show that the problem of finding a good output code given a relatively large set of classifiers $\bar{h}(x)$ is intractable. We would like to note in passing that efficient algorithm for finding a single column might be useful in other settings. For instance in building trees or directed acyclic graphs for multiclass problems (cf. [16]). We leave this for future research.

4 Finding a general discrete output code

In this section we prove that given a set of l binary classifiers $\bar{h}(x)$, finding a code matrix which minimizes the empirical loss $\epsilon_S(M,\bar{h})$ is NP-complete. Given a sample $S = \{(x_1,y_1),\ldots,(x_m,y_m)\}$ and a set of classifiers $\bar{h}$, let us denote by $\tilde{S} = \{(\bar{h}_1,y_1),\ldots,(\bar{h}_m,y_m)\}$ the evaluation of $\bar{h}(\cdot)$ on the sample S, where $\bar{h}_i \stackrel{\text{def}}{=} \bar{h}(x_i)$. We now show that even when $k=2$ and $K(\bar{u},\bar{v}) = \bar{u}\cdot\bar{v}$ the problem is NP-complete. (Clearly, the problem remains NPC for $k > 2$). Following the notation of previous sections, the output code matrix is composed of two rows $\bar{M}_1$ and $\bar{M}_2$ and the predicted class for instance x_i is $H(x_i) = \arg\max_{r=1,2}\{\bar{M}_r \cdot \bar{h}_i\}$. For the simplicity of the presentation of the proof, we assume that both the code M and the hypotheses' values $\bar{h}_i$ are over the set $\{0,1\}$ (instead of $\{-1,+1\}$). This assumption does not change the problem since there is a linear transform between the two sets.

Theorem 2 *The following decision problem is NP-complete.*
`Input:` *A natural number q, a labeled sample*
$$\tilde{S} = \{(\bar{h}_1, y_1), \ldots, (\bar{h}_m, y_m)\},$$
where $y_i \in \{1,2\}$, and $\bar{h}_i \in \{0,1\}^l$.
`Question:` *Does there exist a matrix $M \in \{0,1\}^{2\times l}$ such that the classifier $H(x)$ based on an output code M makes at most q mistakes on $\tilde{S}$.*

Proof: Our proof is based on a reduction technique introduced by Höffgen and Simon [12]. First, note that the problem is in NP as we can pick a random code matrix M and check the number of classification errors it induces in polynomial time.

We show a reduction to Vertex Cover in order to prove that the problem is NP-hard. Given an undirected graph $G = (V, E)$, we will code the structure of the graph as follows. The sample $\tilde{S}$ will be composed of two subsets, $\tilde{S}^E$ and $\tilde{S}^V$ of size $2|E|$ and $|V|$ respectively. We set $l = 2|V| + 1$. Each edge $\{v_i, v_j\} \in E$ is encoded by two examples $(\bar{h}, y)$ in $\tilde{S}^E$. We set for the first vector to $h_i = 1$, $h_j = 1$, $h_l = 1$ and 0 elsewhere. We set the second vector to $h_{i+n} = 1$, $h_{j+n} = 1$, $h_l = 1$ and 0 elsewhere. We set the label y of each example in $\tilde{S}^E$ to 1. Each example $(\bar{h}, y)$ in $\tilde{S}^V$ encodes a node $v_i \in V$ where $h_i = 1$, $h_{i+n} = 1$, $h_l = 1$ and 0 elsewhere. We set the label y of each example in $\tilde{S}^V$ to 2 (second class). We now show that there exists a vertex cover $U \subseteq V$ with at most q nodes if and only if there exists a coding matrix $M \in \{0,1\}^{2\times l}$ that induces at most q classification errors on the sample $\tilde{S}$.

$(\Rightarrow)$: Let $U \subseteq V$ be a vertex cover such that $|U| \leq q$. We show that there exists a code which has at most q mistakes on $\tilde{S}$. Let $\mathbf{u} \in \{0,1\}^{|V|}$ be the characteristic function of U, that is, $u_i = 1$ if $v_i \in U$ and $u_i = 0$ otherwise. Define the output code matrix to be $\bar{M}_1 = (\mathbf{u}, \mathbf{u}, 1)$ and $\bar{M}_2 = (\neg\mathbf{u}, \neg\mathbf{u}, 0)$. Here, $\neg u$ denotes the component-wise logical not operator.

Since U is a cover, for each $\bar{h} \in \tilde{S}^E$ we get
$$\bar{M}_1 \cdot \bar{h} \geq 2 \;\text{ and }\; \bar{M}_2 \cdot \bar{h} \leq 1 \;\;\Rightarrow\;\; \bar{M}_2 \cdot \bar{h} < \bar{M}_1 \cdot \bar{h} \;.$$
Therefore, for all the examples in $\tilde{S}^E$ the predicted label equals the true label and we suffer 0 errors on these examples. For each example $\bar{h} \in \tilde{S}^V$ that corresponds to a node $v \in U$ we have
$$\bar{M}_1 \cdot \bar{h} = 3 > 0 = \bar{M}_2 \cdot \bar{h} \;.$$
Therefore, these examples are misclassified (Recall that the label of each example in $\tilde{S}^V$ is 2). Analogously, for each example in $\tilde{S}^V$ which corresponds to $v \notin U$ we get
$$\bar{M}_1 \cdot \bar{h} = 1 < 2 = \bar{M}_2 \cdot \bar{h} \;,$$
and these examples are correctly classified. We thus have shown that the total number of mistakes according to M is $|U| \leq q$.

$(\Leftarrow)$: Let M be a code which achieves at most q mistakes on $\tilde{S}$. We construct a subset $U \subseteq V$ as follows. We scan $\tilde{S}$ and add to U all vertices v_i corresponding to misclassified examples from $\tilde{S}^V$. Similarly, for each misclassified example from $\tilde{S}^E$ corresponding to an edge $\{v_i, v_j\}$, we pick either v_i or v_j at random and add it to U. Since we have at most q misclassified examples in $\tilde{S}$ the size of U is at most q. We claim that the set U is a vertex cover of the graph G. Assume by contradiction that there is an edge $\{v_i, v_j\}$ for which neither v_i nor v_j belong to the set U. Therefore, by construction, the examples corresponding to the vertices v_i and v_j are classified correctly and we get,
$$\begin{aligned} M_{1,i} + M_{1,i+n} + M_{1,l} &< M_{2,i} + M_{2,i+n} + M_{2,l} \\ M_{1,j} + M_{1,j+n} + M_{1,l} &< M_{2,j} + M_{2,j+n} + M_{2,l} \end{aligned}$$
Summing the above equations yields that,
$$\begin{aligned} &M_{1,i} + M_{1,j} + M_{1,i+n} + M_{1,j+n} + 2M_{1,l} < \\ &\quad M_{2,i} + M_{2,j} + M_{2,i+n} + M_{2,j+n} + 2M_{2,l} \;. \end{aligned} \tag{7}$$
In addition, the two examples corresponding to the edge $\{v_i, v_j\}$ are classified correctly, implying that
$$\begin{aligned} M_{1,i} + M_{1,j} + M_{1,l} &> M_{2,i} + M_{2,j} + M_{2,l} \\ M_{1,i+n} + M_{1,j+n} + M_{1,l} &> M_{2,i+n} + M_{2,j+n} + M_{2,l} \end{aligned}$$
which again by summing the above equations yields,
$$\begin{aligned} &M_{1,i} + M_{1,j} + M_{1,i+n} + M_{1,j+n} + 2M_{2,l} > \\ &\quad M_{2,i} + M_{2,j} + M_{2,i+n} + M_{2,j+n} + 2M_{2,l} \;. \end{aligned} \tag{8}$$
Comparing Eqs. (7) and (8) we get a contradiction. ■

5 Continuous codes

The intractability results of previous sections motivate a relaxation of output codes. In this section we describe a natural relaxation where both the classifiers' output and the code matrix are over the reals.

As before, the classifier $H(x)$ is constructed from a code matrix M and a set of binary classifiers $\bar{h}(x)$. The matrix M is of size $k \times l$ over $\mathbb{R}$ where each row of M corresponds to a class $y \in \mathcal{Y}$. Analogously, each binary classifier $h_t(x) \in \mathcal{H}$ is a mapping $h_t(x) : \mathcal{X} \rightarrow \mathbb{R}$. A column t of M defines a partition of $\mathcal{Y}$ into two disjoint sets. The sign of each element of the tth column is interpreted as the set (+1 or -1) to which the class r belongs and the magnitude $|M_{r,t}|$ is interpreted as the confidence in the associated partition. Similarly, we interpret the sign of $h_t(x)$ as the prediction of the set (+1 or -1) to which the label of the instance x belongs and the magnitude $|h_t(x)|$ as the confidence of this prediction. Given an instance x, the classifier $H(x)$ predicts the label y which maximizes the confidence function $K(\bar{h}(x), \bar{M}_r)$, $H(x) = \arg\max_{r\in\mathcal{Y}}\{K(\bar{h}(x), \bar{M}_r)\}$. In contrast to discrete codes, we can assume here without loss of generality that *exactly one* class attains the maximum value according to the function K. We will concentrate on the problem of finding a good continuous code given a set of binary classifiers $\bar{h}$.

