Conditional Gradients for the Approximately Vanishing Ideal

Elias Wirth
TU Berlin

Sebastian Pokutta
TU Berlin

Abstract

The vanishing ideal of a set of points $X \subseteq \mathbb{R}^n$ is the set of polynomials that evaluate to 0 over all points $x \in X$ and admits an efficient representation by a finite set of polynomials called generators. To accommodate the noise in the data set, we introduce the Conditional Gradients Approximately Vanishing Ideal algorithm (CGAVI) for the construction of the set of generators of the approximately vanishing ideal. The constructed set of generators captures polynomial structures in data and gives rise to a feature map that can, for example, be used in combination with a linear classifier for supervised learning. In CGAVI, we construct the set of generators by solving specific instances of (constrained) convex optimization problems with the Pairwise Frank-Wolfe algorithm (PFW). Among other things, the constructed generators inherit the LASSO generalization bound and not only vanish on the training but also on out-sample data. Moreover, CGAVI admits a compact representation of the approximately vanishing ideal by constructing few generators with sparse coefficient vectors.

1 Introduction

The accuracy of classification algorithms relies on the quality of the available features. Naturally, feature transformations are an important area of research in the field of machine learning. In this paper, we focus on feature transformations for a subsequently applied linear kernel Support Vector Machine (SVM) (Suykens and Vandewalle, 1999), an algorithm that achieves high accuracy only if the features are such that the different classes are linearly separable.

Our approach is based on the idea that a set $X = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^n$ can be succinctly described by the set of algebraic equations satisfied by each point $x \in X$. Put differently, we seek polynomials $g_1, \ldots, g_k \in \mathcal{P}$ such that $g_i(x) = 0$ for all $x \in X$ and $i \in \{1, \ldots, k\}$, where $\mathcal{P}$ is the polynomial ring in $n$ variables. An obvious candidate for a succinct description of $X$ is the vanishing ideal\(^1\) of $X$,

$$\mathcal{I}_X = \{f \in \mathcal{P} \mid f(x) = 0 \text{ for all } x \in X\}.$$  

Even though $\mathcal{I}_X$ contains an infinite number of vanishing polynomials, by Hilbert’s basis theorem (Cox et al., 2013), there exists a finite number of generators of $\mathcal{I}_X$, $g_1, \ldots, g_k \in \mathcal{I}_X$ with $k \in \mathbb{N}$, such that for any $f \in \mathcal{I}_X$, there exist $h_1, \ldots, h_k \in \mathcal{P}$ such that

$$f = \sum_{i=1}^{k} g_i h_i.$$  

The generators have all the points in $X$ as a common root, implying that the set of generators captures the nonlinear structure of the data set $X$ and represents the vanishing ideal $\mathcal{I}_X$. The construction of this set of generators has received a lot of attention (Heldt et al., 2009; Livni et al., 2013; Limbeck, 2013; Iraji and Chitsaz, 2017).

The set of generators can be used to transform the features of $X$ such that linear separability is achieved in the transformed feature space. For example, consider the binary classification task of deciding whether a tumor is cancerous or benign and suppose that the sets $X_c \subseteq X$ and $X_b \subseteq X$ correspond to data points of cancerous and benign tumors, respectively. Then, generators $g_1, \ldots, g_k$ of $\mathcal{I}_{X_c}$ provide a succinct characterization of elements in the class of cancerous tumors and vanish over $X_c$. For points $x \in X_b$, however, we expect that for some $i \in \{1, \ldots, k\}$, the polynomial $g_i$ does not vanish over $x$, that is, $g_i(x) \neq 0$. Similarly, we construct a second set of generators of $\mathcal{I}_{X_b}$, representing the benign tumors. Then, evaluating both

\(^1\)A set of polynomials $\mathcal{I} \subseteq \mathcal{P}$ is called an ideal if it is a subgroup with respect to addition and for $f \in \mathcal{I}$ and $g \in \mathcal{P}$, we have $f \cdot g \in \mathcal{I}$.
sets of generators over the entire data set \(X = X_c \cup X_h\), represents the data set \(X\) mapped into a new feature space in which the two classes are (hopefully) linearly separable. We further develop this idea in Section 7.

To accommodate the noise in the data set, we construct generators \(g\) of the *approximately vanishing ideal* instead of the vanishing ideal, where the approximately vanishing ideal contains polynomials that almost vanish over \(X\), that is, polynomials \(g\) such that \(g(\mathbf{x}) \approx 0\) instead of \(g(\mathbf{x}) = 0\) for all \(\mathbf{x} \in X\).

In this paper, we introduce the Conditional Gradients Approximately Vanishing Ideal algorithm (CGAVI), which takes as input a set of points \(X \subseteq \mathbb{R}^n\) and constructs a set of generators of the approximately vanishing ideal over \(X\). The novelty of our approach is that CGAVI constructs the generators of the approximately vanishing ideal using an oracle (which solves a (constrained) convex optimization problem), in our case, via the *Pairwise Frank-Wolfe algorithm* (PFW) ([Guélat and Marcotte, 1986; Lacoste-Julien and Jaggi, 2015]), whereas related methods such as the *Approximate Vanishing Ideal algorithm* (AVI) ([Heldt et al., 2009] and *Vanishing Component Analysis* (VCA) ([Livni et al., 2013]) rely on *Singular Value Decompositions* (SVD) to construct generators. Our approach admits the following properties:

1. **Generalization bounds:** To the best of our knowledge, CGAVI is the first algorithm that constructs generators that provably vanish approximately on out-sample data.
2. **Compact transformation:** CGAVI constructs a small number of generators with sparse coefficient vectors.
3. **Blueprint:** For the implementation of the oracle in CGAVI, PFW can be replaced by any other solver of (constrained) convex optimization problems. Thus, our paper gives rise to an entire family of procedures for the construction of generators of the approximately vanishing ideal.
4. **Empirical results:** For the combined approach of constructing generators to transform features for a linear kernel SVM, generators constructed with CGAVI lead to test set classification errors and evaluation times comparable to or better than with related methods such as AVI and VCA.

2 Related works

2.1 Approximately vanishing ideal

The first algorithm for constructing generators of the vanishing ideal was the *Buchberger-Möller algorithm* ([Möller and Buchberger, 1982]). Its high susceptibility to noise was first addressed with the introduction of AVI in Heldt et al. (2009), see also Fassino (2010); Limbeck (2013). AVI is similar to CGAVI in that both require a term ordering and construct generators as linear combinations of monomials. The term ordering requirement is the reason why AVI and CGAVI can produce different outputs depending on the order of the features of the data set, an undesirable property in practice. The currently most prevalent, and contrary to AVI and CGAVI, polynomial-based approach for constructing generators of the approximately vanishing ideal is VCA, introduced by Livni et al. (2013) and improved by Zhang (2018). VCA constructs generators not as linear combinations of monomials but of polynomials; that is, VCA constructed generators are, in some sense, polynomials whose terms are other polynomials. Like most polynomial-based approaches, VCA does not require an ordering of terms and the algorithm has been exploited in hand posture recognition, principal variety analysis for nonlinear data modeling, solution selection using genetic programming, and independent signal estimation for blind source separation tasks (Zhao and Song, 2014; Iraji and Chitsaz, 2017; Kera and Iba, 2016; Wang and Ohtsuki, 2018). Variants of VCA have also been studied in connection with kernels ([Király et al., 2014; Hou et al., 2016]). Despite VCA’s prevalence, foregoing the term ordering of monomial-based approaches also gives rise to major disadvantages. VCA constructs more generators than AVI or CGAVI, VCA’s generators are non-sparse in their polynomial representation, and VCA is highly susceptible to the *spurious vanishing problem* (Kera and Iba, 2016; Kera and Hasegawa, 2019, 2020): Polynomials with small coefficient vector entries that vanish over \(X\) still get added to the set of generators even though they do not hold any structurally useful information of the data, and, conversely, polynomials that describe the data well get rejected as non-vanishing due to the size of their (large) coefficient vector entries.

2.2 Conditional Gradients

The core difference between CGAVI, AVI, and VCA is the replacement of the SVD steps in the latter two methods with PFW calls to construct generators. PFW is a variant of the *Frank-Wolfe* ([Frank and Wolfe, 1956]) or *Conditional Gradients* (Levitin and Polyak, 1966) algorithm. Conditional Gradients methods are a family of algorithms which appear as building blocks in a variety of scenarios in machine learning, e.g., structured prediction ([Jaggi and Surovikov, 2010; Giesen et al., 2012; Harchaoui et al., 2012; Freund et al., 2017; Garber et al., 2018], optimal transport ([Courty et al., 2016; Paty and Cuturi, 2019; Luise et al., 2019]), and video co-localization ([Joulin et al., 2014; Bojanowski et al., 2017]).
Recall that we address noise in the data set by constructing approximately vanishing polynomials, which we define via the mean squared error.

**Definition 3.1** (Approximately vanishing polynomial). Let \( X = \{x_1, \ldots, x_m\} \subseteq \mathbb{R}^n \) and \( \psi \geq 0 \). A polynomial \( f \in \mathcal{P} \) is \( \psi \)-approximately vanishing (over \( X \)) if \( \text{MSE}(f, X) \leq \psi \).

Recall the spurious vanishing problem. For \( X = \{x_1, \ldots, x_m\} \subseteq \mathbb{R}^n \), any polynomial \( f \) with \( \text{MSE}(f, X) > \psi \) can be re-scaled to become \( \psi \)-approximately vanishing, regardless of its roots, by multiplying all coefficient vector entries of \( f \) with \( \sqrt{\frac{\psi}{\text{MSE}(f, X)}} \). To address this issue, we require an ordering of terms, in our case, the degree-lexicographical ordering of terms (DegLex) (Cox et al., 2013), denoted by \( <_\sigma \). For example, \( x <_\sigma y <_\sigma x^2 <_\sigma xy <_\sigma y^2 \ldots \). Throughout, for \( \mathcal{O} = \{t_1, \ldots, t_k\} \subseteq \mathcal{T} \), the subscript \( \sigma \) indicates that \( t_1 <_\sigma \cdots <_\sigma t_k \). DegLex allows us to define the leading term (coefficient) of a polynomial.

**Definition 3.2** (Leading term (coefficient)). Let \( f = \sum_{i=1}^k c_i t_i \subseteq \mathcal{P} \). Define the leading term and leading term coefficient of \( f \) as \( \text{LT}(f) := \{t_i \mid t_i >_\sigma t_j \forall j \neq i\} \) and \( \text{LTC}(f) := \{c_i \mid t_i >_\sigma t_j \forall j \neq i\} \), respectively.

