Testing Submodularity

by

Venkata Abhinav Bommireddi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2017

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We show that for any constants $\epsilon > 0$ and $p \geq 1$, given oracle access to an unknown function $f : \{0,1\}^n \to [0,1]$ it is possible to determine if the function is submodular or is ϵ-far from every submodular function, in ℓ_p distance, with a *constant* number of queries to the oracle. We refer to the process of determining if an unknown function has a property, or is far from every function having the property, as *property testing*, and we refer to the algorithm that does that as a tester or a testing algorithm.

A function $f : \{0,1\}^n \to [0,1]$ is a *k-junta* if there is a set $J \subseteq [n]$ of cardinality $|J| \leq k$ such that the value of f on any input x is completely determined by the values x_i for $i \in J$. For any constant $\epsilon > 0$ and a set of k-juntas $\mathcal{F}$, we give an algorithm which determines if an unknown function $f : \{0,1\}^n \to [0,1]$ is $\frac{\epsilon}{10^6}$-close to some function in $\mathcal{F}$ or is ϵ-far from every function in $\mathcal{F}$, in ℓ_2 distance, with a constant number of queries to the unknown function. This result, combined with a recent junta theorem of Feldman and Vondrák (2016) in which they show every submodular function is ϵ-close, in ℓ_2 distance, to another submodular function which is a $\tilde{O}(\frac{1}{\epsilon^2})$-junta, yields the constant-query testing algorithm for submodular functions.

We also give constant-query testing algorithms for a variety of other natural properties of valuation functions, including fractionally additive (XOS) functions, OXS functions, unit demand functions, coverage functions, and self-bounding functions.

Acknowledgements

I would like to thank my advisor Eric Blais for being very supportive and guiding me through my masters. I would also like to thank Lap Chi Lau and Yaoliang Yu for being on my thesis reading committee.

I grateful to my parents and my brother for their unconditional love and support. I would also like to thank my friends, who were my family away from home and made my stay at Waterloo memorable. Finally, I thank "Candy" for keeping me motivated through the writing process.

Dedication

This thesis is dedicated to my grandfather Gudapati Seetharama Swamy.

Table of Contents

Chapter 1

Introduction

In this chapter we introduce the notion of property testing and define what submodular functions are and then state the theorems we prove in the thesis.

1.1 Property testing

Property testing informally can be thought of as a 20 questions game being played between two players Alice and Bob. Alice has an entity which is unknown to Bob and Bob has to figure out if the unknown entity has a property or not by asking Alice questions. There are restrictions on the kind of questions Bob is allowed to ask, if he is allowed to ask any kind of question then he can obviously ask if the entity has that property or not. The lesser the restrictions on the kind of questions Bob can ask the easier it is for him. Bob's objective is to minimize the number of questions he asks Alice. To make it slightly easier for Bob we give him certain promises on the unknown entity and he has to guess if the unknown entity has a certain property or not correctly only 2 out of 3 times.

A simple example is as follows: Alice has a box of 100 numbered balls, each ball is either black or white, Bob has to figure out if the box has all black balls or not. The only kind of question Bob is allowed to ask is what is the colour of ith ball for $1 \leq i \leq 100$. One way to decide if the box has all black balls or not is by asking the colour of the ball for all 100 balls, which takes 100 questions. It is easy to see that if Bob has to answer correctly always then he will have to ask 100 questions, as for any order of questions the last ball could be a white ball. In a sense this example is hard for Bob. One could ask the question that can Bob do better if he has to answer correctly only 2/3 times. In this setting 67

questions are enough. Bob could pick 67 out of the 100 positions uniformly at random and even if there is one white ball with probability $\frac{67}{100} \geq \frac{2}{3}$ he will catch it. Like always we ask the question can Bob do better? It can be shown that Bob can not do better than 67 questions in this case. But let's say Bob is given a promise that the box either has zero white balls or has at least ten percent white balls, nothing in between. With this extra promise it can be shown that Bob just needs 11 questions to answer correctly 2/3 times. This is essentially what property testing is. We try to figure out if a black box entity has a property by querying the black box and the objective is to minimize the queries and we trade off accuracy for fewer queries. This process of randomly approximately determining if an entity has a property is what property testing is about. People also look at the time required to test, but for this work we stick to queries.

In the above problem the promise that the box has zero white balls or at lest ten percent white balls is very powerful. It could seem that reducing the queries from 67 to 11 is not very significant. If we look closely we observe that if the box initially had 1,000,000 balls then the 67 becomes 670,000 but the 11 still remains the same. So the number of queries required with the promise is independent of the initial number of balls in the box, which is extremely useful when the initial number of balls is large.

In property testing we study what kind of promise makes it easy to determine if the given entity has a certain property. And we try to get a relation between the strength of the promise and the number of queries required. We also study what kind of properties are easy to test? For example if we are given 100 bits, we don't know what the value of the bits is but we are allowed to query and we are to determine if xor of the bits is 0 or 1 correctly with probability 2/3. In this case the best we could do is to query all 100 bits, and any reasonable promise will not help us improve the number of queries. We could say that this property is hard to test. Why is this property hard to test while the previous one was not so hard? This is essentially the kind of questions we study in property testing.

The idea of understanding the underlying structure of things is mathematically very appealing. In the last few years property testing as a field has been growing rapidly due to the growth in data. Due to the exponential growth in data people are trying to get faster algorithms at the expense of accuracy. People are no longer happy with linear time/query algorithms, they are trying to get *sublinear* algorithms, property testing is a subfield of sublinear algorithms. The focus is shifting to randomized algorithms and approximation algorithms. The first example we presented illustrates the power of randomization and approximation. If the number of balls is 2^{100} then it takes forever for a computer to determine if all the balls are black or not, but with randomization and the extra promise we just need 11 queries. Note that the 2/3 in the above algorithm is not a special number, any number greater than $\frac{1}{2}$ will do. Just by repeating the 2/3 algorithm 100 times and

taking the majority will boost the success probability to 0.99.