The approach we will take is to cast the code design problem as constrained optimization problem. Borrowing the idea of soft margin [6] we replace the discrete 0-1 multi-class loss with the linear bound

$$\max_r\{K(\bar{h}(x_i), \bar{M}_r) + 1 - \delta_{y_i,r}\} - K(\bar{h}(x_i), \bar{M}_{y_i}) \ . \quad (9)$$

This formulation is also motivated by the generalization analysis of Schapire et al. [2]. The analysis they give is based on the margin of examples where the margin is closely related to the definition of the loss as given by Eq. (9).

Put another way, the correct label should have a confidence value which is larger by at least one than any of the confidences for the rest of the labels. Otherwise, we suffer loss which is linearly proportional to the difference between the confidence of the correct label and the maximum among the confidences of the other labels. The bound on the empirical loss is then,

$$\begin{aligned}\epsilon_S(M, \bar{h}) &= \frac{1}{m}\sum_{i=1}^m [\![H(x_i) \neq y_i]\!] \\ &\leq \frac{1}{m}\sum_{i=1}^m \Big[\max_r\{K(\bar{h}(x_i), \bar{M}_r) + 1 - \delta_{y_i,r}\} \\ &\qquad - K(\bar{h}(x_i), \bar{M}_{y_i})\Big] \ ,\end{aligned}$$

where $\delta_{i,j}$ equals 1 if $i = j$ and 0 otherwise. We say that a sample S is classified correctly using a set of binary classifiers $\bar{h}$ if there exists a matrix M such that the above loss is equal to zero,

$$\forall i \ \max_r\{K(\bar{h}(x_i), \bar{M}_r) + 1 - \delta_{y_i,r}\} - K(\bar{h}(x_i), \bar{M}_{y_i}) = 0 \ . \quad (10)$$

Denote by

$$b_{i,r} = 1 - \delta_{y_i,r} \ . \quad (11)$$

Thus, a matrix M that satisfies Eq. (10) would also satisfy the following constraints,

$$\forall i, r \quad K(\bar{h}(x_i), \bar{M}_{y_i}) - K(\bar{h}(x_i), \bar{M}_r) \geq b_{i,r} \ . \quad (12)$$

Motivated by [21, 2] we seek a matrix M with a small norm which satisfies Eq. (12). Thus, when the entire sample S can be labeled correctly, the problem of finding a good matrix M can be stated as the following optimization problem,

$$\begin{aligned}&\min_M && \|\mathbf{M}\|_p \\ &\text{subject to :} && \forall i, r \ K(\bar{h}(x_i), \bar{M}_{y_i}) - K(\bar{h}(x_i), \bar{M}_r) \geq b_{i,r}\end{aligned}$$

Here p is an integer. Note that m of the constraints for $r = y_i$ are automatically satisfied. This is changed in the following derivation for the non-separable case. In the general case a matrix M which classifies all the examples correctly might not exist. We therefore introduce slack variables $\xi_i \geq 0$ and modify Eq. (10) to be,

$$\forall i \ \max_r\{K(\bar{h}(x_i), \bar{M}_r) + 1 - \delta_{y_i,r}\} - K(\bar{h}(x_i), \bar{M}_{y_i}) = \xi_i \ . \quad (13)$$

The corresponding optimization problem is,

$$\begin{aligned}&\min_M && \beta\|M\|_p + \sum_{i=1}^m \xi_i \quad (14) \\ &\text{subject to :} && \\ &\forall i, r && K(\bar{h}(x_i), \bar{M}_{y_i}) - K(\bar{h}(x_i), \bar{M}_r) \geq b_{i,r} - \xi_i\end{aligned}$$

for some constant $\beta \geq 0$. This is an optimization problem with "soft" constraints. Analogously, we can define an optimization problem with "hard" constraints,

$$\begin{aligned}&\min_M && \sum_{i=1}^m \xi_i \\ &\text{subject to :} && \\ &\forall i, r && K(\bar{h}(x_i), \bar{M}_{y_i}) - K(\bar{h}(x_i), \bar{M}_r) \geq b_{i,r} - \xi_i \\ & && \|M\|_p \leq A, \text{ for some } A > 0\end{aligned}$$

The relation between the "hard" and "soft" constraints and their formal properties is beyond the scope of this paper. For further discussion on the relation between the problems see [21].

5.1 Design of continuous codes using Linear Programming

We now further develop Eq. (14) for the cases $p = 1, 2, \infty$. We deal first with the cases $p = 1$ and $p = \infty$ which result in linear programs. For the simplicity of presentation we will assume that $K(\bar{u}, \bar{v}) = \bar{u} \cdot \bar{v}$.

For the case $p = 1$ the objective function of Eq. (14) becomes $\beta \sum_{i,r} |M_{i,r}| + \sum_i \xi_i$. We introduce a set of auxiliary variables $\alpha_{i,r} = |M_{i,r}|$ to get a standard linear programming setting,

$$\begin{aligned}&\min_{M,\xi,\alpha} && \beta\sum_{r,t}\alpha_{r,t} + \sum_{i=1}^m \xi_i \\ &\text{subject to :} && \forall i, r \quad \xi_i + \bar{h}(x_i)\cdot\bar{M}_{y_i} - \bar{h}(x_i)\cdot\bar{M}_r \geq b_{i,r} \\ & && \forall r, t \quad \alpha_{r,t} \geq \pm M_{r,t}\end{aligned}$$

To obtain its dual program (see also App. A) we define one variable for each constraint of the primal problem. We use $\eta_{i,r}$ for the first set of constraints, and $\gamma^{\pm}_{t,r}$ for the second set. The dual program is,

$$\begin{aligned}&\max_{\eta,\gamma^{\pm}} && \sum_{i,r}\eta_{i,r}b_{i,r} \\ &\text{subject to :} && \forall i, r, t \quad \eta_{i,r}, \gamma^+_{t,r}, \gamma^-_{t,r} \geq 0 \\ & && \forall i \quad \sum_r \eta_{i,r} = 1 \\ & && \forall r, t \quad \gamma^+_{t,r} + \gamma^-_{t,r} = \beta \\ & && \forall r, t \quad \sum_i h_t(x_i)[\delta_{y_i,r} - \eta_{i,r}] = -\gamma^+_{t,r} + \gamma^-_{t,r}\end{aligned}$$

The case of $p = \infty$ is similar. The objective function of Eq. (14) becomes $\beta \max_{i,r} |M_{i,r}| + \sum_i \xi_i$. We introduce a single new variable $\alpha = \max_{i,r} |M_{i,r}|$ to obtain the primal problem,

$$\begin{aligned}&\min_{M,\xi,\alpha} && \beta\alpha + \sum_{i=1}^m \xi_i \\ &\text{subject to :} && \forall i, r \quad \xi_i + \bar{h}(x_i)\cdot\bar{M}_{y_i} - \bar{h}(x_i)\cdot\bar{M}_r \geq b_{i,r} \\ & && \forall r, t \quad \alpha \geq \pm M_{r,t}\end{aligned}$$

Following the technique for $p = 1$, we get that the dual program is,

$$\begin{aligned}
&\max_{\eta,\gamma^{\pm}} && \sum_{i,r}\eta_{i,r}b_{i,r} \\
&\text{subject to :} && \forall i,r,t \;\; \eta_{i,r},\gamma^{+}_{t,r},\gamma^{-}_{t,r}\geq 0 \\
& && \forall i \;\; \sum_r \eta_{i,r}=1 \\
& && \sum_{t,r}(\gamma^{+}_{t,r}+\gamma^{-}_{t,r})=\beta \\
& && \forall r,t \;\; \sum_i h_t(x_i)[\delta_{y_i,r}-\eta_{i,r}]=-\gamma^{+}_{t,r}+\gamma^{-}_{t,r}
\end{aligned}$$

Both programs ($p = 1$ and $0 = \infty$) can be now solved using standard linear program packages.