For \( X = \{x_1, \ldots, x_m\} \subseteq \mathbb{R}^n \), a polynomial \( f \) with \( \text{LTC}(f) = 1 \) that vanishes \( \psi \)-approximately over \( X \) is called \( (\psi, 1) \)-approximately vanishing (over \( X \)). Fixing the LTC of generators prevents rescaling, thus addressing the spurious vanishing problem. We now formally define the approximately vanishing ideal.

**Definition 3.3** (Approximately vanishing ideal). Let \( X = \{x_1, \ldots, x_m\} \subseteq \mathbb{R}^n \) and \( \psi \geq 0 \). The \( \psi \)-approximately vanishing ideal (over \( X \)), \( \mathcal{I}_X^\psi \), is the ideal generated by all \( (\psi, 1) \)-approximately vanishing polynomials over \( X \).

Note that the definition above subsumes the definition of the vanishing ideal, i.e., for \( \psi = 0 \), \( \mathcal{I}_X^0 = \mathcal{I}_X \) is the vanishing ideal. In that case, we write \( \mathcal{I}_X \) instead of \( \mathcal{I}_X^0 \), omitting the superscript. In this paper, we introduce an algorithm addressing the following problem.

**Problem 3.4** (Setting). Let \( X = \{x_1, \ldots, x_m\} \subseteq \mathbb{R}^n \) and \( \psi \geq 0 \). Construct a set of generators of \( \mathcal{I}_X^\psi \).

### 4 Convex optimization

To solve Problem 3.4, we first address the subproblem of certifying (non-)existence of and constructing generators of the \( \psi \)-approximately vanishing ideal, \( \mathcal{I}_X^\psi \), under the restriction that terms of generators are contained in a specific set of terms. As we show in this section, this subtask can be reduced to solving a convex optimization problem.
Let $X = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^n$, $O = \{t_1, \ldots, t_k\}_{\sigma} \subseteq T$, $t \in T$ with $t >_{\sigma} t_i$ for all $i \in [k]$, and $\varepsilon \geq 0$.

**Output:** A polynomial $g \in \mathcal{P}$ with $\text{LT}(g) = t$, $\text{LTC}(g) = 1$, other terms only in $O$, and $\text{MSE}(g, X) \leq \min_{\nu \in \mathcal{R}^{|O|}} \ell(O(X), t(X))(\nu) + \varepsilon$.

Suppose there exists a $(\psi, 1)$-approximately vanishing polynomial $f = \sum_{i=1}^{k} c_i t_i + t$ with $\text{LT}(f) = t$ and other terms only in $O$. Let $c = (c_1, \ldots, c_k)^T$. Then,

$$\psi \geq \text{MSE}(f, X) = \ell(O(X), t(X))(c) \geq \min_{\nu \in \mathbb{R}^k} \ell(O(X), t(X))(\nu),$$

where for $A \in \mathbb{R}^{m \times n}$, $y \in \mathbb{R}^m$, and $x \in \mathbb{R}^n$,

$$\ell(A, y)(x) := \frac{1}{m} \|Ax + y\|_2^2$$

is the squared loss. The right-hand side of (1) is a convex optimization problem and in the argmin-version has the form

$$d = \arg\min_{\nu \in \mathbb{R}^k} \ell(O(X), t(X))(\nu).$$

With $d$, we now construct the polynomial $g = \sum_{i=1}^{k} d_i t_i + t$. Note that $\text{LT}(g) = t$, $\text{LTC}(g) = 1$, and

$$\min_{\nu \in \mathbb{R}^k} \ell(O(X), t(X))(\nu) = \frac{1}{m} \|O(X)d + t(X)\|_2^2 = \text{MSE}(g, X).$$

Thus, $g$ also vanishes $(\psi, 1)$-approximately over $X$, its leading term is $t$, and all of its other terms are in $O$, as required. We obtain the following result.

**Theorem 4.1 (Certificate).** Let $X = \{x_1, \ldots, x_m\} \subseteq \mathbb{R}^n$, $\psi \geq 0$, $O = \{t_1, \ldots, t_k\}_{\sigma} \subseteq T$, $t \in T$ such that $t >_{\sigma} t_i$ for all $i \in [k]$, and $d$ as in (COP). There exists a $(\psi, 1)$-approximately vanishing polynomial $f$ with $\text{LT}(f) = t$ and other terms only in $O$, if and only if $g = \sum_{i=1}^{k} d_i t_i + t$ vanishes $(\psi, 1)$-approximately.

The discussion above not only constitutes a proof of Theorem 4.1 but also gives rise to an algorithmic blueprint, ORACLE, presented in Algorithm 1. We denote the output of running ORACLE with $X, O, t, \varepsilon$ and $g = \text{ORACLE}(X, O, t, \varepsilon)$. In practice, any $\varepsilon$-accurate solver of (COP), e.g., gradient descent, can be used to implement ORACLE in two steps: First, solve problem (COP) to $\varepsilon$-accuracy with the solver yielding a vector $d \in \mathbb{R}^{|O|}$. Second, construct and return $g = \sum_{i=1}^{k} d_i t_i + t$. We then say that the $\varepsilon$-accurate solver is used as ORACLE. If, for example, we implement ORACLE with gradient descent, we say that we use gradient descent as ORACLE.

In case that ORACLE is implemented with an accurate solver of (COP), that is, $\varepsilon = 0$, by Theorem 4.1, either $\text{MSE}(g, X) > \psi$ and we have proof that no $(\psi, 1)$-approximately vanishing polynomial with leading term $t$ and other terms only in $O$ exists, or $g$ vanishes $(\psi, 1)$-approximately with leading term $t$ and other terms only in $O$.

### 5 The OAVI algorithm

In this section, we give a detailed description of the Oracle Approximately Vanishing Ideal algorithm (OAVI), which we refer to as CGAVI when the ORACLE used is a Conditional Gradients algorithm. OAVI is presented in Algorithm 2 and we analyze its properties in Section 6.

#### Algorithm 1 ORACLE

**Input:** $X = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^n$, $O = \{t_1, \ldots, t_k\}_{\sigma} \subseteq T$, $t \in T$ with $t >_{\sigma} t_i$ for all $i \in [k]$, and $\varepsilon \geq 0$.

**Output:** $G \subseteq \mathcal{P}$ and $O \subseteq T$.

1. $d \leftarrow 1$
2. $O = \{t_1\)_\sigma \leftarrow \{1\}_\sigma$
3. $G \leftarrow \emptyset$
4. **while** $\partial_d O = \{u_1, \ldots, u_k\}_\sigma \neq \emptyset$ **do**
5. **for** $i = 1, \ldots, k$ **do**
6. $g \leftarrow \text{ORACLE}(X, O, u_i, \varepsilon) \in \mathcal{P}$
7. **if** $\text{MSE}(g, X) \leq \psi$ **then**
8. $G \leftarrow G \cup \{g\}$
9. **else**
10. $O \leftarrow (O \cup \{u_i\})_\sigma$
11. **end if**
12. **end for**
13. $d \leftarrow d + 1$
14. **end while**

#### Algorithm 2 OAVI

**Input:** $X = \{x_1, \ldots, x_m\} \subseteq \mathbb{R}^n$ and $\psi \geq \varepsilon \geq 0$.

**Output:** $G \subseteq \mathcal{P}$ and $O \subseteq T$.

1. $d \leftarrow 1$
2. $O = \{t_1\}_\sigma \leftarrow \{1\}_\sigma$
3. $G \leftarrow \emptyset$
4. **while** $\partial_d O = \{u_1, \ldots, u_k\}_\sigma \neq \emptyset$ **do**
5. **for** $i = 1, \ldots, k$ **do**
6. $g \leftarrow \text{ORACLE}(X, O, u_i, \varepsilon) \in \mathcal{P}$
7. **if** $\text{MSE}(g, X) \leq \psi$ **then**
8. $G \leftarrow G \cup \{g\}$
9. **else**
10. $O \leftarrow (O \cup \{u_i\})_\sigma$
11. **end if**
12. **end for**
13. $d \leftarrow d + 1$
14. **end while**
ψ = ε = 0, that is, we focus on the construction of generators of the vanishing ideal I_X assuming access to an accurate solver of (COP).

**Initialization:** We keep track of two sets: \( \mathcal{O} \subseteq \mathcal{T} \) for the set of terms such that no generator exists with terms only in \( \mathcal{O} \) and \( \mathcal{G} \subseteq \mathcal{P} \) the set of generators. Since the constant 1 polynomial does not vanish, we initialize \( \mathcal{O} = \{ t_1 \} \) and \( \mathcal{G} \leftarrow \emptyset \).

**Line 4:** After constructing degree \( d - 1 \) generators, in a next step, given \( \mathcal{O}_{\leq d-1} \) and \( \mathcal{G}_{\leq d-1} \), **OAVI** constructs all vanishing polynomials of degree \( d \). Checking for all \( \binom{n}{d} \) monomials of degree \( d \) whether one of them is the leading term of a generator is impractical. Instead, note that there exists a set of generators \( \mathcal{G} \) of \( I_X \) such that none of the terms of generators in \( \mathcal{G} \) are divisible by the leading terms of other generators in \( \mathcal{G} \).

**Lemma 5.1.** Let \( X = \{ x_1, \ldots, x_n \} \subseteq \mathbb{R}^n \). There exists a set of generators \( \mathcal{G} \subseteq \mathcal{P} \) of \( I_X \) such that for \( g, h \in \mathcal{G}, g \neq h, h = \sum_{i=1}^{k} c_i t_i \in \mathcal{G}, \text{LT}(g) \nmid t_i \) for any \( i \in [k] \).

To construct degree \( d \) generators, we thus only consider terms \( t \in \mathcal{T}_d \) such that for all \( g \in \mathcal{G}_{\leq d-1} \), \( \text{LT}(g) \nmid t \). This is equivalent to requiring that all divisors of degree \( \leq d - 1 \) of \( t \) are in \( \mathcal{O}_{\leq d-1} \).

**Definition 5.2** (Border). Let \( \mathcal{O} \subseteq \mathcal{T}_{d-1} \). The (degree \( d \)) border of \( \mathcal{O} \) is defined as \( \partial_d \mathcal{O} := \{ t \in \mathcal{T}_d \mid t \in \mathcal{O}_{\leq d-1} \} \) for all \( t \in \mathcal{T}_{d-1} \) such that \( t \nmid u \).

In other words, we only consider degree \( d \) terms that are contained in the border, drastically reducing the number of redundant generators constructed.

