Testing Juntas (draft)

Eric Blais
Carnegie Mellon University
eblais@cs.cmu.edu

February 3, 2010

Abstract

A function on n variables is called a k-junta if it depends on at most k of its variables. The problem of *testing* whether a function is a k-junta or is "far" from being a k-junta is a central problem in property testing and is closely related to the problem of learning high-dimensional data.

In this note, we give an informal presentation of three recent algorithms for testing juntas efficiently.

1 Introduction

A function $f : \{0,1\}^n \to \{0,1\}$ is said to be a k-junta if it depends on at most k variables. Juntas provide a clean model for studying learning problems in the presence of many irrelevant features [5, 8], and have consequently been of particular interest to the computational learning theory community [6, 8, 18, 16, 17].

As is typical in the machine learning setting, all learning results on k-juntas assume that the unknown function is a k-junta. In practice, however, it is often not known *a priori* whether a function being learned is a k-junta or not. It is therefore desirable to have an efficient algorithm for *testing* whether a function is a k-junta or "far" from being a k-junta before attempting to run any k-junta learning algorithm.

We consider the problem of testing k-juntas in the standard property testing framework originally defined by Rubinfeld and Sudan [20]. In this framework, we say that a function f is *ϵ-far* from being a k-junta if for every k-junta g, the functions f and g disagree on at least an ϵ fraction of all inputs.

An *ϵ-tester for k-juntas* is an algorithm $\mathcal{A}$ that queries an unknown function f on q inputs of its choosing, and then (1) accepts f with probability at least $2/3$ when f is a k-junta, and (2) rejects f with probability at least $2/3$ when f is ϵ-far from being a k-junta.

When the algorithm $\mathcal{A}$ chooses all its queries in advance (i.e., before observing the values of the function on any of its previous queries), it is *non-adaptive*; otherwise it is *adaptive*.

The main question of interest when testing k-juntas is the following:

What is the minimum number of queries required to ϵ-test k-juntas?

This short note provides an informal introduction to three recent algorithms for testing juntas with a very small number of queries. The goal of this note is to highlight the simplicity and naturalness of the algorithms, and as a result many technical (but interesting!) details are omitted from this presentation; for a more thorough presentation of the same topic, the interested reader is referred to the original articles cited throughout.

Outline. We begin in Section 2 with an algorithm of Parnas, Ron, and Samorodnitsky [19] that yields an efficient tester for 1-juntas. In Section 3, we see an algorithm of Fischer, Kindler, Ron, Safra, and Samorodnitsky [13] for testing k-juntas with a number of queries that is polynomial in k and *independent* of n. In Section 4, we present a recent algorithm [4] for testing k-juntas with only $O(k \log k + k/\epsilon)$ queries. Finally, we discuss some related work and open problems in Section 5.

2 Testing 1-juntas

The first result explicitly related to testing juntas was obtained by Parnas, Ron, and Samorodnitsky [19], who generalized a result of Bellare, Goldreich, and Sudan [2] on testing long codes to obtain an algorithm for testing 1-juntas with only $O(1/\epsilon)$ queries.

The class of functions that are 1-juntas is very small: it contains only the constant functions, the *dictator* functions, and the *anti-dictator* functions. The function f is a dictator function if there is an index $i \in [n]$ such that $f(x) = x_i$ for every input $x \in \{0,1\}^n$. And f is an anti-dictator if the function $\bar{f}$ defined by $\bar{f}(x) = 1 - f(x)$ is a dictator function.

Parnas et al. introduced a simple but efficient algorithm for testing dictator functions with only $O(1/\epsilon)$ queries [19]. This algorithm also yields a 1-junta tester: simply run the dictator test on f and $\bar{f}$, and also test if f is constant; accept the function iff any of those three tests accept. Testing constant functions can be trivially done with $O(1/\epsilon)$ queries, so the resulting 1-junta tester also only requires $O(1/\epsilon)$ queries.