5.2 Design of continuous codes using Quadric Programming

We now discuss in detail Eq. (14) for the case $p = 2$. For convenience we use the square of the norm of the matrix (instead the norm itself). Therefore, the primal program becomes,

$$\begin{aligned}
&\min_{M,\xi} && \frac{1}{2}\beta\|M\|_2^2+\sum_{i=1}^{m}\xi_i && (15)\\
&\text{subject to :} && \forall i,r \;\; \xi_i+\bar{h}(x_i)\cdot\bar{M}_{y_i}-\bar{h}(x_i)\cdot\bar{M}_r\geq b_{i,r}
\end{aligned}$$

We solve the optimization problem by finding a saddle point of the Lagrangian :

$$\begin{aligned}
\mathcal{L}(M,\xi_i,\eta) &= \frac{1}{2}\beta\sum_r\|\bar{M}_r\|_2^2+\sum_{i=1}^{m}\xi_i \\
&+ \;\; \textstyle\sum_{i,r}\eta_{i,r}\left[\bar{h}(x_i)\cdot\bar{M}_r-\bar{h}(x_i)\cdot\bar{M}_{y_i}-\xi_i+b_{i,r}\right] \\
&\text{subject to :}\forall i,r \;\; \eta_{i,r}\geq 0 \qquad (16)
\end{aligned}$$

The saddle point we are seeking is a minimum for the primal variables (M,ξ), and the maximum for the dual ones (η). To find the minimum over the primal variables we require,

$$\frac{\partial}{\partial\xi_i}\mathcal{L}=1-\sum_r\eta_{i,r}=0 \quad\Rightarrow\quad \sum_r\eta_{i,r}=1 \qquad (17)$$

Similarly, for $\bar{M}_r$ we require,

$$\frac{\partial}{\partial\bar{M}_r}\mathcal{L}=0 \qquad (18)$$

$$\Rightarrow \sum_i\eta_{i,r}\bar{h}(x_i)-\sum_{i,y_i=r}\bar{h}(x_i)\underbrace{\left(\sum_{r'}\eta_{i,r'}\right)}_{=1}+\beta\bar{M}_r=0$$

$$\Rightarrow \bar{M}_r=\beta^{-1}\left[\sum_i\bar{h}(x_i)(\delta_{y_i,r}-\eta_{i,r})\right] \qquad (19)$$

Eq. (19) implies that when the optimum of the objective function is achieved, each row of the matrix M is a linear combination of $\bar{h}(x_i)$. We say that an example i is a *support pattern* for class r if the coefficient $(\delta_{y_i,r}-\eta_{i,r})$ of $\bar{h}(x_i)$ in Eq. (19) is not zero. There are two settings for which an example i can be a support pattern for class r. The first case is when the label y_i of an example is equal to r, then the ith example is a support pattern if $\eta_{i,r}<1$. The second case is when the label y_i of the example is different from r, then the ith pattern is a support pattern if $\eta_{i,r}>0$.

Loosely speaking, since for all i and r we have $\eta_{i,r}\geq 0$ and $\sum_r\eta_{i,r}=1$, the variable $\eta_{i,r}$ can be viewed as a distribution over the labels for each example. An example i affects the solution for M (Eq. (19)) if and only if $\bar{\eta}_i$ in *not* a point distribution concentrating on the correct label y_i. Thus, only the questionable patterns contribute to the learning process.

We develop the Lagrangian using only the dual variables. Substituting Eqs. (17) and (19) into Eq. (16) and using various algebraic manipulations, we obtain that the target function of the dual program is,

$$\begin{aligned}
\mathcal{Q}(\eta) &= \\
&-\frac{1}{2}\beta^{-1}\sum_{i,j}\bar{h}(x_i)\cdot\bar{h}(x_j)\sum_r(\delta_{y_i,r}-\eta_{i,r})(\delta_{y_j,r}-\eta_{j,r}) \\
&+\sum_{i,r}\eta_{i,r}b_{i,r}
\end{aligned}$$

(Details are omitted due to the lack of space.) Let $\bar{1}_i$ be the vector with all components zero, except for the ith component which is equal to one, and let $\bar{1}$ be the vector whose components are all one. Using this notation we can rewrite the dual program in vector form as

$$\begin{aligned}
&\max_{\eta} && \mathcal{Q}(\eta) \\
&\text{subject to :} && \forall r \;\; \bar{\eta}_i\geq 0 \text{ and } \bar{\eta}_i\cdot\bar{1}=1 \qquad (20)
\end{aligned}$$

where

$$\begin{aligned}
\mathcal{Q}(\eta) &= \\
&-\frac{1}{2}\beta^{-1}\sum_{i,j}\left[\bar{h}(x_i)\cdot\bar{h}(x_j)\right]\left[(\bar{1}_{y_i}-\bar{\eta}_i)\cdot(\bar{1}_{y_j}-\bar{\eta}_j)\right] \\
&+\sum_i\bar{\eta}_i\cdot\bar{b}_i
\end{aligned}$$

It is easy to verify that $\mathcal{Q}(\eta)$ is strictly convex in η. Since the constraints are linear the above problem has a single optimal solution and therefore QP methods can be used to solve it. In Sec. 6 we describe a memory efficient algorithm for solving this special QP problem.

To simplify the equations we denote by $\bar{\tau}_i=\bar{1}_{y_i}-\bar{\eta}_i$ the difference between the correct point distribution and the distribution obtained by the optimization problem, Eq. (19) becomes,

$$\bar{M}_r=\beta^{-1}\sum_i\bar{h}(x_i)\tau_{i,r} \qquad (21)$$

Since we look for the value of the variables which maximize the objective function $\mathcal{Q}$ (and not the optimum of $\mathcal{Q}$ itself), we can omit constants and write the dual problem given by Eq. (20) as,

$$\begin{aligned}
&\max_{\tau} && \mathcal{Q}(\tau) \\
&\text{subject to :} && \forall r \;\; \bar{\tau}_i\leq\bar{1}_{y_i} \text{ and } \bar{\tau}_i\cdot\bar{1}=0 \qquad (22)
\end{aligned}$$

where

$$\mathcal{Q}(\tau)=-\frac{1}{2}\beta^{-1}\sum_{i,j}[\bar{h}(x_i)\cdot\bar{h}(x_j)](\bar{\tau}_i\cdot\bar{\tau}_j)-\sum_i\bar{\tau}_i\cdot\bar{b}_i$$

Finally, the classifier $H(x)$ can be written in terms of the variable τ as,

$$\begin{aligned} H(x) &= \arg\max_r \left\{ \bar{h}(x) \cdot \bar{M}_r \right\} \\ &= \arg\max_r \left\{ \bar{h}(x) \cdot \left[\beta^{-1} \sum_i \bar{h}(x_i) \tau_{i,r} \right] \right\} \\ &= \arg\max_r \left\{ \beta^{-1} \sum_i \tau_{i,r} \left[\bar{h}(x) \cdot \bar{h}(x_i) \right] \right\} \\ &= \arg\max_r \left\{ \sum_i \tau_{i,r} \left[\bar{h}(x) \cdot \bar{h}(x_i) \right] \right\} . \end{aligned} \quad (23)$$

As in Support Vector Machines, the dual program and the classification algorithm depend *only* on inner products of the form $\bar{h}(x_i) \cdot \bar{h}(x)$. Therefore, we can perform the calculations in some high dimensional inner-product space $\mathcal{Z}$ using a transformation $\bar{\phi} : \mathbb{R}^l \rightarrow \mathcal{Z}$. We thus replace the inner-product in Eq. (22) and in Eq. (23) with a general inner-product kernel K that satisfies Mercer conditions [21]. The general dual program is therefore,

$$\begin{array}{ll} \max_\tau & \mathcal{Q}(\tau) \\ \text{subject to :} & \forall i \quad \bar{\tau}_i \leq \bar{1}_{y_i} \text{ and } \bar{\tau}_i \cdot \bar{1} = 0 \end{array} \quad (24)$$

where :

$$\mathcal{Q}(\tau) = -\frac{1}{2}\beta^{-1} \sum_{i,j} K\left(\bar{h}(x_i), \bar{h}(x_j)\right) \, (\bar{\tau}_i \cdot \bar{\tau}_j) - \sum_i \bar{\tau}_i \cdot \bar{b}_i$$

and the classification rule $H(x)$ becomes,

$$H(x) = \arg\max_r \left\{ \sum_i \tau_{i,r} K\left(\bar{h}(x), \bar{h}(x_i)\right) \right\} \quad (25)$$

The general framework for designing output codes using the QP program described above, also provides, as a special case, a new algorithm for building multiclass Support Vectors Machines. Assume that the instance space is the vector space $\mathbb{R}^n$ and define $\bar{h}(\bar{x}) \stackrel{\text{def}}{=} \bar{x}$ (thus $l = n$), then the primal program in Eq. (15) becomes

$$\begin{array}{ll} \min\limits_{M,\xi} & \frac{1}{2}\beta \|M\|_2^2 + \sum_{i=1}^m \xi_i \\ \text{subject to :} & \forall i, r \quad \xi_i + \bar{x}_i \cdot \bar{M}_{y_i} - \bar{x}_i \cdot \bar{M}_r \geq b_{i,r} \end{array} \quad (26)$$

Note that for $k = 2$ Eq. (26) reduces to the primal program of SVM, if we take $\bar{M}_1 = -\bar{M}_2$ and $C = \beta^{-1}$. We would also like to note that this special case is reminiscent of the multiclass approach for SVM's suggested by Weston and Watkins [22]. Their approach compared the confidence $K(x, \bar{M}_y)$ to the confidences of *all* other labels $K(x, \bar{M}_r)$ and had $m(k-1)$ slack variables in the primal problem. In contrast, in our framework the confidence $K(x, \bar{M}_y)$ is compared to $\max_{r \neq y} K(x, \bar{M}_r)$ and has only m slack variables in the primal program.