**While loop:** Suppose that \( \partial_d \mathcal{O} \neq \emptyset \), else **OAVI** terminates. For each term \( u \in \partial_d \mathcal{O} \), starting with the smallest with respect to \( <_\sigma \), we construct a polynomial \( g \) via a call to **ORACLE**. By Theorem 4.1, there exists a vanishing polynomial \( f \) with \( \text{LTC}(f) = 1 \), \( \text{LT}(f) = u \), and other terms only in \( \mathcal{O} \), if and only if \( g \) is a vanishing polynomial. If \( g \) vanishes, we append \( g \) to \( \mathcal{G} \). If \( g \) does not vanish, we append \( \text{LT}(g) = u \) to \( \mathcal{O} \).

**Termination:** For bounded data \( X \subseteq [-1, 1]^n \), the border is guaranteed to become empty and the algorithm terminates, see Theorem 6.1.

**Output:** Throughout, we denote the output of **OAVI** by \( \mathcal{G} \) and \( \mathcal{O} \) that is, \( (\mathcal{G}, \mathcal{O}) = \text{OAVI}(X, \psi, \varepsilon) \).

---

**Remark 5.3** (**AVI** and **VCA**). We use the notation from this paper to refer to the sets corresponding to \( \mathcal{O}, \mathcal{G} \), and \( \partial_d \mathcal{O} \) in **AVI** and **VCA** even though notation in Heldt et al. (2009) and Livni et al. (2013), respectively, may differ. For **AVI**, \( \mathcal{O}, \mathcal{G} \), and \( \partial_d \mathcal{O} \) are identical to the sets used in **OAVI**. **AVI** does, however, employ an SVD step to determine all generators with leading terms in the border at once. **VCA** employs a similar SVD step as **AVI** for generator construction. The main difference between **OAVI**, **AVI**, and **VCA** is that the latter is polynomial-based. Indeed, the set \( \mathcal{O} \) in **VCA** does not contain monomials but polynomials that provably do not vanish approximately over the data. Thus, **VCA** does not require a term ordering. The disadvantage of avoiding the term ordering is that the border in **VCA** cannot be defined as in Definition 5.2 but is instead defined as \( \partial_d \mathcal{O} := \{ t \mid t \in \mathcal{O}, 1 \leq d \} \), which can lead to the construction of unnecessary generators. For example, suppose that \( \mathcal{O}_1 = \{ t_1, \ldots, t_n \} \) and \( \mathcal{O}_2 = \{ t_1^2 \} \). By construction, \( t_1 t_2 \) and \( t_1 t_3 \) are leading terms of approximately vanishing polynomials for all \( i, j \in \{ 2, \ldots, n \} \) and, by Lemma 5.1, there is no need to consider generators with leading terms \( t_1^2 t_2 \), \( t_1^2 t_3 \), \( t_1^2 t_4 \), \( t_1^2 t_5 \), \( t_1^2 t_6 \), \( t_1^2 t_7 \), \( t_1^2 t_8 \), \( t_1^2 t_9 \), and \( t_1^2 t_{10} \).

**6 Analysis**

We begin the analysis of **OAVI** by proving that the algorithm terminates.

**Theorem 6.1** (Termination). Let \( X = \{ x_1, \ldots, x_m \} \subseteq [-1, 1]^m \), \( \psi \geq \varepsilon \geq 0 \), and \( (\mathcal{G}, \mathcal{O}) = \text{OAVI}(X, \psi, \varepsilon) \). Then, **OAVI** terminates.

Thus, for \( \varepsilon = 0 \), **OAVI** constructs a set of generators of the \( \psi \)-approximately vanishing ideal in finite time. Since, in practice, we employ solvers that are \( \varepsilon \)-accurate only for \( \varepsilon > 0 \), in the theorem below, we prove that when \( \varepsilon > 0 \), **OAVI** addresses Problem 3.4 up to tolerance \( \varepsilon \).

**Theorem 6.2** (Maximality). Let \( X = \{ x_1, \ldots, x_m \} \subseteq [-1, 1]^m \), \( \psi \geq \varepsilon \geq 0 \), and \( (\mathcal{G}, \mathcal{O}) = \text{OAVI}(X, \psi, \varepsilon) \). Then, all \( g \in \mathcal{G} \neq \emptyset \) are \( (\psi, 1) \)-approximately vanishing polynomials and there does not exist a \( (\psi - \varepsilon, 1) \)-approximately vanishing polynomial with terms only in \( \mathcal{O} \).

Thus, **OAVI** is guaranteed to construct all \( (\psi - \varepsilon, 1) \)-approximately vanishing polynomials but generators that vanish \( (\lambda, 1) \)-approximately, where \( \lambda \in [\psi - \varepsilon, \psi] \), may not be detected by **OAVI**.
6.1 Computational complexity

For bounded data $X \subseteq [-1, 1]^n$, OAVI returns bounded $|G|$ and $|O|$, a useful property for discussing time and space complexity.

**Proposition 6.3.** Let $X = \{x_1, \ldots, x_m\} \subseteq [-1, 1]^n$, $\psi \geq \varepsilon \geq 0$, and $(G, O) = \text{OAVI}(X, \psi, \varepsilon)$. Then, $|O| \leq m$ and $|G| \leq |O|n \leq mn$.

Proposition 6.3 is an essential tool for the analysis of the computational complexity of OAVI.

**Theorem 6.4** (Computational Complexity). Let $X = \{x_1, \ldots, x_m\} \subseteq [-1, 1]^n$, $\psi \geq \varepsilon \geq 0$, and $(G, O) = \text{OAVI}(X, \psi, \varepsilon)$. In the real number model, the time and space complexities of OAVI are $O(m^2n^2 + mnT_{\text{ORACLE}})$ and $O(m^2n + S_{\text{ORACLE}})$, where $T_{\text{ORACLE}}$ and $S_{\text{ORACLE}}$ are the time and space complexities of ORACLE, respectively.

As we prove in the adaptation of Theorem 5.1 in Livni et al. (2013) to OAVI below, the evaluation complexity of OAVI’s output is identical to that of related methods such as AVI and VCA.

**Theorem 6.5** (Evaluation Complexity). Let $X = \{x_1, \ldots, x_m\} \subseteq [-1, 1]^n$, $\psi \geq \varepsilon \geq 0$, and $(G, O) = \text{OAVI}(X, \psi, \varepsilon)$. In the real number model, we can compute the evaluation vectors of all monomials in $O$ and polynomials in $G$ over a set $Z = \{x_1, \ldots, x_q\} \subseteq [-1, 1]^n$ in time $O(|O|q)$ and $O(|G||O|q)$, respectively.

To reduce the time required to evaluate the feature transformation on new data $Z$, one can reduce the number of terms in $O$, construct fewer polynomials in $G$, increase the sparsity of polynomials in $G$, or reduce the number of samples in the set $Z$.

6.2 Generalization bounds

So far, the construction of generators for OAVI with ORACLE involves solving an unconstrained convex optimization problem. In this section, we replace (COP) with a constrained convex optimization problem and show that this replacement allows OAVI to create generators that approximately vanish not only over in-sample but also over out-sample data.

For $\tau \geq 2$, a polynomial $f = \sum_{i=1}^k c_it_i \in P$ with $c = (c_1, \ldots, c_k)^T$ is said to be $\tau$-bounded in norm $\|\cdot\|$ if the norm of its coefficient vector is bounded by $\tau$, that is, if $\|f\| := \|c\| \leq \tau$. Replacing (COP) by

$$d \in \text{argmin}_{v \in \mathbb{R}^k, \|v\|_1 \leq \tau-1} \ell((O(X), t(X))(v)$$

(allows OAVI to create $\tau$-bounded generators in $\|\cdot\|$. When $\|\cdot\| = \|\cdot\|_1$, (CCOP) is the bounded LASSO, for which generalization bounds are available, see, e.g., Mohri et al. (2018). Thus, $\tau$-bounded generators constructed by OAVI also vanish on out-sample data.

**Theorem 6.6** (Vanishing property). Let $X \subseteq [-1, 1]^n$, $Y \subseteq [-1, 1]$, $D$ be a distribution over $X \times Y$, $S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \subseteq (X \times Y)^m$ sampled according to $D$, and $X = \{x_1, \ldots, x_m\}$. Let $\psi \geq \varepsilon \geq 0$, $\tau \geq 2$, and $(G, O) = \text{OAVI}(X, \psi, \varepsilon)$. Suppose that for all $g \in G$, we have $\|g\|_1 \leq \tau$. Then, for any $g \in G$ and $\delta > 0$, with probability at least $1 - \delta$, we have

$$\mathbb{E}_{(x,y) \sim D} [\text{MSE}(g, \{x\})] \leq \text{MSE}(g, X) + 2\tau^2 \sqrt{\frac{2\log(2m)}{m}} + \tau^2 \sqrt{\frac{\log(\delta^{-1})}{2m}}.$$

Generators created by AVI and VCA are not guaranteed to be $\tau$-bounded and, thus, in general, do not satisfy Theorem 6.6. On the other hand, generators constructed by OAVI with an ORACLE solving (CCOP) instead of (COP) are guaranteed to be $\tau$-bounded in $\|\cdot\|_1$ due to the equivalence of norms in finite dimensions and, thus, vanish also on out-sample data.

We extend our analysis of generalization bounds in Appendix B. Specifically, in Theorem B.3, we present sufficient conditions under which feature transformation algorithms in combination with a linear kernel SVM inherit the margin bound of the SVM.

6.3 On approximately vanishing

In Definition 3.1, the mean squared error is used to define approximately vanishing polynomials but other options are possible. In AVI, for example, a polynomial $f \in P$ vanishes $\psi$-approximately over $X \subseteq [-1, 1]^n$ if $\|f(X)\|_2 \leq \psi$. However, definitions of vanishing approximately that are less similar to ours are imaginable. We could, for example, define vanishing $\psi$-approximately via the maximum loss instead, that is, $f$ would be defined as vanishing $\psi$-approximately over $X$ if $|f(x)| \leq \psi$ for all $x \in X$. Minimizing the maximum loss instead of the squared loss in (COP) results in a convex-concave optimization problem that has to be addressed with specific solvers, for example via the method described in Shalev-Shwartz and Wexler (2016).