To formalize things, the black-box entities we deal with in this work are real valued functions over the boolean hypercube, $f : \{0,1\}^n \to [0,1]$. Let $\mathcal{F}_n$ denote the set of functions mapping $\{0,1\}^n$ to $[0,1]$.

The promise we talked about in the previous paragraphs is given in the form of distance to the property. The promise is that the function has a property or is far from having that property. There are different ways to define the distance between two functions, two most common ones are the Hamming distance and the ℓ_p distance. The definition of distance could also be extended to between a function and a set of functions. Throughout the thesis unless specified all the probabilities and expectations are over the uniform distribution.

Definition 1.1. The *Hamming distance* between two functions $f, g \in \mathcal{F}_n$ is

$$\mathrm{dist}_{\mathrm{Ham}}(f,g) = \Pr_x[f(x) \neq g(x)]$$

and the Hamming distance between f and a set of functions $\mathcal{G} \subset \mathcal{F}_n$ is

$$\mathrm{dist}_{\mathrm{Ham}}(f,\mathcal{G}) = \inf_{g \in \mathcal{G}} \mathrm{dist}_{\mathrm{Ham}}(f,g).$$

Definition 1.2. [47] [4] For $p \geq 1$, the ℓ_p *distance* between two functions $f, g \in \mathcal{F}_n$ is

$$\mathrm{dist}_p(f,g) = \|f-g\|_p = \left(\mathop{\mathrm{E}}_x[|f(x)-g(x)|^p]\right)^{1/p}$$

and the ℓ_p distance between f and a set of functions $\mathcal{G} \subset \mathcal{F}_n$ is

$$\mathrm{dist}_p(f,\mathcal{P}) = \inf_{g \in \mathcal{P}} \mathrm{dist}_p(f,g).$$

A *property* $\mathcal{P}$ of functions in $\mathcal{F}_n$ is a subset of these functions that is invariant under relabeling of the n coordinates. Formally property testing is defined as follows.

Definition 1.3. Given $\epsilon > 0$, An ϵ-*tester* in the Hamming testing model (resp., ℓ_p testing model) for some property $\mathcal{P} \subseteq \mathcal{F}_n$ is a randomized algorithm that

- accepts every function $f \in \mathcal{P}$ with probability at least $\frac{2}{3}$; and
- rejects every function f that satisfies $\mathrm{dist}_{\mathrm{Ham}}(f,\mathcal{P}) \geq \epsilon$ (resp., $\mathrm{dist}_p(f,\mathcal{P}) \geq \epsilon$) with probability at least $\frac{2}{3}$.

A tolerant tester is the generalization of a normal tester, where we have to accept functions that are close to the property in addition to functions that are in the property. The formal definition is below.

Definition 1.4. Given $\epsilon > \epsilon' > 0$, An (ϵ', ϵ)-*tolerant tester* in the Hamming testing model (resp., ℓ_p testing model) for some property $\mathcal{P} \subseteq \mathcal{F}_n$ is a randomized algorithm that

- accepts every function f that satisfies $\text{dist}_{\text{Ham}}(f, \mathcal{P}) \leq \epsilon'$ (resp., $\text{dist}_p(f, \mathcal{P}) \leq \epsilon'$) with probability at least $\frac{2}{3}$; and
- rejects every function f that satisfies $\text{dist}_{\text{Ham}}(f, \mathcal{P}) \geq \epsilon$ (resp., $\text{dist}_p(f, \mathcal{P}) \geq \epsilon$) with probability at least $\frac{2}{3}$.

An ϵ-tester is a special case of an (ϵ', ϵ)-tolerant tester with $\epsilon' = 0$.

1.2 Submodular functions

Submodular functions are a very widely studied class of functions which have applications in fields like optimization, approximation algorithms, game theory, machine learning. They can be informally defined as set functions which have the property of diminishing returns i.e. the increase in function value when you add an element to a set A, is smaller than when you add the same element to a subset of A. Lots of real world functions can be modeled as a submodular function. A very informal example is that happiness can be modeled as a submodular function. For example, consider the following two scenarios 1) You have 1000$ and you got a 1000$ scholarship 2) You have 100,000$ and you got a 1000$ scholarship. The increase in your happiness would be more in the first case than the second. The precise definition of submodular functions is below.

Definition 1.5. A function, $f : \{0,1\}^n \to [0,1]$, is submodular if $f(x) + f(y) \geq f(x \wedge y) + f(x \vee y)$ for every $x, y \in \{0,1\}^n$, where $\wedge$ and $\vee$ are the bitwise AND and OR operations;

Equivalently, a function $f : 2^{[n]} \to [0,1]$, is submodular if for any $i \in [n] := \{1, \cdots, n\}$, $A \subseteq [n] \setminus \{i\}, B \subset A$, $f(B \cup \{i\}) - f(B) \geq f(A \cup \{i\}) - f(A)$.

Note that there is a bijection between subsets on $[n]$ and binary strings of length n. A binary string $x \in \{0,1\}^n$ corresponds to the set $S = \{i : x_i = 1\}$, where x_i is the ith coordinate of x.

In game theory and economics [43] they assume people to have valuations towards goods and they term these as valuation functions. One of the natural properties which they assume the valuation functions to have is submodularity. They design algorithms based on the assumption that a certain valuation function is submodular, a natural question to ask is, is the given function actually submodular or not. Lots of these algorithms are robust, so works fine even in the cases where the function is close to submodular. The case we have to worry about is when the unknown valuation function is far from submodular. In this thesis we address the question of how many times do we need to query the unknown valuation function before we determine if it is submodular or far from being submodular.