In Table 1 we summarize the properties of the programs discussed above. As shown in the table, the advantage of using l_2 in the objective function is that the number of variables in the dual problem in only a function of on k and m and does *not* depend on the number columns l in M. The number of columns in M only affects the evaluation of the inner-product kernel K.

		l_1	l_2	l_∞
Primal	Variables	$m + 2kl$	$m + kl$	$m + kl + 1$
	0-Constraints	0	0	0
	Constraints	$km + 2kl$	km	$km + 2kl$
Dual	Variables	$km + 2kl$	km	$km + 2kl$
	0-Constraints	$km + 2kl$	km	$km + 2kl$
	Constraints	$m + 2kl$	m	$m + kl + 1$

Table 1: Summary of the sizes of the optimization problems for different norms. (See Appendix A for the definitions of the constraints in linear programming.)

The formalism given by Eq. (14) can also be used to construct the code matrix incrementally (column by column). We now outline the incremental (inductive) approach. However, we would like to note that this method only applies when $K(\bar{v}, \bar{u}) = \bar{v} \cdot \bar{u}$. In the first step of the incremental algorithm, we are given a single binary classifier $h_1(x)$ and we need to construct the first column of M. We rewrite Eq. (14) in a scalar form and obtain,

$$\begin{array}{ll} \min_M & \beta \|M\|_p + \sum\limits_{i=1}^m \xi_i \\ \text{subject to :} & \forall i, r \quad h_1(x_i) M_{y_i} - h_1(x_i) M_r \geq b_{i,r} - \xi_i \, . \end{array} \quad (27)$$

Here, $\beta \geq 0$ is a given constant and $b_{i,r} = 1 - \delta_{y_i,r}$, as before. For the rest of the columns we assume inductively that $h_1(x), \ldots, h_l(x)$ have been provided and the first l columns of the matrix M have been found. In addition, we are provided with a new binary classifier $h_{l+1}(x) \stackrel{\text{def}}{=} h'(x)$ for the next column. We need to find a new column of M (indexed $l+1$). We substitute the new classifier and the matrix in Eq. (13) and get,

$$\begin{aligned} \forall i \quad & \max_r \{ \bar{h}(x_i) \cdot \bar{M}_r + h'(x_i) M'_r + 1 - \delta_{y_i,r} \} - \\ & \left(\bar{h}(x_i) \cdot \bar{M}_{y_i} + h'(x_i) M'_{y_i} \right) = \xi_i \, . \end{aligned}$$

The constraints appearing in Eq. (14) now become

$$\begin{aligned} & \bar{h}(x_i) \cdot \bar{M}_{y_i} + h'(x_i) M'_{y_i} - \bar{h}(x_i) \cdot \bar{M}_r - h'(x_i) M'_r \geq \\ & \qquad 1 - \delta_{y_i,r} - \xi_i \\ & \Rightarrow h'(x_i) M'_{y_i} - h'(x_i) M'_r \geq \\ & \qquad -[\bar{h}(x_i) \cdot \bar{M}_{y_i} - \bar{h}(x_i) \cdot \bar{M}_r] + 1 - \delta_{y_i,r} - \xi_i \, . \end{aligned}$$

We now redefine $b_{i,r}$ to be $-[\bar{h}(x_i) \cdot \bar{M}_{y_i} - \bar{h}(x_i) \cdot \bar{M}_r] + 1 - \delta_{y_i,r}$. It is straightforward to verify that this definition of $b_{i,r}$ results in an equation of the same form of Eq. (27). We can thus apply the same algorithms designed for the "batch" case. In the case of l_1 and l_∞, this construction decomposes a single problem into l sub-problems with fewer variables and constraints. However, for l_2 the size of the program remains the same while we lose the ability to use kernels. We therefore concentrate on the batch case for which we need to find the entire matrix at once.

6 An efficient algorithm for the QP problem

The quadratic program presented in Eq. (24) can be solved using standard QP techniques. As shown in Table 1 the dual program depends on mk variables and has $km + m$ constraints all together. Converting the dual program in Eq. (24) to a standard QP form requires storing and manipulating a matrix with $(mk)^2$ elements. Clearly, this would prohibit applications of non-trivial size. We now introduce a memory efficient algorithm for solving the quadratic optimization problem given by Eq. (24).

First, note that the constraints in Eq. (24) can be divided into m disjoint subsets $\{\bar{\tau}_i \leq \bar{1}_{y_i}, \bar{\tau}_i \cdot \bar{1} = 0\}_{i=1}^m$. The algorithm we describe works in rounds. On each round it picks a single set $\{\bar{\tau}_i \leq \bar{1}_{y_i}, \bar{\tau}_i \cdot \bar{1} = 0\}$, and modifies $\bar{\tau}_i$ so as to optimize the reduced optimization problem. The algorithm is reminiscent of Platt's SMO algorithm [15]. Note, however, that our algorithm optimizes *one* example on each round, and not two as in SMO.

Let us fix an example index p and write the objective function only in terms of the variables $\bar{\tau}_p$. For brevity, let $K_{i,j} = K\left(\bar{h}(x_i), \bar{h}(x_j)\right)$. We isolate $\bar{\tau}_p$ in $\mathcal{Q}$.

$$\begin{aligned}
\mathcal{Q}_p(\bar{\tau}_p) &\overset{\text{def}}{=} -\frac{1}{2}\beta^{-1}\sum_{i,j} K_{i,j}(\bar{\tau}_i \cdot \bar{\tau}_j) - \sum_i \bar{\tau}_i \cdot \bar{b}_i \\
&= -\frac{1}{2}\beta^{-1}K_{p,p}(\bar{\tau}_p \cdot \bar{\tau}_p) - \beta^{-1}\sum_{i \neq p} K_{i,p}(\bar{\tau}_p \cdot \bar{\tau}_i) \\
&\quad -\frac{1}{2}\beta^{-1}\sum_{i,j \neq p} K_{i,j}(\bar{\tau}_i \cdot \bar{\tau}_j) - \bar{\tau}_p \cdot \bar{b}_p - \sum_{i \neq p} \bar{\tau}_i \cdot \bar{b}_i \\
&= -\frac{1}{2}\beta^{-1}K_{p,p}(\bar{\tau}_p \cdot \bar{\tau}_p) - \bar{\tau}_p \cdot [\bar{b}_p + \beta^{-1}\sum_{i \neq p} K_{i,p}\bar{\tau}_i] \\
&\quad +[-\frac{1}{2}\beta^{-1}\sum_{i,j \neq p} K_{i,j}(\bar{\tau}_i \cdot \bar{\tau}_j) - \sum_{i \neq p} \bar{\tau}_i \cdot \bar{b}_i] \\
&= -\frac{1}{2}A_p(\bar{\tau}_p \cdot \bar{\tau}_p) - \bar{B}_p \cdot \bar{\tau}_p + C_p \qquad (28)
\end{aligned}$$

where,

$$\begin{aligned}
A_p &= \beta^{-1}K_{p,p} > 0 \qquad (29) \\
\bar{B}_p &= \bar{b}_p + \beta^{-1}\sum_{i \neq p} K_{i,p}\bar{\tau}_i \qquad (30) \\
C_p &= -\frac{1}{2}\beta^{-1}\sum_{i,j \neq p} K_{i,j}(\bar{\tau}_i \cdot \bar{\tau}_j) - \sum_{i \neq p} \bar{\tau}_i \cdot \bar{b}_i \qquad (31)
\end{aligned}$$