In this paper, we limit ourselves to adding an $\ell_2$-regularization term $\frac{\lambda}{2}\|x\|_2^2$ to (SL). Then, a polynomial $f = \sum_{i=1}^k c_it_i + t \in P$ with $\text{LT}(f) = t$ vanishes $(\psi, 1)$-approximately over $X$ if

$$\text{MSE}(f, X) + \frac{\lambda}{2}\|(c_1, \ldots, c_k)^T\|_2^2 \leq \psi.$$

Setting $\lambda > 0$ can prolong the running time of OAVI or make termination impossible altogether. Furthermore, $|G|$ and $|O|$ are no longer guaranteed to be bounded. In our practical experiments, however, these negative properties do not seem to be present, see Section 9.
7 Pipeline

Following the notation of Mohri et al. (2018), we present a detailed overview of the machine learning pipeline for classification problems using generators of the vanishing ideal to transform features for a linear kernel SVM.

Consider an input space $X \subseteq [-1, 1]^n$ and an output or target space $Y = [k]$. We receive a training sample $S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \in (X \times Y)^m$ drawn i.i.d. from some unknown distribution $D$. The problem is to determine a hypothesis $h: X \rightarrow Y$ with small generalization error

$$P_{(x,y) \sim D}[h(x) \neq y].$$

Let $X = \{x_1, \ldots, x_m\}$. For each class $i \in [k]$, let $X^i \subseteq X$ denote the set of feature vectors corresponding to class $i$ and construct a set of generators $G^i = \{g^{(i)}_j\}_{j=1}^{|G^i|}$ for the vanishing ideal $I_{X^i}$. As proposed in Livni et al. (2013), we can then transform samples $x \in X$ via the feature transformation

$$x \mapsto \tilde{x} = \left(\ldots, |g^{(i)}_1(x)|, \ldots, |g^{(i)}_{|G^i|}(x)|, \ldots\right)^\top,$$  

where $g^{(i)}_j$ is the $j$-th vanishing polynomial corresponding to class $i$. In other words, $g^{(i)}_j$ vanishes over all $x \in X^i$ and (hopefully) attains non-zero values over points $x \in X \setminus X^i$. We then train a linear kernel SVM on the feature transformed data $\tilde{S} = \{(\tilde{x}_1, y_1), \ldots, (\tilde{x}_m, y_m)\}$ with $\ell_1$-regularization to keep the number of used features as small as possible. As shown in Livni et al. (2013), if the underlying classes of $S$ belong to disjoint algebraic sets, they become linearly separable in the feature space corresponding to transformation (FT). If we restrict the number and norm of the coefficient vectors of generators used to transform the features, e.g., by bounding the maximum degree of generators and, in the case of OAVI, replacing (COP) by (CCOP), this combination of feature transformation and SVM satisfies Theorem B.3, guaranteeing that test set accuracy improves as the number of training samples increases.

8 Putting the CG in CGAVI

Any algorithm guaranteeing an $\varepsilon$-accurate solution to (COP) or (CCOP) can be used to construct the polynomial $g$ returned by ORACLE, allowing the practitioner to choose a solver with specific properties for any given task. Our goal is to construct a set of generators $G$ consisting of few and sparse polynomials to obtain a compact representation of the approximately vanishing ideal. The former property is already achieved by restricting leading terms of generators to be in the border, see Section 5. The latter property, we address with our choice of solver for (COP) or (CCOP).

To do so, we formalize the notion of sparsity. Consider the execution of OAVI (or AVI) and suppose that, currently, $O = \{t_1, \ldots, t_k\}$ and $g = \sum_{i=1}^k c_i t_i + t$ with $\text{LT}(g) = t \not\in O$ gets appended to $G$. Let the number of entries, the number of zero entries, and the number of non-zero entries in the coefficient vector of $g$ be denoted by $g_e := k$, $g_z := |\{c_i = 0 \mid i \in [k]\}|$, and $g_n := g_e - g_z$, respectively. We then define the sparsity of $g$ as \(\text{SPAR}(g) := g_z / g_e \in [0, 1]\). Larger \(\text{SPAR}(g)\) indicates a more thinly populated coefficient vector of $g$. Recall that for classification as in Section 7 we construct sets of vanishing polynomials $G^i$ corresponding to classes $i$ and then transform the samples via (FT) using all polynomials in $G := \bigcup_i G^i$. To measure sparsity of the feature transformation, define the sparsity of $G$ as

$$\text{SPAR}(G) := \frac{\sum_{g \in G} g_z}{\sum_{g \in G} g_e} \in [0, 1],$$

which is the weighted average of the sparsity of all polynomials in $G$ with respect to $g_e$.

In this paper, we focus on the implementation of ORACLE with the Pairwise Frank-Wolfe algorithm (PFW), presented in Algorithm 3 in Appendix C. The algorithm converges linearly when optimizing the squared loss over the $\ell_1$-ball using line search for its step-size rule and the iterate returned by PFW tends to be sparse (Lacoste-Julien and Jaggi, 2015). Furthermore, when optimizing over a bounded region, the returned iterate is bounded and the corresponding generator constructed in ORACLE is bounded, which is necessary for the generalization bounds presented in this paper, Theorems 6.6 and B.3, to hold.

9 Numerical experiments

We use different algorithms that construct generators of the approximately vanishing ideal to transform features for a linear kernel SVM and compare them to each other and a polynomial kernel SVM.

Set-up. We apply different feature transformation algorithms for a linear kernel SVM using the set-up from Section 7 and a polynomial kernel SVM to classification problems. Let $G := \bigcup_i G^i$, where $G^i$ is the set of generators constructed for class $i$ as in Section 8. We compare the number of generators constructed, $|\bigcup_i G^i|$, for VCA, define $g_e$, $g_z$, and $g_n$ with respect to the representation of $g$ as a linear combination of polynomials.

---

3Recall that a set $U \subseteq \mathbb{R}^n$ is algebraic if there exists a finite set of polynomials $U = \{u_1, \ldots, u_k\} \subseteq \mathcal{P}$, such that $U$ is the set of the common roots of $U$.

4For VCA, define $g_e$, $g_z$, and $g_n$ with respect to the representation of $g$ as a linear combination of polynomials.
Table 1: Overview of the data sets used in the numerical experiments. Each row contains the data set used, its full name, the number of samples, the number of features, and the number of classes.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Full name</th>
<th># samples</th>
<th># features</th>
<th># classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>bank</td>
<td>banknote authentication</td>
<td>1372</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>cancer</td>
<td>breast cancer Wisconsin</td>
<td>569</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>htru2</td>
<td>HTRU2 (Lyon et al., 2016)</td>
<td>17898</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>iris</td>
<td>iris</td>
<td>150</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>seeds</td>
<td>seeds</td>
<td>210</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>sonar</td>
<td>connectionist bench (sonar, mines vs. rocks)</td>
<td>208</td>
<td>60</td>
<td>2</td>
</tr>
<tr>
<td>spam</td>
<td>spambase</td>
<td>14601</td>
<td>57</td>
<td>2</td>
</tr>
<tr>
<td>voice</td>
<td>LSVT voice rehabilitation (Tsanas et al., 2013)</td>
<td>126</td>
<td>310</td>
<td>2</td>
</tr>
<tr>
<td>wine</td>
<td>wine</td>
<td>178</td>
<td>13</td>
<td>3</td>
</tr>
</tbody>
</table>

the sparsity of the feature transformation, (SPAR), the number of non-zero entries in the coefficient vectors of generators in $G$, $\sum_{g \in G} g_n$, the test time in seconds, Test time, and the classification error on the test set in %, Test error. The results are averaged over ten random 60%/40% train/test partitions.

Implementation. All experiments are performed on an NVIDIA TITAN RTX GPU with 24GB RAM and a 64-core AMD Ryzen Threadripper 3990X CPU at 3.30 GHz with 128 GB RAM. Our code is publicly available on GitHub. The algorithms are:

$\ell_1$-CGAVI: We implement OAVI with PFW and $\ell_2$-regularized squared loss as ORACLE for (CCOP) with $\varepsilon = \psi/2$ and radius $\tau = 50^7$ using line search. PFW is terminated when the polynomial corresponding to the constructed iterate is guaranteed to vanish $\psi$-approximately, if the Frank-Wolfe gap is smaller than $\varepsilon$, or if no significant progress is made during an iteration.

$\ell_2$-CGAVI: The implementation is identical to $\ell_1$-CGAVI, except that we replace PFW with vanilla Conditional Gradients and the $\ell_1$- by the $\ell_2$-ball.

AGDAVI: We implement Accelerated Gradient Descent (AGD) (Nesterov, 1983) with $\ell_2$-regularized squared loss as ORACLE in OAVI. AGD is terminated when the polynomial corresponding to the constructed iterate is guaranteed to vanish $\psi$-approximately or if no significant progress is made over 5 iterations.

AVI: We implement AVI as in Heldt et al. (2009) with improved Stabilized Reduced Row Echelon Form algorithm as in Limbeck (2013).

VCA: We implement VCA as in Livni et al. (2013).

SVM: We implement a polynomial kernel multiclass SVM with one-versus-rest approach using the scikit-learn software package (Pedregosa et al., 2011). We run the SVM up to tolerance 1e-3 with $\ell_2$-regularization. For the SVM, $\sum_i |G_i^r|$, (SPAR), and $\sum_{g \in G} g_n$ do not exist.

All algorithms above are terminated after constructing degree 10 polynomials, even if the upcoming border is not empty. After applying the algorithms, except for SVM, we subsequently apply a linear kernel SVM as in Section 7. We implement the linear kernel SVM using the scikit-LEARN software package and run it for up to 1000 iterations, up to tolerance 1e-4, with $\ell_1$-penalized squared hinge loss, and one versus the rest approach.

The abbreviations for the feature transformation algorithms above also refer to the complete approach of feature transformation with subsequently applied linear kernel SVM. Further, OAVI refers to $\ell_1$-CGAVI, $\ell_2$-CGAVI, and AGDAVI.

Data sets. All data sets are retrieved from the UCI Machine Learning Repository (Dua and Graff, 2017). An overview is given in Table 1. For each data set, we apply min-max feature scaling into the range $[0, 1]$ as a preprocessing step.

Hyperparameters. We tune the hyperparameters using three-fold cross-validation. For all approaches except the polynomial kernel SVM, we tune the $\ell_1$-regularization coefficient for the linear kernel SVM. For OAVI, we tune the vanishing parameter $\psi$ and the $\ell_2$-regularization parameter. For AVI, we tune the vanishing parameter $\psi$ and the tolerance $\tau$ controlling the sparsity of the constructed generators. For VCA, we tune the vanishing parameter $\psi$. For the polynomial kernel SVM, we tune the $\ell_2$-regularization parameter and the degree of the polynomial kernel.