1.3 Testing submodularity

In this thesis we show that submodularity can be tested using constant number of queries in the ℓ_p distance model. The question of testing properties of real valued functions is best considered in the ℓ_p *testing* framework introduced by Berman, Raskhodnikova, and Yaroslavtsev [4], ℓ_p distance model is a more natural setting to consider for testing real valued functions compared to the Hamming distance model. Let $f : \{0,1\}^n \to [0,1]$ be a real valued function and let f' be obtained from f by adding a very small noise to the output of f. The hamming distance between f and f' would be 1 even though the noise is very small and the outputs of f and f' are very close. We want a distance measure which says that f and f' are close and which increases with the increase in noise, ℓ_p distance measure has these desired properties. Below is the exact theorem we prove for testing submodularity.

Theorem 1.6. *For any $\epsilon > 0$ and any $p \geq 1$, there is an ϵ-tester for submodularity in the ℓ_p testing model with query complexity $2^{\tilde{O}(1/\epsilon^{\max\{2,p\}})}$.*

We are going to prove the above theorem by exploiting the fact that submodular functions are close to functions which are dependent on only a few variables, which was proved by Feldman and Vondrák in [30]. The functions which depend on only a few variables are called *juntas*. A function $f : \{0,1\}^n \to [0,1]$ is a *k-junta* if there is a set $J \subseteq [n]$ of cardinality $|J| \leq k$ such that the value of f on any input x is completely determined by the values x_i for each $i \in J$.

The precise statement of the junta theorem proved by Feldman and Vondrák is below.

Feldman–Vondrák junta theorem. *Fix any $\epsilon \in (0, \frac{1}{2})$. For every submodular function $f : \{0,1\}^n \to [0,1]$, there exists a submodular function $g : \{0,1\}^n \to [0,1]$ that is a $O(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon})$-junta such that $\|f - g\|_2 \leq \epsilon$.*

The fact that submodular functions are close to juntas combined with a general theorem on testing properties close to juntas which we prove in the thesis will yield Theorem 1.6 for $p = 2$ and that combined with the relationship between ℓ_2 and ℓ_p tester gives Theorem 1.6. The general theorem of testing properties close to juntas is stated below.

Theorem 1.7. *For any $0 < \epsilon < \frac{1}{2}$ and any property $\mathcal{P}$ of functions mapping $\{0,1\}^n \to [0,1]$, if $k \geq 1$ is such that for every function $f \in \mathcal{P}$, there is a k-junta h that satisfies $\|f - h\|_2 \leq \frac{\epsilon}{10^6}$, then there is an ϵ-tester for $\mathcal{P}$ in the ℓ_2 testing model with query complexity $\frac{2^{O(k \log k)}}{\epsilon^{10}}$.*

Theorem 1.7 is a consequence of a stronger theorem we prove about tolerant testing of junta properties which is stated below.

Theorem 1.8. *For any $0 < \epsilon < \frac{1}{2}$ and any property $\mathcal{F}$ of functions mapping $\{0,1\}^n \to [0,1]$, if $k \geq 1$ is such that every function $f \in \mathcal{F}$, is a k-junta , then there is an $(\frac{\epsilon}{10^6}, \epsilon)$-tolerant tester for $\mathcal{F}$ in the ℓ_2 testing model with query complexity $\frac{2^{O(k \log k)}}{\epsilon^{10}}$.*

This result is obtained by extending the technique of *testing by implicit learning* of Diakonikolas et al. [23]. The technique of testing by implicit learning is discussed at length in Chapter 4.

Theorem 1.7 also gives constant query testers for many other properties of valuation functions and we discuss those in Chapter 6. We first prove all our results in the ℓ_2 testing model and then extend it to general p using the relationships between different testing models which we discuss in section 6.2.

The outline of the thesis is as follows: In Chapter 2 we discuss some of the work done on testing, learning, approximating and optimizing submodular functions. In Chapter 3 we define a notion of importance of variables called influence and discuss some of the properties of influence and juntas which will be helpful in proving the main theorems. In Chapter 4 we discuss the general framework of testing by implicit learning and why it is a natural approach for achieving an efficient tester for submodularity. We also mention some of the limitations of the existing implicit learning algorithms which hinders them from achieving an efficient tester for submodularity. In Chapter 5 we give our implicit learning tester, that overcomes the limitations mentioned in the previous chapter, and analyze the algorithm to prove Theorem 1.8. In Chapter 6 we mention some of the other properties that can be tested by our tester and conclude by discussing some open problems.

The results in this thesis were published at ITCS '17 [7]

Chapter 2

Related work

In this chapter we discuss the previous work on testing submodularity and also discuss some of the work done in learning, approximating and optimizing submodular functions.

2.1 Testing submodularity

The question of testing submodularity was first raised by Parnas, Ron and Rubinfeld in [45]. They studied the question in two dimensions and gave a tester that could determine if a function $f : [n_1] \times [n_2] \to \mathbb{R}$ is submodular or ϵ-far(Hamming distance) from being submodular using $O(\frac{\log n_1 \log n_2}{\epsilon})$ queries. We say a function $f : [n_1] \times [n_2] \to \mathbb{R}$ is submodular if for every $0 < i < i^{'} \leq n_1$ and $0 < j < j^{'} \leq n_2$, $f(i^{'}, j^{'}) - f(i, j^{'}) \leq f(i^{'}, j) - f(i, j)$. They leave it as an open question if submodularity can be tested efficiently in higher dimensions.