For brevity, we will omit the index p and drop constants (that do not affect the solution). The reduced optimization has k variables and $k + 1$ constraints,

$$\begin{aligned}
&max_\tau \qquad \mathcal{Q}(\bar{\tau}) = -\frac{1}{2}A(\bar{\tau} \cdot \bar{\tau}) - \bar{B} \cdot \bar{\tau} \\
&\text{subject to :} \quad \bar{\tau} \leq \bar{1}_y \text{ and } \bar{\tau} \cdot \bar{1} = 0 \qquad (32)
\end{aligned}$$

Although this program can be solved using a standard QP technique, it still requires large amount of memory when k is large, and a straightforward solution is also time consuming. Furthermore, this problem constitutes the core and inner-loop of the algorithm. We therefore further develop the algorithm and describe a more efficient method for solving Eq. (32). We write $\mathcal{Q}(\bar{\tau})$ in Eq. (32) using a completion to quadratic form,

$$\begin{aligned}
\mathcal{Q}(\bar{\tau}) &= -\frac{1}{2}A(\bar{\tau} \cdot \bar{\tau}) - \bar{B} \cdot \bar{\tau} \\
&= -\frac{1}{2}A[(\bar{\tau} + \frac{\bar{B}}{A}) \cdot (\bar{\tau} + \frac{\bar{B}}{A})] + \frac{\bar{B} \cdot \bar{B}}{2A}
\end{aligned}$$

Since $A > 0$ the program from Eq. (32) becomes,

$$\begin{aligned}
&\min_\nu \qquad \mathcal{Q}(\bar{\nu}) = \|\bar{\nu}\|^2 \\
&\text{subject to :} \quad \bar{\nu} \leq \bar{D} \text{ and } \bar{\nu} \cdot \bar{1} = \bar{D} \cdot \bar{1} - 1
\end{aligned}$$

where,

$$\bar{\nu} = \bar{\tau} + \frac{\bar{B}}{A} \qquad \bar{D} = \frac{\bar{B}}{A} + \bar{1}_y \qquad (33)$$

In Sec. 6.1 we discuss an analytic solution to Eq. (33) and in Sec. 6.2 we describe a time efficient algorithm for computing the analytic solution.

6.1 An analytic solution

While the algorithmic solution we describe in this section is simple to implement and efficient, its derivation is quite complex. Before describing the analytic solution to Eq. (33), we would like to give some intuition on our method. Let us fix some vector $\bar{D}$ and denote $\mu = \bar{\nu} \cdot \bar{1}$. First note that $\bar{\nu} = \bar{D}$ is not a feasible point since the constraint $\bar{\nu} \cdot \bar{1} = \bar{D} \cdot \bar{1} - 1$ is not satisfied. Hence for any feasible point some of the constraints $\bar{\nu} \leq \bar{D}$ are not tight. Second, note that the differences between the bounds D_r and the variables ν_r sum to one. Let us induce a uniform distribution over the components of $\bar{\nu}$. Then, the variance of $\bar{\nu}$ is

$$\sigma^2 = \mathbf{E}[\nu^2] - [\mathbf{E}\nu]^2 = \frac{1}{k}\|\bar{\nu}\|^2 - \frac{1}{k^2}\mu^2$$

Since the expectation μ is constrained to a given value, the optimal solution is the vector achieving the smallest variance. That is, the components of of $\bar{\nu}$ should attain similar values, as much as possible, under the inequality constraints $\bar{\nu} \leq \bar{D}$. In Fig. 1 we illustrate this motivation. We picked $\bar{D} = (1.0, 0.2, 0.6, 0.8, 0.6)$ and show plots for two different feasible values for $\bar{\nu}$. The x-axis is the index r of the point and the y-axis designates the values of the components of $\bar{\nu}$. The norm of $\bar{\nu}$ on the plot on the right hand side plot is smaller than the norm of the plot on the left hand side. The right hand side plot is the optimal solution for $\bar{\nu}$. The sum of the lengths of the arrows in both plots is $\bar{D} \cdot \bar{1} - \bar{\nu} \cdot \bar{1}$. Since both sets of points are feasible, they satisfy the constraint $\bar{\nu} \cdot \bar{1} = \bar{D} \cdot \bar{1} - 1$. Thus, the sum of the lengths of the "arrows" in both plots is one. We exploit this observation in the algorithm we describe in the sequel.

We therefore seek a feasible vector $\bar{\nu}$ whose most of its components are equal to some threshold θ. Given θ we define a vector $\bar{\nu}$ whose its rth component equal to the minimum between θ and D_r, hence the inequality constraints are satisfied. We define

$$\nu_r^\theta = \begin{cases} \theta & \theta \leq D_r \\ D_r & \theta > D_r \end{cases} \qquad (34)$$

We denote by

$$F(\theta) = \bar{\nu}^\theta \cdot \bar{1} = \sum_{r=1}^k \nu_r^\theta \; .$$

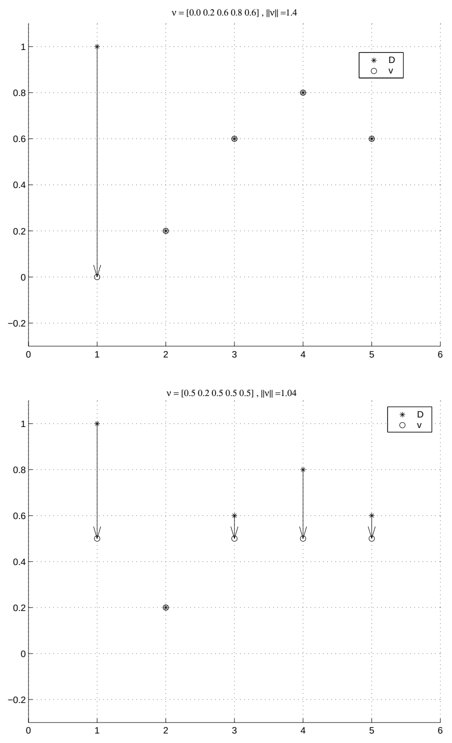

Figure 1: An illustration of two feasible points for the reduced optimization problem with $\bar{D} = (1, 0.2, 0.6, 0.8, 0.6)$. The x-axis is the index of the point, and the y-axis denotes the values $\bar{\nu}$. The bottom plot has a smaller variance hence it achieves a better value for $\mathcal{Q}$.

Using F, the equality constraint from Eq. (33) becomes $F(\theta) = \bar{D} \cdot \bar{1} - 1$.

Let us assume without loss of generality that the components of the vector $\bar{\nu}$ are given in a descending order, $D_1 \geq D_2 \geq \ldots D_k$ (this can be done in $k \log k$ time). Let $D_{k+1} = -\infty$ and $D_0 = \infty$. To prove the main theorem of this section we need the following lemma.

Lemma 3 *$F(\theta)$ is piecewise linear with a slope r in each range (D_{r+1}, D_r) for $r = 0, \ldots, k$.*

Proof: Let us develop $F(\theta)$.

$$\begin{aligned} F(\theta) &= \sum_{r=1}^{k} \begin{cases} \theta & \theta \leq D_r \\ D_r & \theta > D_r \end{cases} \\ &= \sum_{r=1}^{k} \{ [\![\theta > D_r]\!] D_r + [\![\theta \leq D_r]\!] \theta \} \\ &= \sum_{r=1}^{k} [\![\theta > D_r]\!] D_r + \theta \sum_{r=1}^{k} [\![\theta \leq D_r]\!] \end{aligned}$$

Note that if $\theta > D_r$ then $\theta > D_u$ for all $u \geq r$. Also, the equality $\sum_{r'=1}^{k} [\![\theta \leq D_{r'}]\!] = r$ holds for each θ in the range $D_{r+1} < \theta < D_r$. Thus, for $D_{r+1} < \theta < D_r$ $(r = 0 \cdots k)$, the function $F(\theta)$ has the form,

$$F(\theta) = \sum_{u=r+1}^{k} D_u + r\theta \tag{35}$$

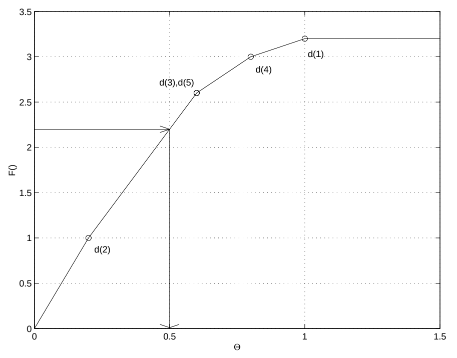

Figure 2: An illustration of the solution of the QP problem using the inverse of $F(\theta)$ for $\bar{D} = (1, 0.2, 0.6, 0.8, 0.6)$. The optimal value is the solution for the equation $F(\theta) = 2.2$ which is 0.5.