Results. $\sum_i |G_i^r|$: As can be seen by Table 2, for all data sets except for bank and cancer, OAVI constructs fewer generators than AVI and VCA. For seeds, sonar, spam, voice, and wine, VCA constructs up to an order of magnitude more generators than OAVI due to employing a different border set as discussed in Remark 5.3. For most data sets, AVI also produces significantly more generators than OAVI but the reason for this is not yet understood.
### Table 2: We compare the number of generators constructed, $\sum_{g \in G} |g|$, the sparsity of the feature transformation, (SPAR), the number of non-zero entries in the coefficient vectors of generators in $G$, $\sum_{g \in G} g_n$, the test time in seconds, Test time, and the classification error on the test set in %, Test error. The results are averaged over ten random 60%/40% train/test partitions and the best results in each category are in bold.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Data sets</th>
<th>bank</th>
<th>cancer</th>
<th>htru2</th>
<th>iris</th>
<th>seeds</th>
<th>sonar</th>
<th>spam</th>
<th>voice</th>
<th>wine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_1$-CGAVI</td>
<td>$</td>
<td>G</td>
<td>$</td>
<td>19</td>
<td>153</td>
<td>20</td>
<td>13</td>
<td>26</td>
<td>345</td>
<td>118</td>
</tr>
<tr>
<td>$\ell_2$-CGAVI</td>
<td>$</td>
<td>G</td>
<td>$</td>
<td>17</td>
<td>115</td>
<td>22</td>
<td>13</td>
<td>24</td>
<td>399</td>
<td>122</td>
</tr>
<tr>
<td>AGDAVI</td>
<td>$</td>
<td>G</td>
<td>$</td>
<td>21</td>
<td>132</td>
<td>18</td>
<td>14</td>
<td>25</td>
<td>475</td>
<td>120</td>
</tr>
<tr>
<td>AVI</td>
<td>$</td>
<td>G</td>
<td>$</td>
<td>15</td>
<td>288</td>
<td>101</td>
<td>22</td>
<td>47</td>
<td>1205</td>
<td>123</td>
</tr>
<tr>
<td>VCA</td>
<td>$</td>
<td>G</td>
<td>$</td>
<td>14</td>
<td>104</td>
<td>81</td>
<td>99</td>
<td>332</td>
<td>3350</td>
<td>2413</td>
</tr>
<tr>
<td>$\ell_1$-CGAVI</td>
<td>$\sum_{g \in G} g_n$</td>
<td>6.8e+1</td>
<td>5.2e+2</td>
<td>4.2e+1</td>
<td>3.0e+1</td>
<td>9.9e-1</td>
<td>1.2e+3</td>
<td>2.4e+2</td>
<td>1.9e+3</td>
<td>2.5e+2</td>
</tr>
<tr>
<td>$\ell_2$-CGAVI</td>
<td>$\sum_{g \in G} g_n$</td>
<td>1.3e+2</td>
<td>1.3e+3</td>
<td>1.2e+2</td>
<td>4.0e+1</td>
<td>7.8e+1</td>
<td>9.5e+3</td>
<td>4.5e+2</td>
<td>2.5e+3</td>
<td>2.0e+3</td>
</tr>
<tr>
<td>AGDAVI</td>
<td>$\sum_{g \in G} g_n$</td>
<td>2.2e+2</td>
<td>1.8e+3</td>
<td>7.3e+1</td>
<td>5.0e+1</td>
<td>1.3e+2</td>
<td>1.2e+4</td>
<td>4.1e+2</td>
<td>1.8e+3</td>
<td>1.6e+3</td>
</tr>
<tr>
<td>AVI</td>
<td>$\sum_{g \in G} g_n$</td>
<td>7.1e+1</td>
<td>1.2e+4</td>
<td>2.4e+3</td>
<td>1.3e+2</td>
<td>3.8e+2</td>
<td>6.5e+4</td>
<td>3.7e+2</td>
<td>2.3e+4</td>
<td>4.6e+3</td>
</tr>
<tr>
<td>VCA</td>
<td>$\sum_{g \in G} g_n$</td>
<td>1.7e+2</td>
<td>8.4e+3</td>
<td>4.7e+3</td>
<td>4.3e+3</td>
<td>3.8e+3</td>
<td>4.5e+4</td>
<td>4.0e+6</td>
<td>1.7e+6</td>
<td>9.0e+4</td>
</tr>
<tr>
<td>$\ell_1$-CGAVI</td>
<td>Test time</td>
<td>2.2e-3</td>
<td>1.1e-3</td>
<td>1.8e-3</td>
<td>9.1e-4</td>
<td>1.1e-3</td>
<td>1.8e-3</td>
<td>1.3e-3</td>
<td>1.1e-3</td>
<td>9.6e-3</td>
</tr>
<tr>
<td>$\ell_2$-CGAVI</td>
<td>Test time</td>
<td>2.0e-3</td>
<td>9.1e-4</td>
<td>1.9e-3</td>
<td>1.1e-3</td>
<td>8.4e-4</td>
<td>2.1e-3</td>
<td>1.4e-3</td>
<td>1.1e-3</td>
<td>3.1e-3</td>
</tr>
<tr>
<td>AGDAVI</td>
<td>Test time</td>
<td>2.6e-3</td>
<td>1.1e-3</td>
<td>1.6e-3</td>
<td>1.2e-2</td>
<td>1.2e-3</td>
<td>2.5e-3</td>
<td>1.5e-3</td>
<td>6.9e-4</td>
<td>2.5e-3</td>
</tr>
<tr>
<td>AVI</td>
<td>Test time</td>
<td>1.2e-3</td>
<td>1.6e-3</td>
<td>3.7e-3</td>
<td>1.6e-3</td>
<td>1.7e-3</td>
<td>2.1e-3</td>
<td>1.5e-3</td>
<td>2.0e-3</td>
<td>2.0e-3</td>
</tr>
<tr>
<td>VCA</td>
<td>Test time</td>
<td>7.6e-4</td>
<td>8.9e-4</td>
<td>2.3e-3</td>
<td>2.6e-3</td>
<td>3.2e-3</td>
<td>2.5e-3</td>
<td>8.5e-3</td>
<td>1.5e-3</td>
<td>2.4e-3</td>
</tr>
<tr>
<td>SVM</td>
<td>Test time</td>
<td>3.4e-4</td>
<td>3.8e-4</td>
<td>8.3e-2</td>
<td>1.6e-4</td>
<td>2.2e-4</td>
<td>3.2e-4</td>
<td>2.8e-2</td>
<td>2.6e-4</td>
<td>2.0e-4</td>
</tr>
<tr>
<td>$\ell_1$-CGAVI</td>
<td>Test error</td>
<td>0.09</td>
<td>3.42</td>
<td>2.05</td>
<td>4.33</td>
<td>5.95</td>
<td>20.95</td>
<td>5.90</td>
<td>19.80</td>
<td>2.08</td>
</tr>
<tr>
<td>$\ell_2$-CGAVI</td>
<td>Test error</td>
<td>0.05</td>
<td>3.29</td>
<td>2.09</td>
<td>4.17</td>
<td>5.48</td>
<td>19.88</td>
<td>6.08</td>
<td>18.24</td>
<td>2.64</td>
</tr>
<tr>
<td>AGDAVI</td>
<td>Test error</td>
<td>0.09</td>
<td>3.39</td>
<td>2.10</td>
<td>4.33</td>
<td>6.43</td>
<td>17.50</td>
<td>5.92</td>
<td>21.76</td>
<td>1.94</td>
</tr>
<tr>
<td>AVI</td>
<td>Test error</td>
<td>0.00</td>
<td>3.46</td>
<td>2.11</td>
<td>4.00</td>
<td>4.76</td>
<td>26.43</td>
<td>6.64</td>
<td>23.14</td>
<td>3.33</td>
</tr>
<tr>
<td>VCA</td>
<td>Test error</td>
<td>0.00</td>
<td>5.44</td>
<td>2.15</td>
<td>4.17</td>
<td>5.71</td>
<td>31.90</td>
<td>7.13</td>
<td>29.02</td>
<td>3.06</td>
</tr>
<tr>
<td>SVM</td>
<td>Test error</td>
<td>0.00</td>
<td>2.72</td>
<td>2.05</td>
<td>3.17</td>
<td>6.79</td>
<td>21.07</td>
<td>7.22</td>
<td>18.43</td>
<td>3.19</td>
</tr>
</tbody>
</table>

(SPAR): $\ell_1$-CGAVI produces sparse, AVI less sparse, and other algorithms practically non-sparse feature transformations.

$\sum_{g \in G} g_n$: For all data sets but seeds and voice, $\ell_1$-CGAVI creates the feature transformation with the smallest number of non-zero entries in the coefficient vectors of generators in $G$. $\ell_2$-CGAVI, AGDAVI, and AVI produce similar amounts of non-zero entries. Since VCA constructs a lot of generators, VCA often constructs the most non-zero entries. For seeds, sonar, spam, and voice, VCA constructs multiple orders of magnitude more non-zero entries than other algorithms.

Test time: The test times for the SVM are the fastest for small data sets. For large data sets, that is, htru2 and spam, the SVM is slower than the other approaches by a factor of at least 3. The test times for the monomial-based algorithms $\ell_1$-CGAVI, $\ell_2$-CGAVI, AGDAVI, and AVI are very similar. Since $\ell_1$-CGAVI represents the approximately vanishing ideal with few and sparse generators, for all data sets except for bank and cancer, $\ell_1$-CGAVI enjoys either the second-fastest or third-fastest test time. Except for bank and cancer, VCA’s test time is slow. Especially for spam, a large data set with a lot of features, VCA’s test time is 5 times slower than the test times of the monomial-based approaches and the polynomial kernel SVM.

Test error: All six algorithms enjoy comparable error rates on the test set. For most data sets, VCA tends to perform slightly worse than the other algorithms, especially on cancer, sonar, spam, and voice. OAVI tends to outperform the SVM on more complex data sets, whereas on easier data sets, the SVM has better baseline results.

### 10 Discussion

Here we present an ORACLE-based approach for the construction of generators of the approximately vanishing ideal. OAVI constructs few and, in the case of $\ell_1$-CGAVI, sparse generators. Most importantly, this compact representation of the approximately vanishing ideal does not lead to worse error rates on the test set but sometimes leads to superior generalization performance than AVI, VCA, and the polynomial kernel SVM. Under mild assumptions, we also derive two generalization bounds for CGAVI.
Acknowledgements

This research was partially funded by Deutsche Forschungsgemeinschaft (DFG) through the DFG Cluster of Excellence MATH+.

References


A Proofs

A.1 Proof of Lemma 5.1.