Testing submodularity in higher dimensions was first studied by Seshadhri and Vondrák [49], they considered the Hamming testing model and gave a tester with query complexity $\epsilon^{-O(\sqrt{n}\log n)}$. They define a square on elements $i \in [n], j \in [n], A \subseteq [n] \setminus \{i, j\}$ as $f(A), f(A \cup \{i\}), f(A \cup \{j\})$ and $f(A \cup \{i, j\})$, if the function is submodular the square satisfies the property that $f(A \cup i) - f(A) \geq f(A \cup \{i, j\}) - f(A \cup j)$. Their algorithm is to pick squares uniformly at random and if it does not satisfy the property then output the function is not submodular. They prove this by proving that if the function is ϵ-far from being submodular then $\epsilon^{\sqrt{n}\log n}$ fraction of the squares violate the property.

On the other hand they also prove a lower bound of $\Omega(\sqrt{n})$ by giving a reduction from monotonicity to submodularity, from a function $f : \{0, 1\}^n \to \mathbb{R}$ they construct a function $g : \{0, 1\}^{n+1} \to \mathbb{R}$ that has the following properties

- If f is monotonically non-increasing then g is submodular
- If f if ϵ far from being monotone then g is $\frac{\epsilon}{2}$-far from being submodular.
- The value of g can be computed by looking at two values of f

With the help of this reduction and the $\Omega(\sqrt{n})$ lower bound for testing monotonicity they get the lower bound for submodularity. Blais et al. in [8] improve the lower bound for testing monotonicity to $\Omega(n)$ thereby improving the lower bound for testing submodularity to $\Omega(n)$. Later Hatami and Vondrák improved the lower bound to $\tilde{\Omega}(n^2)$ using the techniques in [8]. There is still a huge gap between the upper bound and the lower bound for testing submodularity in hamming setting. It is an interesting open question to bridge the gap. Note that Seshadhri and Vondrák's algorithm works for functions with unbounded range.

Testing submodularity in the ℓ_p setting was first studied by Feldman and Vondrák in [30], they consider functions on boolean hypercube with range $[0, 1]$. In the ℓ_1 testing model they give a tester with query complexity $\text{poly}(\frac{1}{\epsilon}) \log n + 2^{\tilde{O}(1/\epsilon)}$ which gives a tester in ℓ_2 testing model with query complexity $\text{poly}(\frac{1}{\epsilon}) \log n + 2^{\tilde{O}(1/\epsilon^2)}$. They get this result with the help of a technical result they prove in the same paper that submodular functions can be approximated by juntas. The first step of their testing algorithm involves learning the relevant variables.

Though query complexity of Feldman and Vondrák's tester is pretty good, it still has dependence on the dimension. Lot's of natural properties have been show to have testers with constant query complexity i.e. independent of the dimension. Is it possible to test submodularity with constant queries? It is not possible to achieve it using the technique of Feldman and Vondrák as a simple information theoretic argument shows that if you want to learn the relevant variables then you need at least $\log(n)$ queries. Our tester gets around the step of learning the variables and thereby getting rid of the $\log(n)$.

Submodular functions have also been studied in the contexts of approximation, learning, optimization. Though it is not exhaustive, we mention some of the works below.

2.2 Approximating and learning submodular functions

Before we talk about learning submodularity let's define the different learning models.

For this work we are only concerned with proper learning and when ever we say learning we mean proper learning. In learning(proper) you have an unknown function f belonging to some function class $\mathcal{F}$ and you get samples on that function from some unknown distribution $\mathcal{D}$. The learner has to process the samples and output a hypothesis function $h \in \mathcal{F}$ such that f and h are close with respect to the same unknown distribution $\mathcal{D}$. The different learning models we look at in this section differ in the definition of distance between functions.

Below is the definition of PAC learner in ℓ_1 distance model.

Definition 2.1. [50] Let $\mathcal{F}$ be a family of non-negative, real-valued functions with domain $\{0,1\}^n$. We say that an algorithm $\mathcal{A}$ *PAC-learns* $\mathcal{F}$ with error parameter $\epsilon \geq 0$ if for any distribution $\mathcal{D}$ over $\{0,1\}^n$ and for any target function $f \in \mathcal{F}$

- The input to $\mathcal{A}$ is a sequence of pairs $\{(x^{(i)}, f(x^{(i)}))\}_{1 \leq i \leq l}$ where each $x^{(i)}$ is chosen independently from distribution $\mathcal{D}$.
- The output of $\mathcal{A}$ is a function $h \in \mathcal{F}$ that satisfies $\Pr_{x^{(1)},\ldots,x^{(l)} \in D}[\mathbb{E}_{x \in \mathcal{D}}[|f(x) - h(x)|] \geq \epsilon] \leq \frac{1}{3}$

The efficiency of the learner is measured in terms of the number of samples, l, it needs.

Another learning model we consider is the $PMAC$ learning model. Here we allow the hypothesis function to be arbitrarily far from the target function on a small fraction of the input. On rest of the inputs f and h are at most an α factor apart, α need not be a constant. Below is the definition of a $PMAC$ learner.