The dictator testing algorithm of Parnas et al. uses the fact that dictator functions are the only non-zero functions that are both *linear* and *monotone*. Their test is the natural one: test whether the function is linear (or at least, is not far from linear), test that it is not the zero function, then test whether it is monotone. There is one twist to the algorithm: in order to guarantee that it correctly rejects functions ϵ-far from dictator functions with high probability, it also uses the *self-correction* property of nearly-linear functions.

DICTATOR TEST (f, ϵ)

1. Verify that f is linear using an ϵ-tester for linearity.
2. Using self-correction, verify that $f(1) = 1$.
3. For 64 randomly selected pairs x, y,
   3.1. Verify with self-correction that $f(x \wedge y) = f(x) \wedge f(y)$.
4. **Accept** the function iff all the above tests passed.

For more details on the algorithm and its analysis, readers are referred to the original article of Parnas et al. [19]. More information on linearity and self-correction can also be found in [9].

3 Testing k-juntas

The algorithm outlined in the previous section relies on special properties of 1-juntas, so it cannot be generalized to test k-juntas for any $k \geq 2$. Nonetheless, Fischer, Kindler, Ron, Safra, and Samorodnitsky [13] showed that for any constant k, it is possible to test k-juntas with a number of queries that is *independent* of n.

3.1 Algorithm

Fischer et al. [13] introduced multiple testing algorithms for testing k-juntas. In this section we focus on their simplest algorithm. We briefly discuss some of its more query-efficient variants in Section 3.3.

The algorithm for testing k-juntas relies on two basic components: the *independence test* and the idea of *partitioning* the coordinates.

Independence Test. The independence test is a simple test for verifying that a given function f does not depend on a set $S \subseteq [n]$ of coordinates: it selects a random pair (x, y) of inputs whose values differ only in the coordinates in S, and tests whether $f(x) = f(y)$.

INDEPENDENCE TEST (f, S)

1. Generate $x, y \in \{0,1\}^n$ uniformly at random from all the pairs for which $x_{\bar{S}} = y_{\bar{S}}$.
2. **Accept** iff $f(x) = f(y)$.

It is easy to verify that when f is independent of the coordinates in S, then the Independence test always accepts. Conversely, Fischer et al. [13] showed that when f "strongly" depends on the coordinates in S, then the test rejects with non-negligeable probability, so running the independence test multiple times on S leads to a at least one rejection with high probability. (This fact is discussed in more details in the next section.)

Partitioning. There is a very simple algorithm for testing whether a function is a k-junta: run the independence test (sufficiently many times) on each singleton set $S = \{1\}, \{2\}, \ldots, \{n\}$ and accept iff at most k of those sets are rejected. The resulting algorithm is a valid k-junta tester, but has a very large query complexity. In order to reduce the query complexity, Fischer et al. [13] showed that one can also test k-juntas by randomly partitioning the coordinates into poly(k) buckets and running the independence test on each bucket (a sufficient number of times).

> k-JUNTA TEST (f, ϵ)
>
> 1. Randomly partition the coordinates into $O(k^2)$ buckets.
> 2. Run the INDEPENDENCE TEST $\tilde{O}(k^2/\epsilon)$ times on each bucket.
> 3. **Accept** iff at most k buckets fail the independence test.

This k-junta testing algorithm has query complexity $\tilde{O}(k^4/\epsilon)$. It is clear that it always accepts k-juntas, since at most k buckets contain a relevant coordinate and those are the only buckets that may be rejected by the independence test. In the next section, we briefly discuss the methods that Fischer et al. used to show that the algorithm also correctly rejects functions ϵ-far from k-juntas with high probability.