This completes the proof. ■

Corollary 4 *There exists a unique $\theta_0 \leq D_1$ such that*

$$F(\theta_0) = \bar{D} \cdot \bar{1} - 1 \; .$$

Proof: From Eq. (35) we conclude that $F(\theta)$ is strictly increasing and continuous in the range $\theta \leq D_1$. Therefore, $F(\theta)$ has an inverse in that range, using the theorem that every strictly increasing and continuous function has an inverse. Since $F(\theta) = k\theta$ for $\theta \leq D_k$ then $F(\theta) \to -\infty$ as $\theta \to -\infty$. Hence, the range of F for the interval $(-\infty, D_1]$ is the interval $(-\infty, \bar{D} \cdot \bar{1}]$ which clearly contains $\bar{D} \cdot \bar{1} - 1$. Thus $\theta_0 \overset{\text{def}}{=} F^{-1}(\bar{D} \cdot \bar{1} - 1) \in [-\infty, D_1]$ as needed. Uniqueness of θ_0 follows the fact that the function F is a one-to-one mapping onto $(-\infty, \bar{D} \cdot \bar{1}]$. ■

We now can prove the main theorem of this section.

Theorem 5 *Let θ_0 be the unique solution of $F(\theta) = \bar{D} \cdot \bar{1} - 1$. Then $\bar{\nu}^{\theta_0}$ is the optimum value of the optimization problem stated in Eq. (33).*

The theorem tells us that the optimum value of Eq. (33) is of the form defined by Eq. (34) and that there is exactly one value of θ for which the equality constraint $F(\theta) = \bar{\nu}^{\theta} \cdot \bar{1} = \bar{D} \cdot \bar{1} - 1$ holds. A plot of $F(\theta)$ and the solution for θ from Fig. 1 are shown in Fig. 2.

Proof: Corollary 4 implies that a solution exists and is unique. Note also that from definition of θ_0 we have that the vector $\bar{\nu}^{\theta_0}$ is a feasible point of Eq. (33). We now prove that $\bar{\nu}^{\theta_0}$ is the optimum of Eq. (33) by showing that $\|\bar{\nu}\|^2 > \|\bar{\nu}^{\theta_0}\|^2$ for all feasible points $\bar{\nu} \neq \bar{\nu}^{\theta}$.

Assume, by contradiction, that there is a vector $\bar{\nu}$ such that $\|\bar{\nu}\|^2 \leq \|\bar{\nu}^{\theta_0}\|^2$. Let $\bar{\epsilon} \overset{\text{def}}{=} \bar{\nu} - \bar{\nu}^{\theta} \neq \bar{\mathbf{0}}$, and define $I \overset{\text{def}}{=} \{r : \nu_r^{\theta} = D_r\} = \{r : \theta_0 > D_r\}$. Since both $\bar{\nu}$ and $\bar{\nu}^{\theta_0}$ satisfy the equality constraint of Eq. (33), we have,

$$\begin{aligned} \bar{\nu} \cdot \bar{1} = \bar{\nu}^{\theta_0} \cdot \bar{1} \quad &\Rightarrow \quad (\bar{\nu} - \bar{\nu}^{\theta_0}) \cdot \bar{1} = 0 \\ &\Rightarrow \quad \bar{\epsilon} \cdot \bar{1} = \sum_{r=1}^{k} \epsilon_r = 0 \end{aligned} \tag{36}$$

```
Input : $\bar{D}$.
Initialize $\nu = \bar{D}$.
Define $D_{k+1} = -\infty$.
Sort the components of $\bar{D}$, such that $D_{i_1} \geq D_{i_2} \ldots D_{i_{k+1}}$.
Define $r = 0$; $\Phi(1) = 1$.
While $\Phi(r+1) > 0$
    • $r \leftarrow r + 1$.
    • $\Phi(r+1) = \Phi(r) - r(D_{i_r} - D_{i_{r+1}})$.        Eq. (39)
Compute $\theta_0 \leftarrow D_{i_r} - \frac{\Phi(r)}{r}$.        Eq. (40)
For $q = 1, \ldots, r$ assign $\nu_{i_q} = \theta_0$.
Return $\bar{\nu}$.
```

Figure 3: The algorithm for finding the optimal solution of the reduced quadratic program (Eq. (33)).

Since $\bar{\nu}$ is a feasible point we have $\bar{\nu} = \bar{\nu}^{\theta_0} + \bar{\epsilon} \leq \bar{D}$. Also, by the definition of the set I we have that $\nu_r^{\theta} = D_r$ for all $r \in I$. Combining the two properties we get,

$$\epsilon_r \leq 0 \ \text{ for all } r \in I \tag{37}$$

We start with the simpler case of $\epsilon_r = 0$ for all $r \in I$. In this case, $\bar{\nu}$ differs from $\bar{\nu}^{\theta_0}$ only on a subset of the coordinates $r \notin I$. However, for these coordinates the components of $\bar{\nu}^{\theta_0}$ are equal to θ_0, thus we obtain a zero variance from the constant vector whose components are all θ_0. Therefore, no other feasible vector can achieve a better variance. Formally, since $\epsilon_r = 0$ for all $r \in I$, then the terms for $r \in I$ cancel each other,

$$\begin{aligned} \|\bar{\nu}\|^2 - \|\bar{\nu}^{\theta_0}\|^2 &= \sum_{r=1}^{k} (\nu_r^{\theta_0} + \epsilon_r)^2 - \sum_{r=1}^{k} (\nu_r^{\theta_0})^2 \\ &= \sum_{r \notin I} (\nu_r^{\theta_0} + \epsilon_r)^2 - \sum_{r \notin I} (\nu_r^{\theta_0})^2 \ . \end{aligned}$$

From the definition of $\bar{\nu}^{\theta_0}$ in Eq. (34) we get that $\nu_r^{\theta_0} = \theta_0$ for all $r \notin I$,

$$\begin{aligned} \|\bar{\nu}\|^2 - \|\bar{\nu}^{\theta_0}\|^2 &= \sum_{r \notin I} (\theta_0 + \epsilon_r)^2 - \sum_{r \notin I} \theta_0^2 \\ &= 2\theta_0 \sum_{r \notin I} \epsilon_r + \sum_{r \notin I} \epsilon_r^2 \ . \end{aligned}$$

We use now the assumption that $\epsilon_r = 0$ for all $r \in I$ and the equality $\sum_{r=1}^{k} \epsilon_r = 0$ (Eq. (36)) to obtain,

$$\|\bar{\nu}\|^2 - \|\bar{\nu}^{\theta_0}\|^2 = 2\theta_0 \sum_{r=1}^{k} \epsilon_r + \sum_{r=1}^{k} \epsilon_r^2 = \sum_{r=1}^{k} \epsilon_r^2 > 0$$

and we get a contradiction since $\bar{\epsilon} \neq \bar{\mathbf{0}}$.

We now turn to prove the complementary case in which $\sum_{r \in I} \epsilon_r < 0$. Since $\sum_{r \in I} \epsilon_r < 0$, then there exists $u \in I$ such that $\epsilon_u < 0$. We use again Eq. (36) and conclude that there exists also $v \notin I$ such that $\epsilon_v > 0$. Let us assume without loss of generality that $\epsilon_u + \epsilon_v < 0$ (The case $\epsilon_u + \epsilon_v \geq 0$ follows analogously by switching the roles of u and v). Define $\bar{\nu}'$ as follows,

$$\nu'_r = \begin{cases} \nu_u + \epsilon_v & r = u \\ \nu_v - \epsilon_v & r = v \\ \nu_r & \text{otherwise} \end{cases}$$

```
Input : $\{(\bar{h}(x_1), y_1), \ldots, (\bar{h}(x_m), y_m)\}$.
Choose $\{\bar{\tau}_i\}$ - a feasible point for Eq. (24).
Iterate.
    • Choose an example $p$
    • Compute $A_p$ and $B_p$                    Eqs. (29) and (30)
    • $\bar{D}_p \leftarrow \frac{\bar{B}_p}{A_p} + \bar{1}_{y_p}$.          Eq. (33)
    • Compute $\theta = F^{-1}(\bar{D}_p \cdot \bar{1} - 1)$        Fig. 3
    • $\bar{\tau}_p \leftarrow \bar{\nu}_p^{\theta} - \frac{\bar{B}_p}{A_p}$             Eq. (33)
Output the final hypothesis:                     Eq. (25)
    $$H(x) = \arg\max_r \left\{ \sum_i \tau_{i,r} K\left(\bar{h}(x), \bar{h}(x_i)\right) \right\}$$
```

Figure 4: A skeleton of the algorithm for finding a classifier based on an output code by solving the quadratic program defined in Eq. (24).