Definition 2.2. [3] Let $\mathcal{F}$ be a family of non-negative, real-valued functions with domain $\{0,1\}^n$. We say that an algorithm $\mathcal{A}$ *PMAC-learns* $\mathcal{F}$ with approximation factor α if, for any distribution $\mathcal{D}$ over $\{0,1\}^n$, for any target function $f \in \mathcal{F}$, and for $\epsilon \geq 0$ and $\delta \geq 0$ sufficiently small:

- The input to $\mathcal{A}$ is a sequence of pairs $\{(x^{(i)}, f(x^{(i)}))\}_{1 \leq i \leq l}$ where each $x^{(i)}$ is chosen independently from distribution $\mathcal{D}$.
- The output of $\mathcal{A}$ is a function $h \in \mathcal{F}$ that satisfies $\Pr_{x^{(1)},\ldots,x^{(l)} \in D}[\Pr_{x \in \mathcal{D}}[h(x) \leq f(x) \leq \alpha h(x)] \geq 1 - \epsilon] \geq 1 - \delta$

There has been quite a bit of work done on learning and approximating submodular functions. Goemans et al. [34] studied approximating submodular functions and showed

that given oracle access to a non-negative, monotone submodular function, using poly(n) queries you can get a hypothesis function which approximates the initial submodular function within a factor of $\tilde{O}(\sqrt{n})$. Balcan and Harvey in [3] studied learning non-negative, monotone submodular functions in the PMAC setting. They show that under additional assumptions that the function is Lipschitz and the distribution is a product distribution you can PMAC learn with a constant approximation factor and for arbitrary distributions they give a PMAC learning algorithm with an approximation factor $O(\sqrt{n})$ and also showed that any PMAC learning algorithm with poly(n) sample complexity will have an approximation factor $\tilde{\Omega}(n^{\frac{1}{3}})$. For uniform distribution Feldman and Vondrák in [29] give a PMAC learning algorithm with constant approximation factor, this algorithm works even for non-monotone submodular functions. In the same paper they also give an algorithm for PAC learning of submodular functions, in ℓ_1 distance setting, with sample complexity $\text{poly}(\frac{1}{\epsilon}) \log n + 2^{\tilde{O}(1/\epsilon^2)}$. Except for the PAC learning algorithm of Feldman and Vondrák, rest of the algorithms in this section work for functions with unbounded range. The PAC learning algorithm assumes the range is $[0, 1]$.

2.3 Optimizing submodular functions

Submodular functions have been studied in the context of optimization too. Submodular function minimization(SFM) has a rich history and it was shown to have a polynomial time algorithm by Grötschel et al., in [37]. They give an ellipsoid algorithm to minimize submodular functions. The first combinatorial algorithm for SFM was given by Cunningham in [22] after that there has been a line of work which gave better combinatorial algorithms for SFM. And recently Lee at al. in [40] showed that ellipsoid method(cutting plane method) can be used to achieve faster algorithms too and they improved the then best running times by a factor of $O(n^2)$. In [18] Chakrabarty et al. give the first subquadratic function minimization algorithm using stochastic projected gradient descent.

Submodular function maximization is also well studied and it has been shown to be NP-hard. But there are constant factor approximation algorithms known. The current best is a deterministic approximation algorithm with approximation factor $\frac{1}{2}$ [14][15]. This algorithm does not assume anything about the submodular function.

For approximating, learning, optimizing submodular functions the algorithm assumes access to a submodular oracle. Using the testing algorithms for submodularity we can quickly check if the given oracle is actually submodular or not.

Chapter 3

Juntas and influence

In this chapter we define a notion of importance of variables called *influence* and discuss various properties of juntas and influence which will help us in proving Theorem 1.8. At the end we briefly mention the background work on testing k-juntas and also discuss about properties that can be approximated by juntas.

3.1 Juntas

We refer to a function $f : \{0,1\}^n \to \mathbb{R}$ as a k-junta if the output of the function is dependent on at most k of the n variables. They can be formally defined as follows:

Definition 3.1. The function $f : \{0,1\}^n \to [0,1]$ is a *junta on the set* $J \subseteq [n]$ if for every $x, y \in \{0,1\}^n$ that satisfy $x_i = y_i$ for every $i \in J$, we have $f(x) = f(y)$, where x_i is the ith coordinate of x. The function f is a *k-junta* if it is a junta on some set $J \subseteq [n]$ of cardinality $|J| \leq k$.

For any $f \in \mathcal{F}_n$ and $S \subseteq [n]$ with complement $\overline{S} = [n] \setminus S$, when $x \in \{0,1\}^S$ and $y \in \{0,1\}^{\overline{S}}$, we write $f(x,y)$ to denote the value $f(z)$ for the input z that satisfies $z_i = x_i$ for each $i \in S$ and $z_i = y_i$ otherwise.

Throughout the thesis, unless otherwise specified all probabilities and expectations are over the uniform distribution on the random variable's domain.

Definition 3.2. For any function $f : \{0,1\}^n \to [0,1]$ and set $J \subseteq [n]$, the *J-junta projection* of f is the function $f_J : \{0,1\}^J \to [0,1]$ defined by setting

$$f_J(x) = \mathop{\mathrm{E}}_{y \in \{0,1\}^{\overline{J}}}[f(x,y)]$$

for every $x \in \{0,1\}^J$.

The function $f_{\text{core}} : \{0,1\}^k \to [0,1]$ is a *core function* of the k-junta $f : \{0,1\}^n \to [0,1]$ if there is a projection $\psi : \{0,1\}^n \to \{0,1\}^k$ defined by setting $\psi(x) = (x_{i_1}, \ldots, x_{i_k})$ for some distinct $i_1, \ldots, i_k \in [n]$ such that for every $x \in \{0,1\}^n$, $f(x) = f_{\text{core}}\big(\psi(x)\big)$. The core function essentially captures the structure of the function and does not care what the labeling of the variables is.

A basic fact that we will require is that f_J is the J-junta that is closest to f under the ℓ_2 metric.

Proposition 3.3. *For every $f : \{0,1\}^n \to [0,1]$ and $J \subseteq [n]$, if $g : \{0,1\}^n \to [0,1]$ is a J-junta, then* $\text{dist}_2(f, f_J) \leq \text{dist}_2(f, g)$.