3.2 Analysis

Fischer et al. [13] used the *influence* of sets of variables to establish the correctness of their junta testing algorithm.[1] The influence of the set $S \subseteq [n]$ in f is defined to be

$$\mathrm{Inf}_f(S) = 2 \Pr_{x,y \in \{0,1\}^n}[f(x) \neq f(x_S y_{\bar{S}})].$$

That is, the influence of S is proportional to the probability that re-randomizing the values of the variables in S of a randomly chosen input changes the value of the function. (The "2" multiplier normalizes the measure to lie in the interval $[0, 1]$.)

The first key insight of Fischer et al. in the analysis of their algorithm is that the probability that the Independence Test rejects a set S is exactly $\mathrm{Inf}_f(S)/2$. The second

[1] In [13], the notion of influence is called *variation*, but is otherwise as described here.

insight is that when a function is ϵ-far from being a k-junta, then a random partition of its variables into $O(k^2)$ buckets leaves at least $k+1$ buckets with influence at least $O(\epsilon/k^2)$ with high probability. Both of those facts are established through Fourier analysis; the details can be found in the original work of Fischer et al. [13].

3.3 Improvements

The algorithm for testing k-juntas presented above is extremely elegant and simple, but it does not have optimal query complexity: by querying carefully chosen *sets* of buckets, it is possible to reduce the total number of queries required by the algorithm. This is just what Fischer et al. [13] did, and as a result they introduced testers for k-juntas with query complexity as low as $\tilde{O}(k^2/\epsilon)$.

The junta-testing algorithms of Fischer et al. remained the most query-efficient ways to test juntas until very recently, when the current author introduced an algorithm for testing boolean functions for the property of being k-juntas with $\tilde{O}(k^{3/2})/\epsilon$ queries [3]. This algorithm again used the independence test and the partitioning idea, but obtained a lower query complexity by combining two distinct strategies: one designed to reject functions with $k+1$ variables with high influence, and a separate one to reject functions with many variables with low but non-negligeable influence.

3.4 Lower bounds

In conjunction with their algorithms for testing k-juntas, Fischer et al. also introduced the first non-trivial lower bound on the query complexity of the junta testing problem. Specifically, they showed that for $k = o(\sqrt{n})$, non-adative algorithms for testing k-juntas must make at least $\tilde{\Omega}(\sqrt{k})$ queries [13]. This lower bound immediately implies a lower bound of $\Omega(\log k)$ queries for all adaptive k-junta testers as well.

Shortly afterwards, Chockler and Gutfreund [10] improved the lower bound and showed that $\Omega(k)$ queries are needed to test k-juntas (adaptively or non-adaptively). As we will see in the next section, this bound is nearly tight.

4 Testing k-juntas nearly optimally

4.1 Algorithm

The junta testing algorithm we saw in the previous section was based on the simple Independence Test; the algorithm we will describe in this section is based on a different simple but useful idea, due to Blum, Hellerstein, and Littlestone [7]: if we have two inputs $x, y \in \mathcal{X}$ such that $f(x) \neq f(y)$, then the set S of coordinates in which x and y disagree contains a coordinate that is relevant in f. Furthermore, by performing a binary search over the

hybrid inputs formed from x and y, we can identify the relevant coordinate with $O(\log |S|)$ queries.

Just as the algorithm in the last section combined the Independence Test with the idea of partitioning the variables to reduce the query complexity of the testing algorithm, we build on the previous observation by noting that if we have a partition of the coordinates into s parts and only care to identify a part that contains a relevant coordinate (rather than the coordinate itself), then we can optimize the binary search to only take $O(\log s)$ queries.