Proof of Lemma 5.1. By Hilbert’s basis theorem (Cox et al., 2013), there exists a finite set of generators \( \mathcal{G} \subseteq \mathcal{P} \) of \( \mathcal{I}_X \). We modify \( \mathcal{G} \) such that it satisfies the property described in the lemma while remaining a set of generators of \( \mathcal{I}_X \). Let \( h = \sum_{i=1}^{k} c_i t_i \in \mathcal{G} \) for some \( k \in \mathbb{N} \) and let \( t_j, j \in [k] \), be the \( <_\sigma \)-largest term which is divisible by the leading term of any generator in \( \mathcal{G} \) other than \( h \), that is, \( t_i \geq \sigma t \) for all \( t \in \{ t_i : i \in [k], \exists g \in \mathcal{G}, g \neq h \} \). Let \( g \in \mathcal{G} \) such that \( LT(g) | t_j \). Since both \( h \) and \( g \) vanish, so does \( \tilde{h} = h - \frac{g}{LT(g)} \).

Further, either none of the terms in \( \tilde{h} \) are divisible by the leading terms of generators in \( \mathcal{G} \) other than \( h \), or the new \( <_\sigma \)-largest term in \( \tilde{h} \) divisible by the leading term of some generator in \( \mathcal{G} \) other than \( h \) is \( <_\sigma \)-smaller than \( t_j \). Repeatedly applying this procedure results in a formulation \( h = \sum_i \alpha_i g_i + \tilde{h} \), where \( \alpha_i \in \mathbb{R} \) for all \( i \in [k] \) and \( \tilde{h} \in \mathcal{P} \) is a vanishing polynomial none of whose terms are divisible by the leading terms of generators in \( \mathcal{G} \) other than \( h \). We can thus replace \( h \) in \( \mathcal{G} \) by \( \tilde{h} \) and the updated \( \mathcal{G} \) is still a set of generators. Performing this procedure for all generators in \( \mathcal{G} \), from the generator with \( <_\sigma \)-smallest leading term to the generator with \( <_\sigma \)-largest leading term, transforms \( \mathcal{G} \) into a set of generators with the property described in the lemma. \( \square \)

A.2 Proof of Theorem 6.1

Proof of Theorem 6.1. Let \( i \in \mathbb{N}_{\geq 1} \). Since \( X \subseteq [-1,1]^n \) is bounded, for any term \( t \in \mathcal{T}_1 \), we have that

\[
\frac{1}{m} \left\| t^{i+2}(X) - t^i(X) \right\|_2^2 \leq 2(\max\{|t(x)| : x \in X, |t(x)| < 1\})^i.
\]

As \( i \to \infty \), the right-hand side tends to 0. Consider the polynomial \( g := t^{i+2} - t^i \). Then, for any \( \psi \geq \varepsilon \geq 0 \), there exists \( i^* \in \mathbb{N} \) such that

\[
MSE(g, X) = \frac{1}{m} \left\| t(X)^{i^*+2} - t(X)^{i^*} \right\|_2^2 \leq \varepsilon \leq \psi.
\]

Thus, if no polynomial with leading term divisible by \( i \) is appended to \( \mathcal{G} \) before reaching degree \( i^* + 2 \), \( \text{OAVI} \) appends a polynomial \( g \) with \( LT(g) = t^{i^*+2} \) to \( \mathcal{G} \). The same observation holds for all other terms in \( \mathcal{T}_1 \) and, thus, there exists a degree \( d^* \in \mathbb{N} \) such that for any \( t \in \mathcal{T}_1 \), there exists a polynomial \( g \in \mathcal{G} \) with degree at most \( d^* \) and \( t | LT(g) \). Therefore, all terms in \( \partial_{d^*} \mathcal{O} \) are divisible by at least one leading term of a polynomial in \( \mathcal{G} \) and \( \partial_{d^*} \mathcal{O} = \emptyset \), guaranteeing termination of \( \text{OAVI} \). \( \square \)

A.3 Proof of Theorem 6.2

Proof of Theorem 6.2. We prove the first statement. By Theorem 6.1, \( \text{OAVI} \) terminates with \( \mathcal{G} \neq \emptyset \). Thus, by construction, at the end of \( \text{OAVI} \)'s execution, \( \mathcal{G} \) contains \( (\psi, 1) \)-approximately vanishing polynomials. For the second statement, suppose towards a contradiction that there exists a \( (\psi - \varepsilon, 1) \)-approximately vanishing polynomial \( f \) with terms only in \( \mathcal{O} \) and \( LT(f) = t \in \mathcal{O} \). Let \( \mathcal{U} := \{ u \in \mathcal{O} : t >_\sigma u \} = \{ u_1, \ldots, u_k \}_\sigma \). At some point during its execution \( \text{OAVI} \) constructs a polynomial \( g \) with \( LT(g) = t \) and other terms only in \( \mathcal{U} \). By \( \varepsilon \)-accuracy of \( \text{ORACLE} \),

\[
MSE(g, X) \leq \min_{v \in \mathbb{R}^k} \frac{1}{m} \left\| U(X)v + t(X) \right\|_2^2 + \varepsilon \leq MSE(f, X) + \varepsilon \leq \psi - \varepsilon + \varepsilon \leq \psi,
\]

a contradiction. \( \square \)

A.4 Proof of Proposition 6.3

Proof of Proposition 6.3. Consider the execution of \( \text{OAVI} \) with \( X \) and \( \psi \geq \varepsilon \geq 0 \). Suppose that \( \text{OAVI} \) is currently inside the while loop corresponding to \( \mathcal{O} = \{ t_1, \ldots, t_m \}_\sigma \) and border \( \partial_i \mathcal{O} \). Should this never occur, \( |\mathcal{O}| \leq m \) is satisfied automatically. Let \( \mathcal{R} \subseteq \mathcal{T}_d \) denote the remaining terms in \( \partial_i \mathcal{O} \) for which \( \text{OAVI} \) has not yet checked whether they are leading terms of approximately vanishing polynomials. Since the \( m \) columns of \( \mathcal{O}(X) \) are linearly independent, for any \( u \in \mathcal{R} \), there exists \( c \in \mathbb{R}^m \) such that

\[
\mathcal{O}(X)c + u(X) = 0.
\]
Hence, there exists a \((0, 1)\)-approximately vanishing polynomial with leading term \(u\) and other terms only in \(\mathcal{O}\). By \(\varepsilon\)-accuracy of \text{ORACLE}, \(\psi \geq \varepsilon\), and Theorem 4.1, \text{OAVI} appends a \((\psi, 1)\)-approximately vanishing polynomial with leading term \(u\) and other terms only in \(\mathcal{O}\) to \(\mathcal{G}\). Since this observation holds for any term in the remainder of the current border or the upcoming border, that is, \(\mathcal{R}\) or \(\partial_{d+1}\mathcal{O}\), respectively, no more terms get appended to \(\mathcal{O}\). If no terms get appended to \(\mathcal{O}\) for all terms in the border corresponding to a specific degree, then the border corresponding to the next higher degree is empty and \text{OAVI} terminates, proving the first statement.

For the second statement, suppose that \text{OAVI} terminates after performing the while loop corresponding to degree \(D \in \mathbb{N}\). By construction, leading terms of polynomials in \(\mathcal{G}\) are contained in \(\bigcup_{d=1}^{D} \partial_d \mathcal{O}\). Hence, \(|\mathcal{G}| \leq \left|\bigcup_{d=1}^{D} \partial_d \mathcal{O}\right| \leq |\mathcal{O}|n \leq mn\).

\[\square\]

### A.5 Proof of Theorem 6.4

**Proof of Theorem 6.4.** By Theorem 6.1, \text{OAVI} is guaranteed to terminate. Suppose that \text{OAVI} terminates after performing the while loop corresponding to degree \(D \in \mathbb{N}\). We first address the computational cost of \text{OAVI}, which is determined by two factors, the time required to construct the evaluation vectors of terms contained in any border, i.e., \(\bigcup_{d=1}^{D} \partial_d \mathcal{O}\), and the time required to perform all the calls to \text{ORACLE}, which is exactly once for every term in \(\bigcup_{d=1}^{D} \partial_d \mathcal{O}\).

We start with a brief overview of how the degree \(d\) border, \(\partial_d \mathcal{O}\), is constructed in \text{OAVI}. First, we construct \(C_d = \{ u \in T_d \mid u = s \cdot t\ for s \in \mathcal{O}_1\ and\ t \in \mathcal{O}_{d-1}\}\), a superset of \(\partial_d \mathcal{O}\). Second, we determine \(C_d \setminus \partial_d \mathcal{O}\), that is, the set of terms \(u \in C_d\) such that there exists a polynomial \(g \in \mathcal{G}_{\leq d-1}\) with \(\text{LT}(g) \mid u\). Third, we remove \(C_d \setminus \partial_d \mathcal{O}\) from \(C_d\) to obtain \(\partial_d \mathcal{O}\). Fourth, we evaluate all terms in \(\partial_d \mathcal{O}\).

We now show that it requires time \(O(m^2n^2)\) to construct \(\bigcup_{d=1}^{D} \partial_d \mathcal{O}\) and the evaluation vectors of terms therein. Note that the evaluation vectors of terms in \(\partial_d \mathcal{O}\) are identical to the elements of input \(X\) of \text{OAVI} and thus require time \(O(1)\) to construct. We thus consider any \(d \in \{2, \ldots, D\}\) and suppose that \(\mathcal{O}_1, \mathcal{G}_1,\) and \(\partial_1 \mathcal{O}\) and the evaluation vectors of \(\mathcal{O}_1, \mathcal{G}_1,\) and \(\partial_1 \mathcal{O}\) are already determined for all \(i \leq d - 1\). First, assuming that multiplying two terms requires constant time, to construct \(C_d\), every term in \(\mathcal{O}_{d-1}\) gets multiplied by all terms in \(\mathcal{O}_1\) exactly once, requiring a total of \(O(|\mathcal{O}_{d-1}|n)\) multiplications of terms. Second, assuming that it takes constant time to determine whether a term is divisible by another term, it requires \(O(|C_d| |\mathcal{G}_{\leq d-1}|) \leq O(|C_d| |\mathcal{G}|)\) to determine all terms in \(C_d\) that have to be deleted. Third, assuming that a term can be deleted in constant time, it takes \(O(|C_d|)\) time to perform all deletions of terms in \(C_d\). Fourth, evaluating a term \(t \in \partial_d \mathcal{O}\) requires entrywise multiplication of two \(m\)-dimensional evaluation vectors of terms in \(\mathcal{O}_{\leq d-1}\), requiring time \(O(m)\). Since we have to evaluate at most \(|C_d|\) terms, the evaluation step requires time \(O(|C_d|m)\). Thus, it requires time \(O(|C_d|(|\mathcal{G}| + 1 + m)) = O(|C_d| |\mathcal{G}|)\) to construct the evaluation vectors of terms in \(\partial_d \mathcal{O}\), where equality follows due to Proposition 6.3. Thus, to construct \(\bigcup_{d=1}^{D} \partial_d \mathcal{O}\) and the evaluation vectors of terms therein, it requires time \(O\left(\sum_{d=1}^{D} |\mathcal{C}_d||\mathcal{G}| = O(|\mathcal{C}| |\mathcal{G}|) = O(m^2n^2)\right)\), where the first equality holds because the sets \(C_d\) for \(d \in [D]\) are pairwise disjoint and the second equality holds due to \(O(|\mathcal{C}|) = O(|\mathcal{O}||\mathcal{O}_1|) = O(mn)\) and Proposition 6.3.