The vector $\bar{\nu}'$ satisfies the constraints of Eq. (33) since $\nu'_u = \nu_u + \epsilon_v = D_u + \epsilon_u + \epsilon_v < D_u$ and $\nu'_v = \nu_v - \epsilon_v = \theta_0 + \epsilon_v - \epsilon_v = \theta_0 \leq D_v$. Since $\bar{\nu}$ and $\bar{\nu}'$ are equal except for their u and v components we get,

$$\|\bar{\nu}'\|^2 - \|\bar{\nu}\|^2 \quad = \quad [(\nu'_u)^2 + (\nu'_v)^2] - [(\nu_u)^2 + (\nu_v)^2]$$

Substituting the values for ν'_u and ν'_v from the definition of $\bar{\nu}'$ we obtain,

$$\begin{aligned} & \|\bar{\nu}'\|^2 - \|\bar{\nu}\|^2 \\ &= [(\nu_u + \epsilon_v)^2 + (\nu_v - \epsilon_v)^2] - [(\nu_u)^2 + (\nu_v)^2] \\ &= \epsilon_v^2 + 2\nu_u\epsilon_v + \epsilon_v^2 - 2\nu_v\epsilon_v \\ &= 2\epsilon_v^2 + 2(\nu_u - \nu_v)\epsilon_v \end{aligned}$$

Using the definition of $\bar{\nu}$ and $\bar{\nu}^{\theta_0}$ for $\nu_u = \nu_u^{\theta_0} + \epsilon_u = D_u + \epsilon_u$ and for $\nu_v = \nu_v^{\theta_0} + \epsilon_v = \theta_0 + \epsilon_v$ we obtain,

$$\begin{aligned} \|\bar{\nu}'\|^2 - \|\bar{\nu}\|^2 &= 2\epsilon_v^2 + 2(D_u + \epsilon_u - \theta_0 - \epsilon_v)\epsilon_v \\ &= 2(D_u + \epsilon_u - \theta_0)\epsilon_v \\ &= 2\epsilon_u\epsilon_v + 2(D_u - \theta_0)\epsilon_v \end{aligned}$$

The first term of the bottom equation is negative since $\epsilon_u < 0$ and $\epsilon_v > 0$. Also $u \in I$, hence $\theta_0 > D_u$ and the second term is also negative. We thus get,

$$\|\bar{\nu}'\|^2 - \|\bar{\nu}\|^2 \quad < \quad 0 \ .$$

which is a contradiction. ■

6.2 An efficient algorithm for computing the analytic solution

The optimization problem of Eq. (33) can be solved using standard QP methods, and interior point methods in particular [10]. For these methods the computation time is $\Theta(k^2)$. In this section we give an algorithm for solving that optimization problem in $O(k \log k)$ time, by solving the equation $F(\theta) = \bar{D} \cdot \bar{1} - 1$.

As before, we assume that the components of the vector $\bar{\nu}$ are given in a descending order, $D_1 \geq D_2 \geq \ldots D_k$ and we denote $D_{k+1} = -\infty$. The algorithm searches for the interval $[D_{r+1}, D_r)$ which contains θ_0. We now use simple algebraic manipulations to derive the search scheme for θ_0. Since $F(D_1) = F(\theta_0) + 1$, then $\theta_0 \in [D_{r+1}, D_r)$, iff

$$1 > F(D_1) - F(D_r) \text{ and } F(D_1) - F(D_{r+1}) \geq 1 \; .$$

For convenience, we define the potential function

$$\Phi(r) = 1 - [F(D_1) - F(D_r)] \; , \tag{38}$$

and obtain,

$$\theta_0 \in [D_{r+1}, D_r) \quad \Leftrightarrow \quad \Phi(r) > 0 \text{ and } \Phi(r+1) \leq 0$$

Also note that,

$$\begin{aligned} \Phi(r) - \Phi(r+1) &= \{1 - [F(D_1) - F(D_r)]\} \\ &- \{1 - [F(D_1) - F(D_{r+1})]\} \\ &= F(D_r) - F(D_{r+1}) \; . \end{aligned}$$

Recall that the function $F(\theta)$ is linear in each interval $[D_{r+1}, D_r)$ with a slope r (Lemma 3), hence,

$$F(D_r) - F(D_{r+1}) = r(D_r - D_{r+1})$$
$$\Rightarrow \quad \Phi(r+1) = \Phi(r) - r(D_r - D_{r+1}) \; . \tag{39}$$

To solve the equation $F(\theta) = \bar{D} \cdot \bar{1} - 1$, we first find r such that $\Phi(r) > 0$ and $\Phi(r+1) \leq 0$, which implies that $\theta_0 \in [D_{r+1}, D_r)$. Using Eq. (38) and the equation $F(D_1) = F(\theta_0) + 1$ we get,

$$F(D_r) - F(\theta_0) = \Phi(r) \; .$$

Using the linearity of $F(\theta)$ we obtain,

$$F(D_r) - F(\theta_0) = r(D_r - \theta_0) \quad \Rightarrow \quad r(D_r - \theta_0) = \Phi(r)$$

therefore

$$\theta_0 = D_r - \frac{\Phi(r)}{r} \; . \tag{40}$$

The complete algorithm is described in Fig. 3. Since it takes $O(k \log k)$ time to sort the vector $\bar{D}$ and another $O(k)$ time for the loop search, the total run time is $O(k \log k)$.

We are finally ready to give the algorithm for solving learning problem described by Eq. (24). Since the output code is constructed of the supporting patterns we term our algorithm SPOC for Support Pattern Output Coding. The SPOC algorithm is described in Fig. 4. We have also developed methods for choosing an example p to modify on each round and a stopping criterion for the entire optimization algorithm. Due to lack of space we omit the details which will appear in a full paper.

We have performed preliminary experiments with synthetic data in order to check the actual performance of our algorithm. We tested the special case corresponding to multiclass SVM by setting $\bar{h}(x) = x$. The code matrices we test are of $k = 4$ rows (classes) and $l = 2$ columns. We varied the size of the training set size from $m = 10$ to $m = 250$. The examples were generated using the uniform distribution over $[-1, 1] \times [-1, 1]$. The domain $[-1, 1] \times [-1, 1]$ was partitioned into four quarters of equal size: $[-1, 0] \times [-1, 0]$, $[-1, 0] \times [0, 1]$, $[0, 1] \times [-1, 0]$, and $[0, 1] \times [0, 1]$. Each quarter was associated with a different label. For each sample size we tested, we ran the algorithm three times, each run used a different randomly generated training set. We compared the standard quadratic optimization routine available from Matlab with our algorithm which was also implemented in Matlab. The average running time results are shown in Fig. 5. Note that we used a log-scale for the y (run-time) axis. The results show that the efficient algorithm can be two orders of magnitude faster than the standard QP package.

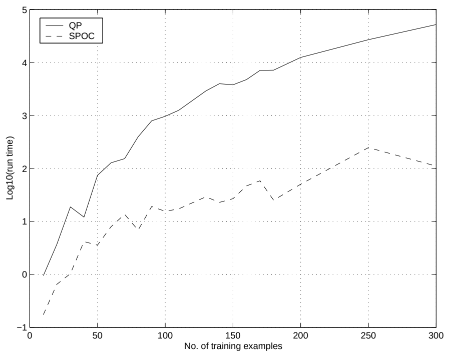

Figure 5: Run time comparison of two algorithms for code design using quadratic programming: Matlab's standard QP package and the proposed algorithm (denoted SPOC). Note that we used a logarithmic scale for the run-time (y) axis.

7 Generalization properties of the algorithm

In this section we will analyze the generalization properties of the multiclass SVM algorithm. We will use another scheme for reducing multiclass problems to multiple binary problems, proposed by Platt [16]. This method also contains two stages. In the training stage the set of all $\binom{k}{2}$ binary classifiers is constructed, where each classifier is trained to distinguish between some pair of distinct labels. In the classification stage the algorithm maintains a list of all possible labels for a given test instance (initialized to the list of all labels). The algorithm runs in steps, in each step it picks two of the labels from that list, and applies the binary classifier which distinguish between the two labels. The label which was rejected by the binary classifier is deleted from the list. After $k - 1$ such steps there is only one possible label is the list, which is the prediction of the multiclass classifier.