The k-junta testing algorithm uses the optimized binary search in the obvious way. It maintains a set S of coordinates that includes all the variables that are known to be relevant to the function, and partitions the coordinates into a large number of sets. It then generates pairs of inputs $x, y \in \mathcal{X}$ at random, and checks if $f(x) \neq f(x_S y_{\bar{S}})$. When a pair that satisfies the inequality is found, the algorithm uses a binary search to identify a set that contains a relevant coordinate, and adds all the coordinates in that set to S. If the algorithm identifies $k+1$ different sets with relevant coordinates, it rejects the function; otherwise, it accepts the function.

k-JUNTA TEST (f, ϵ)

Additional parameters: $s = O(k^9/\epsilon^5)$, $r = O(k \log k/\epsilon)$, $S \leftarrow \emptyset$

1. Randomly partition the coordinates in $[n]$ into s sets $I_1, \ldots, I_s$.
2. For each of r rounds,
   2.1. Generate a pair (x, y) at random such that $x_S = y_S$.
   2.2. If $f(x) \neq f(y)$, then do a binary search to identify a set I_j with a relevant variable and add I_j to S.
   2.3. If S contains $> k$ sets, quit and **reject**.
3. **Accept**.

Like the other two algorithms we have seen, it always accepts k-juntas, and once again the non-trivial part of the analysis of the algorithm is the proof that functions ϵ-far from k-juntas are rejected by the algorithm with high probability.

4.2 Analysis

The key to showing that functions ϵ-far from k-juntas are rejected with high probability is the following lemma:

Lemma 4.1. *Let $f : \{0,1\}^n \to \{0,1\}$ be ϵ-far from being a k-junta, and let $\mathcal{I}$ be a sufficiently fine partition of $[n]$. Then with high probability, for any set S formed by taking the union of at most k parts of $\mathcal{I}$, $\mathrm{Inf}_f(\overline{S}) \geq \epsilon/2$.*

Once the Lemma's proof is complete, the rest of the analysis of the algorithm is straightforward: by the Lemma, when f is ϵ-far from being a k-junta and at most k sets with relevant variables have been identified, the set of remaining variables has influence at least $\epsilon/2$. So each pair (x, y) sampled in step 2.1 will satisfy $f(x) \neq f(y)$ with probability at least $\epsilon/2$, and with high probability at least $k+1$ sets with relevant variables will be identified.

The proof of Lemma 4.1 can be completed with Fourier analysis. Alternatively, and more generally, it can also be completed using the *Efron-Stein decomposition* of functions [12]. This is the approach taken in [4], and it enables the analysis of the algorithm to hold even when the algorithm is testing non-boolean functions for the property of being k-juntas.[2]

5 Discussion and open problems

5.1 Closing the gap

The algorithm we saw in the last section and the lower bound of Chockler and Gutfreund [10] mentioned previously almost determine the query complexity of the problem of testing juntas: at least $\Omega(k)$ and at most $O(k \log k + k/\epsilon)$ queries are required to ϵ-test k-juntas.

The obvious question that remains open is to close the gap completely. Besides being an interesting question in its own right, closing the gap could also have some implications with respect to property testing testing with *quantum* algorithms: Atıcı and Servedio have shown that quantum algorithms can ϵ-test k-juntas with $O(k/\epsilon)$ queries [1], so an improvement of the lower bound would establish a gap between the query complexity of testing juntas with classical and quantum algorithms.

5.2 Adaptive vs. non-adaptive testing

Gonen and Ron [15], and Goldreich and Ron [14] recently began a systematic study of the benefits of adaptivity for testing properties in the dense-graph model. Interestingly, they showed that for some properties, there is a gap between the query complexity of the best adaptive and non-adaptive testing algorithms, while for other properties no such gap exists.

The current gap between query complexity of the best adaptive and non-adaptive algorithms for testing k-juntas ($O(k \log k + k/\epsilon)$ and $\tilde{O}(k^{3/2}/\epsilon)$, respectively) leaves an interesting problem open: does adaptivity help when testing k-juntas?

5.3 Tolerant testing and testing by implicit learning

One last problem that is quite interesting is the question of determining the optimal query complexity of a *tolerant* junta tester. Given $0 \leq \epsilon_1 < \epsilon_2$, an (ϵ_1, ϵ_2)-tolerant k-junta tester

[2] We note that the result in [4] was not the first one to generalize the analysis of a junta testing algorithm to non-boolean functions; Diakonikolas et al. [11] did so as well with a more technically intricate argument.

is a tester that, with high probability, accepts functions that are ϵ_1-close to being k-juntas and rejects functions that are ϵ_2-far from being k-juntas.