Proof. By applying the identity $\|f - g\|_2^2 = \|f - f_J + f_J - g\|_2^2$ and by expanding the right-hand side, we obtain

$$\begin{aligned}\|f - g\|_2^2 &= \mathop{\mathrm{E}}_{x \in \{0,1\}^J}\left[\mathop{\mathrm{E}}_{y \in \{0,1\}^{\overline{J}}}\left[\big(f(x,y) - f_J(x,y) + f_J(x,y) - g(x,y)\big)^2\right]\right] \\ &= \|f - f_J\|_2^2 + \|f_J - g\|_2^2 + 2\mathop{\mathrm{E}}_{x}\left[\mathop{\mathrm{E}}_{y}\left[\big(f(x,y) - f_J(x,y)\big)\big(f_J(x,y) - g(x,y)\big)\right]\right].\end{aligned}$$

Since $f_J - g$ is a J-junta, it does not depend on y and, by the definition of f_J, the last term equals 0. Therefore, $\|f - g\|_2^2 = \|f - f_J\|_2^2 + \|f_J - g\|_2^2$ and the proposition follows. □

The property $\mathcal{P} \subseteq \mathcal{F}_n$ is a *property of k-juntas* if every function $f \in \mathcal{P}$ is a k-junta. The *core property* of a property $\mathcal{P}$ of k-juntas is the property $\mathcal{P}_{\text{core}} \subseteq \mathcal{F}_k$ defined by $\mathcal{P}_{\text{core}} = \{f_{\text{core}} : f \in \mathcal{P}\}$. For any $\gamma > 0$, the *γ-discretized approximation* of a function $f \in \mathcal{F}_n$ is the function $f^{(\gamma)}$ obtained by rounding the value $f(x)$ for each $x \in \{0,1\}^n$ to the nearest multiple of γ. The γ-discretized approximation of a property $\mathcal{P}$ is the property $\mathcal{P}^{(\gamma)} = \{f^{(\gamma)} : f \in \mathcal{P}\}$.

3.2 Influence

In this section we define influence and also discuss some of its properties.

Before we define influence we state the most important theorem about boolean functions, the fourier expansion theorem.

Theorem 3.4. *([44]) For every function $f : \{0,1\}^n \to \mathbb{R}$, there exists $\hat{f}(S) \in \mathbb{R}$ for each $S \subseteq [n]$ such that for any $x \in \{0,1\}^n$*

$$f(x) = \sum_{S \subseteq [n]} \hat{f}(S)(-1)^{\sum_{i \in S} x_i}$$

This expression is called the Fourier expansion of f, and the real number $\hat{f}(S)$ is called the Fourier coefficient of f on S. Collectively, the coefficients are called the Fourier spectrum of f. For the proof of the theorem and other properties of boolean functions see for example [44].

The notion of variance will be helpful in defining influence. Variance of a function, like expectation, gives a lot of information about the function. This information can be used to determine the importance of a set of variables.

Definition 3.5. For any function $f : \{0,1\}^n \to \mathbb{R}$, the variance of the function is defined as

$$\operatorname*{Var}_{x \in \{0,1\}^n} f(x) = \operatorname*{E}_{x \in \{0,1\}^n} \left[(f(x) - \operatorname*{E}_{x \in \{0,1\}^n} [f(x)])^2 \right] = \frac{1}{2} \operatorname*{E}_{x,x' \in \{0,1\}^n} \left[\left(f(x) - f(x') \right)^2 \right]$$

There are different formulae for variance which are equivalent, for this work we use the above two versions. We switch from one formula to the other based on convenience. By expanding both the formulae we can see that both evaluate to $\mathrm{E}_{x \in \{0,1\}^n} \left[(f(x))^2 \right] - \mathrm{E}_{x \in \{0,1\}^n} \left[f(x) \right]^2$ which is another formula for variance one comes across.

The notion of influence of coordinates in functions over the Boolean hypercube plays a central role in both our algorithm and its analysis. Informally, the influence of a set of coordinates measures how much re-randomizing these coordinates affects the value of the function. This notion is made precise as follows.

Definition 3.6. The *influence* of a set $S \subseteq [n]$ of coordinates in the function $f : \{0,1\}^n \to [0,1]$ is

$$\mathrm{Inf}_f(S) := \operatorname*{E}_{x \in \{0,1\}^{\overline{S}}} \left[\operatorname*{Var}_{y \in \{0,1\}^S} f(x,y) \right] = \tfrac{1}{2} \operatorname*{E}_{x \in \{0,1\}^{\overline{S}}} \left[\operatorname*{E}_{y,y' \in \{0,1\}^S} \left[\left(f(x,y) - f(x,y') \right)^2 \right] \right].$$

The second equality in the above definition follows from the formula of variance in Definition 3.5. This representation of influence will help us later in the analysis.

3.2.1 Properties of influence

A standard fact that will be useful in our proof is that the influence of S in f has a natural representation in terms of the Fourier coefficients of f.