It is convenient to represent the classifier using a rooted binary directed acyclic graph (DAG). Each node of the graph represents some list of possible labels, which is elements is a subset of all the labels (we call such list a state). Each node also contains the identity of the binary classifier to be applied on the instance when the state represented by that node is achieved. The leafs of the DAG corresponds to the k singleton states (where there is only one label in the list). From each internal node there are two outgoing edges defined as following. Given a node containing the state $\{r_1, r_2, \ldots, r_u\}$ and the binary classifier that distinguishes between the labels r_1 and r_2. Then the two possible states for the next steps are $\{r_2, \ldots, r_u\}$ (if r_1 was rejected) and $\{r_1, r_3, , \ldots, r_u\}$ (if r_2 was rejected). The root of the DAG is the state corresponding to the list of all the labels.

In the general case there can be up to $2^k - 1$ states. But, it is possible to use only $\binom{k}{2} + k$ states, where $\binom{k}{2}$ of them are internal nodes (one for each possible binary classifier) and the rest k are the singleton leaves. The structure of such classifier is a DAG since a state can be visited using different paths from the root. The binary classifiers used by Platt were Support Vector Machine.

Given a decoding matrix M of size $k \times l$ over $\mathbb{R}$ where each row of M corresponds to a class $y \in \mathcal{Y}$, the set of all binary classifiers can be constructed as following. Assume we want to build a binary classifier to distinguish between class r and class s. Recall that the multi-classifier is given by $H(x) = \arg\max\{x \cdot \bar{M}_r\}$. Similarly, for the binary case we have that

$$\text{The correct label is not r} \Leftrightarrow \bar{M}_r < \bar{M}_s$$

Note that the classifier rules out a label and not points the correct label, because that there is a possibility that the correct label is neither r nor s. Define $w_{r,s}$ to be the (normalized) vector $\bar{M}_r - \bar{M}_s$, then we result the binary SVM $h_{r,s}(x) = \text{sign}(w_{r,s} \cdot x)$, where we interpret a positive output as a prediction of the label r (rejecting the label s).

Define the margin of the binary classifier $h_{r,s}(x)$ to be $\gamma_{r,s} = \min_{(x,y)\in S, y=r,s}\{|w_{r,s} \cdot x|\}$, where y is the label of the example (x, y). We now ready to use Theorem 1 from Platt [16].

Theorem 6 *Suppose we are able to classify a random m sample S of labeled examples using a SVM DAG on k classes containing $K = \binom{k}{2}$ decision nodes (and k leaves) with margin $\gamma_{r,s}$ at node $\{r, s\}$, then we can bound the generalization error with probability greater than $1 - \delta$ to be less than*

$$\frac{130R^2}{m}\left(D' \log(4em) log(4m) + \log\frac{2(2m)^K}{\delta}\right)$$

where $D' = \sum_{\{r,s\}}^{K} \frac{1}{\gamma_{r,s}^2}$, and R is the radius of a ball containing the support of the distribution.

8 Conclusions and future research

In this paper we investigated the problem of *designing* output codes for solving multiclass problems. We first discussed discrete codes and showed that while the problem is intractable in general we can find the first column of a code matrix in polynomial time. The question whether the algorithm can be generalized to $l \geq 2$ columns with running time of $O(2^l)$ or less remains open. Another closely related question is whether we can find efficiently the next column given previous columns. Also left open for future research is further usage of the algorithm for finding the first column as a subroutine in constructing codes based on trees or directed acyclic graphs [16], and as a tool for incremental (column by column) construction of output codes.

Motivated by the intractability results for discrete codes we introduced the notion of continuous output codes. We described three optimization problems for finding good continuous codes for a given a set of binary classifiers. We have discussed in detail an efficient algorithm for one of the three problems which is based on quadratic programming. As a special case, our framework also provides a new efficient algorithm for multiclass Support Vector Machines. The importance of this efficient algorithm might prove to be crucial in large classification problems with many classes such as Kanji character recognition. We also devised efficient implementation of the algorithm. The implementation details of the algorithm, its convergence, generalization properties, and more experimental results were omitted due to the lack of space and will be presented elsewhere. Finally, an important question which we have tackled barely in this paper is the problem of interleaving the code design problem with the learning of binary classifiers. A viable direction in this domain is combining our algorithm for continuous codes with the support vector machine algorithm.

Acknowledgement

We would like to thank Rob Schapire for numerous helpful discussions, to Vladimir Vapnik for his encouragement and support of this line of research, and to Nir Friedman for useful comments and suggestions.

References

[1] D. W. Aha and R. L. Bankert. Cloud classification using error-correcting output codes. In *Artificial Intelligence Applications: Natural Science, Agriculture, and Environmental Science*, volume 11, pages 13–28, 1997.

[2] E.L. Allwein, R.E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. In *Machine Learning: Proceedings of the Seventeenth International Conference*, 2000.

[3] A. Berger. Error-correcting output coding for text classification. In *IJCAI'99: Workshop on machine learning for information filtering*, 1999.

[4] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth & Brooks, 1984.

[5] V. Chvatal. *Linear Programming*. Freeman, 1980.

[6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[7] Ghulum Bakiri Thomas G. Dietterich. Achieving high-accuracy text-to-speech with machine learning. In *Data mining in speech synthesis*, 1999.

[8] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, January 1995.

[9] Tom Dietterich and Eun Bae Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Oregon State University, 1995. Available via the WWW at http://www.cs.orst.edu:80/∼tgd/cv/tr.html.

[10] R. Fletcher. *Practical Methods of Optimization*. John Wiley, second edition, 1987.

[11] Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. *The Annals of Statistics*, 26(1):451–471, 1998.

[12] Klaus-U. Höffgen and Hans-U. Simon. Robust trainability of single neurons. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 428–439, Pittsburgh, Pennsylvania, July 1992.

[13] G. James and T. Hastie. The error coding method and PiCT. *Journal of computational and graphical stastistics*, 7(3):377–387, 1998.

[14] Eun Bae Kong and Thomas G. Dietterich. Error-correcting output coding corrects bias and variance. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 313–321, 1995.

[15] J.C. Platt. Fast training of Support Vector Machines us-

ing sequential minimal optimization. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

[16] J.C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin dags for multiclass classification. In *Advances in Neural Information Processing Systems 12*. MIT Press, 2000. (To appear.).

[17] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[18] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing – Explorations in the Microstructure of Cognition*, chapter 8, pages 318–362. MIT Press, 1986.

[19] Robert E. Schapire. Using output codes to boost multiclass learning problems. In *Machine Learning: Proceedings of the Fourteenth International Conference*, pages 313–321, 1997.

[20] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):1–40, 1999.

[21] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[22] J. Weston and C. Watkins. Support vector machines for multi-class pattern recognition. In *Proceedings of the Seventh European Symposium On Artificial Neural Networks*, April 1999.

A Linear programming

Using Chvatal's [5] notation, given the linear program:

$$\begin{array}{lll} \max\limits_{x} & \sum_{j=1}^{n} c_j x_j & \\ \text{subject to :} & \sum_{j=1}^{n} a_{ij} x_j \le b_i & (i \in I) \\ & \sum_{j=1}^{n} a_{ij} x_j = b_i & (i \in E) \\ & x_j \ge 0 & (j \in R) \text{ (0-Constraints)} \\ & x_j \gtrless 0 & (j \in F) \text{ (Unconstrained variables)} \end{array} ,$$

its dual program is:

$$\begin{array}{lll} \min\limits_{y} & \sum_{i=1}^{m} b_i y_i & \\ \text{subject to :} & \sum_{i=1}^{m} a_{ij} y_i \ge c_j & (j \in R) \\ & \sum_{i=1}^{m} a_{ij} y_i = c_j & (j \in F) \\ & y_i \ge 0 & (j \in I) \text{ (0-Constraints)} \\ & y_i \gtrless 0 & (j \in E) \text{ (Unconstrained variables)} \end{array}$$

B Legend

Var. name	Description	Section
S	Sample	2
M	Matrix code	2
m	Sample size	2
k	No. of classes (No. of rows in M)	2
l	No. of hypotheses (No. of columns in M)	2
i	Index of an example	2
r	Index of a class	2
y	Correct label (class)	2
t	Index of an hypothesis	2
ξ	Slack variables in optimization problem	5
η	Dual variables in quadric problem	5.2
τ	$\tau_{i,r} = \delta_{y_i,r} - \eta_{i,r}$	5.2
A_p, A	Coefficient in reduced optimization prob.	6
B_p, B	Coefficient in reduced optimization prob.	6
C_p	Coefficient in reduced optimization prob.	6
$\bar{D}$	$\bar{\tau} + \frac{\bar{B}}{A}$	6
$\bar{\nu}$	$\frac{\bar{B}}{A} + \bar{1}_y$	6