We next determine the time complexity of the \text{ORACLE} calls during \text{OAVI}’s execution. \text{ORACLE} is called exactly once for every border term, i.e., \(O\left(\bigcup_{d=1}^{D} \partial_d \mathcal{O}\right) = O(|\mathcal{G}| + |\mathcal{O}|) = O(mn)\) times, requiring time \(O(mnT_{\text{ORACLE}})\). Thus, \text{OAVI}’s total time complexity is \(O(m^2n^2 + mnT_{\text{ORACLE}})\).

Throughout the algorithm’s execution, we store \(\mathcal{O}, \mathcal{G},\) and \(\mathcal{C}\), and the evaluation vectors of terms therein. Assuming that we can store a term \(t \in \mathcal{T}\) and the corresponding evaluation vector \(t(X) \in \mathbb{R}^m\) in space \(O(m)\), \text{OAVI} requires space \(O(|\mathcal{C}| + |\mathcal{G}| + |\mathcal{O}|)m = O(m^2n)\). To store the generators in \(\mathcal{G}\), we further store \(|\mathcal{G}|\) vectors of length \(O(m)\). Since the space complexity of \text{ORACLE} affects the total space complexity only additively, the total space complexity is \(O(m^2n + S_{\text{ORACLE}})\). 

\[\square\]

### A.6 Proof of Theorem 6.5

**Proof of Theorem 6.5.** The proof is an adaptation of the proof of Theorem 5.1 in Livni et al. (2013) to \text{OAVI}. Let \(\mathcal{O} = \{ t_1, \ldots, t_k \}_\nu\) with \(k \leq m\). Since \(t_1 = 1\), we can evaluate it in time \(O(1)\). Since the evaluation vectors of monomials in \(\mathcal{O}_1\) over \(Z\) are identical to the data points \(z \in Z\), all degree 1 monomials can be evaluated in time \(O(1)\). The evaluation of a term \(t \in \mathcal{T}\) of degree greater than 1 over \(Z\) can be computed by multiplying the evaluation vectors of two \(q\)-dimensional monomials element-wise, requiring time \(O(q)\). Hence, we can construct
In this section, we derive a margin bound for the approach of using \( \tau \) and degree-bounded generators of the approximately vanishing ideal to transform features for a linear kernel SVM.

To do so, we recall some definitions from Mohri et al. (2018). Let \( \mathcal{X} \subseteq [-1,1]^n \) and \( \mathcal{Y} = \{0,1\} \) and recall that a hypothesis \( h \) is a mapping from \( \mathcal{X} \) to \( \mathcal{Y} \).

**Definition B.1** (Margin loss function). For any \( \rho > 0 \), the \( \rho \)-margin loss is the function \( L_\rho : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_{>0} \) defined for all \( y, y' \in \mathbb{R} \) as

\[
L_\rho(y, y') = \min \left\{ 1, \max \left\{ 0, 1 - \frac{y y'}{\rho} \right\} \right\}.
\]
The margin loss functions is used to define the empirical margin loss, i.e., the error on the training set.

**Definition B.2** (Empirical margin loss). Let $\mathcal{X} \subseteq [-1,1]^n$, $\mathcal{Y} = \{0,1\}$, $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$, $S = \{(x_1,y_1),\ldots,(x_m,y_m)\} \in (\mathcal{X} \times \mathcal{Y})^m$ be sampled according to $\mathcal{D}$, and $h$ be a hypothesis. The empirical margin loss of $h$ is defined as

$$\hat{R}_{S,\rho}(h) = \frac{1}{m} \sum_{i=1}^{m} L_{\rho}(h(x_i), y_i).$$

In the theorem below, we present a margin bound for the approach of using $\tau$- and degree-bounded generators of the approximately vanishing ideal to transform features for a linear kernel SVM.

**Theorem B.3** (Margin bound for feature transformations for a linear kernel SVM). Let $\mathcal{X} \subseteq [-1,1]^n$, $\mathcal{Y} = \{0,1\}$, $\mathcal{D}$ be a distribution over $\mathcal{X} \times \mathcal{Y}$, $\mathcal{G} = \{g_1,\ldots,g_k\} \subseteq \mathcal{P}$ such that $|g(x)| \leq \tau$ for all $g \in \mathcal{G}$ and $x \in \mathcal{X}$, $S = \{(x_1,y_1),\ldots,(x_m,y_m)\} \in (\mathcal{X} \times \mathcal{Y})^m$ be sampled according to $\mathcal{D}$, and $\tilde{S} = \{(\tilde{g}(x),y) \mid (x,y) \in S\}$. Let $K: [\tau,\tau]^k \times [\tau,\tau]^k \to \mathbb{R}$ denote the linear kernel and $\mathcal{H} = \{\mathbb{R}^k \ni v \mapsto w^T v : \|w\|_2 \leq \Lambda\}$ for some $\Lambda \geq 0$. Fix $\rho > 0$. Then, for any $\delta > 0$, with probability at least $1 - \delta$, for any $h \in \mathcal{H}$, we have

$$\mathbb{P}(x,y) \sim_{\mathcal{D}} [h(\tilde{g}(x)) \neq y] \leq \hat{R}_{S,\rho}(h) + 2 \frac{\sqrt{\frac{\kappa^2 \Lambda^2}{\rho^2 m} + \frac{\log(\delta^{-1})}{2m}}}{\sqrt{}}.$$

**Proof.** Let $\tilde{\mathcal{X}} = \{\tilde{g}(x) \mid x \in \mathcal{X}\}$. Given a distribution $\tilde{\mathcal{D}}$ over $\tilde{\mathcal{X}} \times \mathcal{Y}$, let $\tilde{\mathcal{D}}$ denote an auxiliary distribution, samples $(\tilde{x},y) \sim \tilde{\mathcal{D}}$ of which are obtained by drawing a sample $(x,y) \sim \mathcal{D}$ and letting $\tilde{x} := \tilde{g}(x)$. Then, for the linear kernel $K: \tilde{\mathcal{X}} \times \tilde{\mathcal{X}} \to \mathbb{R}$, we have

$$\sup_{\tilde{x} \in \tilde{\mathcal{X}}} K(\tilde{x},\tilde{x}) = \langle \tilde{x},\tilde{x}\rangle \leq k \tau^2.$$

We then apply the generalization bound for kernel-based classifiers (Mohri et al., 2018, Corollary 6.13) to data sampled according to $\tilde{\mathcal{D}}$, proving the theorem.

For $\mathcal{X} \subseteq [-1,1]^n$ and any set of $\tau$-bounded polynomials $\mathcal{G} \subseteq \mathcal{P}$ in norm $\| \cdot \|_1$, it holds that $|g(x)| \leq \tau$ for all $x \in \mathcal{X}$ and $g \in \mathcal{G}$. Thus, using the notation of Section 7, if we apply a linear kernel SVM to the data set transformed according to the feature transformation (FT) corresponding to a set of $\tau$-bounded generators in norm $\| \cdot \|_1$ constructed by OAVI, AVI, or VCA, Theorem B.3 is satisfied. However, the margin bound in Theorem B.3 relies on the number of polynomials in $\mathcal{G} = \bigcup_i \mathcal{G}^i$ to be $o(m)$, where $\mathcal{G}^i$ is the set of generators constructed for class $i$. This can be achieved by bounding the number of generators used to transform the features of the original data set, for example, via an upper bound on the maximum degree of generators.

To summarize, since generators constructed with CGAVI are $\tau$-bounded in norm $\| \cdot \|_1$, whereas generators constructed with AVI or VCA are not necessarily $\tau$-bounded in norm $\| \cdot \|_1$, Theorem B.3 applies to CGAVI when the algorithm is terminated after reaching a certain degree, but, in general, does not apply to AVI or VCA. Thus, if we apply a linear kernel SVM to the data set transformed according to the feature transformation (FT) corresponding to a set of generators of bounded degree constructed with CGAVI, increasing the number of training samples and decreasing the training error decreases the probability of misclassifying out-sample data.

### C The PF\textsuperscript{W} algorithm

The Pairwise Frank-Wolfe algorithm PF\textsuperscript{W} is presented in Algorithm 3. Let $f: \mathbb{R}^n \to \mathbb{R}$ be a convex and smooth function and $\mathcal{A} = \{v_1,\ldots,v_k\} \subseteq \mathbb{R}^n$ a set of vectors. Then, PF\textsuperscript{W} solves

$$\min_{x \in \text{conv} (\mathcal{A})} f(x),$$

where conv$(\mathcal{A})$ is the convex hull of $\mathcal{A}$. At iteration $t = 0,\ldots,T$, PF\textsuperscript{W} writes the current iterate, $x_t$, as a convex combination of elements of $\mathcal{A}$, that is, $x_t = \sum_{v \in \mathcal{A}} \lambda_v^{(t)} v$, where $\sum_{v \in \mathcal{A}} \lambda_v^{(t)} = 1$ and $\lambda_v^{(t)} \in [0,1]$ for all $v \in \mathcal{A}$. At iteration $t$, a vertex $v$ whose corresponding weight $\lambda_v^{(t)}$ is not zero is referred to as an *active vertex* and the set $S^{(t)} = \{v \in \mathcal{A} \mid \lambda_v^{(t)} > 0\}$ is referred to as the *active set* at iteration $t$. During each iteration, PF\textsuperscript{W} determines
Algorithm 3 PFW

**Input:** A smooth and convex function \( f \), a set of atoms \( \mathcal{A} \subseteq \mathbb{R}^n \), a vertex \( x_0 \in \mathcal{A} \), and \( T \in \mathbb{N} \).