Proposition 3.7. *The influence of $S \subseteq [n]$ in $f \in \mathcal{F}_n$ is $\mathrm{Inf}_f(S) = \sum_{T:T\cap S\neq\emptyset} \hat{f}(T)^2$.*

Proof. Applying the Fourier decomposition of f,

$$\mathop{\mathrm{E}}_{y\in\{0,1\}^S}\big[f(x,y)\big] = \sum_{T\subseteq\overline{S}} \hat{f}(T)(-1)^{\sum_{i\in T} x_i}$$

and the influence evaluates to

$$\begin{aligned}
\mathrm{Inf}_f(S) &= \mathop{\mathrm{E}}_{x\in\{0,1\}^{\overline{S}}}\Big[\mathop{\mathrm{Var}}_{y\in\{0,1\}^S} f(x,y)\Big] \\
&= \mathop{\mathrm{E}}_{x\in\{0,1\}^{\overline{S}}}\Big[\mathop{\mathrm{E}}_{y\in\{0,1\}^S}[(f(x,y) - \mathop{\mathrm{E}}_{y\in\{0,1\}^S}[f(x,y)])^2]\Big] \\
&= \mathop{\mathrm{E}}_{z\in\{0,1\}^n}\Big[(\sum_{T:T\cap S\neq\emptyset} \hat{f}(T)(-1)^{\sum_{i\in T} z_i})^2\Big] \qquad \text{by using Fourier expansion of } f \\
&= \sum_{T:T\cap S\neq\emptyset} \hat{f}^2(T). \qquad \square
\end{aligned}$$

This representation also immediately implies that influence is monotone and subadditive.

Proposition 3.8. *For every $f \in \mathcal{F}_n$ and $S, T \subseteq [n]$, we have $\mathrm{Inf}_f(S) \leq \mathrm{Inf}_f(S\cup T) \leq \mathrm{Inf}_f(S) + \mathrm{Inf}_f(T)$.*

Proof. The proof follows from Proposition 3.7.

$$\begin{aligned}
\mathrm{Inf}_f(S) + \mathrm{Inf}_f(T) &= \sum_{M:M\cap S\neq\emptyset} \hat{f}^2(M) + \sum_{M:M\cap T\neq\emptyset} \hat{f}^2(M) \\
&\geq \sum_{M:M\cap(S\cup T)\neq\emptyset} \hat{f}^2(M) = \mathrm{Inf}_f(S\cup T) \\
&\geq \sum_{M:M\cap S\neq\emptyset} \hat{f}^2(M) = \mathrm{Inf}_f(S) \qquad \square
\end{aligned}$$

Another critical fact about the influence of a set of coordinates is that it corresponds to the ℓ_2 distance of the function to the junta on its complement.

Proposition 3.9. *For every $f \in \mathcal{F}_n$ and $J \subseteq [n]$, we have $\mathrm{Inf}_f(\overline{J}) = \mathrm{dist}_2(f, f_J)^2$.*

Proof. The proof follows from elementary operations.

$$\begin{aligned}
\mathrm{Inf}_f(\overline{J}) &= \mathop{\mathrm{E}}_{x\in\{0,1\}^J}\Big[\mathop{\mathrm{Var}}_{y\in\{0,1\}^{\overline{J}}} f(x,y)\Big] \\
&= \mathop{\mathrm{E}}_{x\in\{0,1\}^J}\Big[\mathop{\mathrm{E}}_{y\in\{0,1\}^{\overline{J}}}[(f(x,y) - \mathop{\mathrm{E}}_{y\in\{0,1\}^{\overline{J}}}[f(x,y)])^2]\Big] \\
&= \mathop{\mathrm{E}}_{x\in\{0,1\}^J}\Big[\mathop{\mathrm{E}}_{y\in\{0,1\}^{\overline{J}}}[(f(x,y) - f_J(x,y))^2]\Big] \\
&= \mathop{\mathrm{E}}_{z\in\{0,1\}^n}\big[(f(z) - f_J(z))^2\big] \\
&= \mathrm{dist}_2(f, f_J)^2
\end{aligned}$$

□

This fact can be used to show that when two functions f and g are close in the ℓ_2 metric, the influence of any set $S \subseteq [n]$ is almost the same in both f and g.

Proposition 3.10. *Fix $\epsilon > 0$, and let $f, g : \{0,1\}^n \to [0,1]$ satisfy $\mathrm{dist}_2(f,g) \leq \epsilon$. Then for any set $S \subseteq [n]$, $|\mathrm{Inf}_f(S)^{\frac{1}{2}} - \mathrm{Inf}_g(S)^{\frac{1}{2}}| \leq \epsilon$.*

Proof. By Proposition 3.9, we have $\mathrm{Inf}_f(S)^{\frac{1}{2}} = \|f - f_{\overline{S}}\|_2$ and $\mathrm{Inf}_g(S)^{\frac{1}{2}} = \|g - g_{\overline{S}}\|_2$. By Proposition 3.3, we also have that $\|f - f_{\overline{S}}\|_2 \leq \|f - g_{\overline{S}}\|_2$. Combining these observations with the triangle inequality, we obtain $\mathrm{Inf}_f(S)^{\frac{1}{2}} - \mathrm{Inf}_g(S)^{\frac{1}{2}} = \|f - f_{\overline{S}}\|_2 - \|g - g_{\overline{S}}\|_2 \leq \|f - g_{\overline{S}}\|_2 - \|g - g_{\overline{S}}\|_2 \leq \|f - g\|_2 \leq \epsilon$. Hence $\mathrm{Inf}_f(S)^{\frac{1}{2}} - \mathrm{Inf}_g(S)^{\frac{1}{2}} \leq \epsilon$ and, similarly, $\mathrm{Inf}_g(S)^{\frac{1}{2}} - \mathrm{Inf}_f(S)^{\frac{1}{2}} \leq \epsilon$ as well. □

Below is a concentration inequality which we use extensively in this work.

Hoeffding's inequality. *Let $X_1, ..., X_n$ be independent random variables bounded by $a_i \leq X_i \leq b_i$. Let $X = X_1 + X_2 + \cdots + X_n$ have expected value $E[X] = \mu$. Then for any $t > 0$,*

$$\Pr[|X - \mu| \geq t] \leq 2e^{-\frac{2t^2}{\sum_{i=1}^n (b_i - a_i)^2}}.$$

Throughout the algorithm we estimate the influence of variables many times and the algorithm for estimation of influence is given below.