One very compelling reason for examining tolerant junta testers is the work of Diakonikolas, Lee, Matulef, Onak, Rubinfeld, Servedio, and Wan on *testing by implicit learning* [11]. They showed that a tolerant junta tester can be used as a basic building block to obtain efficient testing algorithms for a large class of properties related to the complexity of functions; improvements on the query complexity of tolerant junta tester could potentially yield improved query complexities on many of those properties as well.

References

[1] Alp Atıcı and Rocco A. Servedio. Quantum algorithms for learning and testing juntas. *Quantum Information Processing*, 6(5):323–348, 2007.

[2] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, PCPs and non-approximability – towards tight results. *SIAM J. Comput.*, 27(3):804–915, 1998.

[3] Eric Blais. Improved bounds for testing juntas. In *Proc. 12th Workshop RANDOM*, pages 317–330, 2008.

[4] Eric Blais. Testing juntas nearly optimally. In *Proc. 41st Symposium on Theory of computing*, pages 151–158, 2009.

[5] Avrim Blum. Relevant examples and relevant features: thoughts from computational learning theory. In *AAAI Fall Symposium on 'Relevance'*, 1994.

[6] Avrim Blum. Learning a function of r relevant variables. In *Proc. 16th Conference on Computational Learning Theory*, pages 731–733, 2003.

[7] Avrim Blum, Lisa Hellerstein, and Nick Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. *J. of Comp. Syst. Sci.*, 50(1):32–40, 1995.

[8] Avrim Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(2):245–271, 1997.

[9] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

[10] Hana Chockler and Dan Gutfreund. A lower bound for testing juntas. *Information Processing Letters*, 90(6):301–305, 2004.

[11] Ilias Diakonikolas, Homin K. Lee, Kevin Matulef, Krzysztof Onak, Ronitt Rubinfeld, Rocco A. Servedio, and Andrew Wan. Testing for concise representations. In *Proc. 48th Symposium on Foundations of Computer Science*, pages 549–558, 2007.

[12] Brad Efron and Charles Stein. The jackknife estimate of variance. *Ann. of Stat.*, 9(3):586–596, 1981.

[13] Eldar Fischer, Guy Kindler, Dana Ron, Shmuel Safra, and Alex Samorodnitsky. Testing juntas. *J. Comput. Syst. Sci.*, 68(4):753–787, 2004.

[14] Oded Goldreich and Dana Ron. Algorithmic aspects of property testing in the dense graphs model. In *Proc. 12th Workshop RANDOM*, pages 520–533, 2009.

[15] Mira Gonen and Dana Ron. On the benefits of adaptivity in property testing of dense graphs. In *Proc. 11th Workshop RANDOM*, pages 525–539, 2007.

[16] David Guijarro, Jun Tarui, and Tatsuie Tsukiji. Finding relevant variables in PAC model with membership queries. In *Proc. 10th Conference on Algorithmic Learning Theory*, pages 313–322, 1999.

[17] Richard J. Lipton, Evangelos Markakis, Aranyak Mehta, and Nisheeth K. Vishnoi. On the Fourier spectrum of symmetric boolean functions with applications to learning symmetric juntas. In *Proc. 20th Conference on Computational Complexity*, pages 112–119, 2005.

[18] Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning functions of k relevant variables. *J. Comput. Syst. Sci.*, 69(3):421–434, 2004.

[19] Michal Parnas, Dana Ron, and Alex Samorodnitsky. Testing basic boolean formulae. *SIAM J. Discret. Math.*, 16(1):20–46, 2003.

[20] Ronitt Rubinfeld and Madhu Sudan. Self-testing polynomial functions efficiently and over rational domains. In *Proc. 3rd Symposium on Discrete Algorithms*, pages 23–32, 1992.