**Output:** A point \( x_T \in \text{conv}(\mathcal{A}) \).

1. \( S^{(0)} \leftarrow \{x_0\} \)
2. \( \lambda^{(0)}_v \leftarrow 1 \) for \( v = x_0 \) and 0 otherwise
3. for \( t = 0, \ldots, T \) do
   4. \( s_t \leftarrow \text{argmin}_{s \in \mathcal{A}} \langle \nabla f(x_t), s \rangle \)
   5. \( d_t^{\text{FW}} \leftarrow s_t - x_t \)
   6. \( v_t \leftarrow \text{argmax}_{v \in S^{(t)}} \langle \nabla f(x_t), v \rangle \)
   7. \( d_t^{\text{AW}} \leftarrow x_t - v_t \)
   8. \( d_t \leftarrow d_t^{\text{FW}} + d_t^{\text{AW}} \)
   9. \( \gamma_t = \text{argmin}_{\gamma \in [0, \lambda_{v_t}]} f(x_t + \gamma d_t) \)
10. \( x_{t+1} \leftarrow x_t + \gamma_t d_t \)
11. \( \lambda^{(t+1)}_v \leftarrow \lambda^{(t)}_v + \gamma_t \)
12. \( \lambda^{(t+1)}_{v_t} \leftarrow \lambda^{(t)}_{v_t} - \gamma_t \)
13. \( S^{(t+1)} \leftarrow \{v \in \mathcal{A} | \lambda^{(t+1)}_v > 0\} \)
14. end for

Two vertices requiring access to a first-order oracle and a linear minimization oracle. In Line 4, PFW determines the Frank-Wolfe vertex, which minimizes the scalar product with the gradient of \( f \) at iterate \( x_t \). Taking a step of appropriate size towards the Frank-Wolfe vertex, in the Frank-Wolfe direction, reduces the objective function value. In Line 7, PFW determines the Away vertex in the active set, which maximizes the scalar product with the gradient of \( f \) at iterate \( x_t \). Taking a step away from the Away vertex, in the Away direction, reduces the objective function value. In Line 8, PFW combines the Away direction and the Frank-Wolfe direction into the Pairwise direction and takes a step with optimal step size in the Pairwise direction, shifting weight from the Away vertex to the Frank Wolfe vertex. In each iteration, PFW thus only modifies two entries of the iterate \( x_t \), which is the main reason why PFW tends to return a sparse iterate \( x_T \). The iterate \( x_T \) returned by PFW satisfies \( f(x_T) - \min_{x \in \text{conv}(\mathcal{A})} f(x) = O (e^{-T}) \) (Lacoste-Julien and Jaggi, 2015) when optimizing the squared loss over the \( \ell_1 \)-ball. However, PFW's linear convergence rate is theoretically cumbersome to bound due to its dependence on various constants and the number of so-called swap steps when a vertex in the active set gets replaced by another vertex. To address this issue, one could replace PFW with a swap-step free version called Blended Pairwise Conditional Gradients, whose convergence rate constants are smaller than those of PFW (Tsuji et al., 2021). In practice, instead of computing the number of iterations to reach \( \varepsilon \)-accuracy, we use the Frank-Wolfe gap as a stopping criterion for PFW and we do not have to compute \( T \).

When using PFW as ORACLE for OAVI, we implement ORACLE in Line 6 as follows: Run PFW with \( f, \mathcal{A} \) the set of vertices of the \( \ell_1 \)-ball of radius \( \tau - 1 \), \( x_0 = (1, 0, \ldots, 0)^T \), and \( T \in \mathbb{N} \) such that PFW achieves \( \varepsilon \)-accuracy and obtain \( x_T \). Then, \((x_T^T, 1)^T\) is the coefficient vector of the polynomial \( g \) returned by ORACLE.

D Additional numerical experiments

We use different algorithms that construct generators of the approximately vanishing ideal to transform features for a linear kernel SVM and compare them to each other and a polynomial kernel SVM.

As in Section 9, let \( \mathcal{G} := \bigcup_i \mathcal{G}^i \), where \( \mathcal{G}^i \) is the set of generators constructed for class \( i \) and let \( O^i \) denote the set of terms that are not leading terms of generators constructed for class \( i \). By Theorem 6.5, \((\mathcal{G}^i, O^i) = OAVI(X, \psi, \varepsilon)\) for \( X \subseteq [-1, 1]^n \) and \( \psi \geq \varepsilon \geq 0 \), the time required to evaluate a test set of \( q \) data points is \( O(|\mathcal{G}^i|^{|O^i|q}) \). It thus remains to compare \( \sum_i |O^i| \), the number of terms that are provably not leading terms of generators summed over all classes for the monomial-based algorithms, \( \ell_1\)-CGAVI, \( \ell_2\)-CGAVI, AGDAVI, and AVI, and the number of non-approximately vanishing polynomials constructed for the polynomial-based algorithm, VCA.

Set-up. We apply different feature transformation algorithms for a linear kernel SVM and a polynomial kernel SVM to classification problems using the set-up from Section 9. We compare the number of terms constructed,

---

\(^8\)Recall that for VCA, \( O^i \) is a set of non-approximately vanishing polynomials instead.
Table 3: We compare the number of terms constructed, $\sum_i |O^i|$, the maximum degree of generators constructed, Degree, the number of entries in the coefficient vectors of generators in $G$, $\sum_{e \in G} g_e$, the hyperparameter optimization time in seconds, Hyp. time, the training time in seconds, Train time, and the classification error on the training set in %, Train error. The results are averaged over ten random 60%/40% train/test partitions and the best results in each category are in bold.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Data sets</th>
<th>Bank</th>
<th>Cancer</th>
<th>Htru2</th>
<th>Iris</th>
<th>Seeds</th>
<th>Sonar</th>
<th>Spam</th>
<th>Voice</th>
<th>Wine</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_1$-CGAVI</td>
<td>1.6e+2</td>
<td>3.6e+1</td>
<td>1.3e+2</td>
<td>6.1e+3</td>
<td>3.9e+2</td>
<td>3.9e+2</td>
<td>6.9e-2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\ell_2$-CGAVI</td>
<td>1.3e+2</td>
<td>7.8e+1</td>
<td>9.5e-3</td>
<td>4.5e-2</td>
<td>2.5e+0</td>
<td>2.5e+3</td>
<td>2.0e+3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVI</td>
<td>7.2e+1</td>
<td>1.3e+4</td>
<td>2.4e+3</td>
<td>4.5e-3</td>
<td>3.8e+6</td>
<td>4.0e+6</td>
<td>1.7e+6</td>
<td>9.0e+4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VCA</td>
<td>1.5e-1</td>
<td>2.0e-1</td>
<td>4.0e-1</td>
<td>1.4e+2</td>
<td>3.1e+2</td>
<td>3.4e+2</td>
<td>1.2e+3</td>
<td>1.2e+2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Degree</td>
<td>$\ell_1$-CGAVI</td>
<td>2.5e+2</td>
<td>9.4e+2</td>
<td>1.1e+3</td>
<td>7.1e+2</td>
<td>2.3e+3</td>
<td>1.2e+3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\ell_2$-CGAVI</td>
<td>1.9e+2</td>
<td>5.8e+2</td>
<td>1.5e+3</td>
<td>6.6e+2</td>
<td>3.8e+3</td>
<td>9.8e+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVI</td>
<td>8.0e+1</td>
<td>1.8e+2</td>
<td>2.3e+2</td>
<td>5.3e+2</td>
<td>1.1e+3</td>
<td>3.4e+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VCA</td>
<td>4.7e+1</td>
<td>9.2e+2</td>
<td>3.4e+2</td>
<td>7.5e+1</td>
<td>1.1e+3</td>
<td>1.7e+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train error</td>
<td>$\ell_1$-CGAVI</td>
<td>3.2e-1</td>
<td>9.9e+1</td>
<td>1.2e-0</td>
<td>2.0e-1</td>
<td>2.1e+0</td>
<td>9.9e-1</td>
<td>2.5e+0</td>
<td>5.9e-1</td>
<td></td>
</tr>
<tr>
<td>$\ell_2$-CGAVI</td>
<td>2.1e-1</td>
<td>6.2e-2</td>
<td>9.9e+2</td>
<td>3.7e+0</td>
<td>9.9e-1</td>
<td>3.5e+0</td>
<td>1.2e+0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVI</td>
<td>1.4e-2</td>
<td>4.0e-2</td>
<td>6.8e-2</td>
<td>1.5e-0</td>
<td>8.6e-1</td>
<td>7.9e-1</td>
<td>3.4e-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VCA</td>
<td>3.8e-2</td>
<td>4.8e-2</td>
<td>1.0e-1</td>
<td>8.4e-1</td>
<td>1.3e-0</td>
<td>3.1e-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVM</td>
<td>2.5e-3</td>
<td>4.7e-2</td>
<td>8.6e-1</td>
<td>1.0e-1</td>
<td>2.1e+0</td>
<td>5.1e-1</td>
<td>2.0e-1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Implementation, data sets, and hyperparameters. Implementation, data sets, and hyperparameters are the same as in Section 9. For the polynomial kernel SVM, $\sum_i |O^i|$, Degree, and $\sum_{g \in G} g_e$ do not exist.

Results. 
$\sum_i |O^i|$: The three OAVI algorithms construct a similar number of terms that are not leading terms of generators for all data sets and for htru2, iris, seeds, sonar, voice, and wine, construct fewer terms that are not leading terms of generators than AVI and VCA.

Degree: Even though we use $\ell_2$-regularized squared loss for OAVI, the maximum degrees of the generators constructed by OAVI are comparable to the maximum degrees of generators constructed by AVI and VCA.

$\sum_{g \in G} g_e$: For all data sets but bank, one of the OAVI algorithms creates the feature transformation with the smallest number of entries in the coefficient vectors of generators in $G$. For most data sets, VCA constructs the feature transformation with the largest number of entries in the coefficient vectors of generators in $G$ and for seeds, sonar, spam, and voice, VCA constructs multiple orders of magnitude more entries in the coefficient vectors.
of generators in $G$ than the other algorithms.

Hyp. time: All algorithms except the SVM achieve comparable hyperparameter optimization times. For all data sets, tuning the hyperparameters for AGDAVI requires less time than tuning the hyperparameters for $\ell_1$- and $\ell_2$-CGAVI. For all data sets, the SVM requires the least amount of time for hyperparameter optimization.

Train time: All algorithms except the SVM achieve comparable training times. For all data sets, training AGDAVI requires less time than training $\ell_1$- and $\ell_2$-CGAVI. For five of the nine data sets, VCA can be trained faster than OAVI and AVI. For all data sets, the SVM requires the least amount of time for training.

Train error: All algorithms achieve comparable errors on the training data. However, it is noteworthy that VCA achieves perfect training accuracy on four of the nine data sets.