Proposition 3.11. *There is an algorithm* EstimateInf *such that for every* $f : \{0,1\}^n \to [0,1]$, $S \subseteq [n]$, $m \geq 1$, *and* $t \geq 0$, *it makes* m *queries to* f *and returns an estimate of the influence of* S *in* f *that satisfies*

$$\Pr\left[|\mathrm{Inf}_f(S) - \textsc{EstimateInf}(f, S, m)| \geq t\right] \leq 2e^{-2mt^2}.$$

Proof. The proof of Proposition 3.11 is obtained by considering the EstimateInf algorithm below.

Algorithm 1: $\textsc{EstimateInf}(f, S, m)$

1 Draw $x^{(1)}, \ldots, x^{(m)}$ uniformly and independently at random from $\{0,1\}^{\overline{S}}$;
2 Draw $y^{(1)}, \ldots, y^{(m)}, y'^{(1)}, \ldots, y'^{(m)}$ uniformly and independently at random from $\{0,1\}^S$;
3 Return $\frac{1}{2m}\sum_{i=1}^{m}\left(f(x^{(i)}, y^{(i)}) - f(x^{(i)}, y'^{(i)})\right)^2$;

The concentration of the estimated influence is obtained via the standard version of Hoeffding's inequality. We get this because the expected value of the estimated influence is the influence, $\mathrm{E}_{x\in\{0,1\}^{\overline{S}}, y,y'\in\{0,1\}^S}\frac{1}{2m}\left[\sum_{i=1}^{m}\left(f(x^{(i)}, y^{(i)}) - f(x^{(i)}, y'^{(i)})\right)^2\right] = \mathrm{Inf}_f(S)$, by Definition 3.6. □

Below is a theorem on intersecting families of subsets which will be helpful in proving Lemma 3.13. Understanding the theorem statement requires the following definitions.

A family $\mathcal{G}$ of subsets of $[n]$ is t-intersecting if for every pair of sets $S, T \in \mathcal{G}$, their intersection size is at least $|S \cap T| \geq t$. For $0 < p < 1$, a *p-biased measure* for such a family is defined as $\mu_p(\mathcal{G}) := \Pr_J[J \in \mathcal{G}]$, where the probability over J is obtained by including each coordinate in J independently with probability p.

Theorem 3.12. *(Dinur and Safra[25]; Friedgut[33]) Let* $\mathcal{G}$ *be a* t*-intersecting family of subsets of* $[n]$ *for some* $t \geq 1$. *For any* $p < \frac{1}{t+1}$, *the* p*-biased measure of* $\mathcal{G}$ *is bounded by* $\mu_p(\mathcal{G}) \leq p^t$.

We also use the following key lemma from [10].

Lemma 3.13 (Lemma 2.3 in [10])**.** *Let* $f : \{0,1\}^n \to [0,1]$ *be a function that is* ϵ*-far in* ℓ_2 *metric from* k*-juntas and* P *be a random partition of* $[n]$ *into* $r > 20k^2$ *parts. Then with probability at least* $\frac{5}{6}$, $\mathrm{Inf}_f(\overline{J}) \geq \frac{\varepsilon^2}{4}$ *for any union* J *of* k *parts from* P.

Proof. For $0 \leq t \leq \frac{1}{2}$, let $\mathcal{G}_t = \{J \subseteq [n] : \mathrm{Inf}_f(\overline{J}) < t\varepsilon^2\}$ be the family of all the sets whose complements have influence less than $t\varepsilon^2$. For any two sets $J, K \in \mathcal{G}_{\frac{1}{2}}$, the sub-additivity of influence implies that

$$\mathrm{Inf}_f(\overline{J \cap K}) = \mathrm{Inf}_f(\overline{J} \cup \overline{K}) \leq \mathrm{Inf}_f(\overline{J}) + \mathrm{Inf}_f(\overline{K}) < \varepsilon^2.$$

But f is ε-far from every k-junta, so for any two sets $J, K \in \mathcal{G}_{\frac{1}{2}}$, $|J \cap K| > k$, from Proposition 3.10. Which means $\mathcal{G}_{\frac{1}{2}}$ is a $k+1$ intersecting family. There are two cases now, first one is, there is at least one set $J \in \mathcal{G}_{\frac{1}{2}}$ such that $|J| < 2k$, second one is all the sets $J \in \mathcal{G}_{\frac{1}{2}}$ will have $|J| \geq 2k$. We will show that in both the cases our lemma holds. In the first case let $J \in \mathcal{G}_{\frac{1}{2}}$ be a set which has fewer than $2k$ elements, with high probability the set J is completely separated by the partition $\mathcal{P}$, and we know that for any $K \in \mathcal{G}_{\frac{1}{2}}$, $|J \cap K| \geq k+1$, which means K is not covered by any union of k-parts in $\mathcal{P}$. Therefore, $\mathrm{Inf}_f(\overline{J}) \geq \frac{\varepsilon^2}{2} > \frac{\varepsilon^2}{4}$ as we wanted to show.

Consider the case where, all the sets in $\mathcal{G}_{\frac{1}{2}}$ have more than $2k$ elements. Then $G_{\frac{1}{4}}$ is a $2k$ intersecting family. Otherwise, if there are two sets $J, K \in \mathcal{G}_{\frac{1}{4}}$ such that $|J \cap K| < 2k$, then $\mathrm{Inf}_f(\overline{J \cap K}) \leq \mathrm{Inf}_f(\overline{J}) + \mathrm{Inf}_f(\overline{K}) < \frac{\varepsilon^2}{4} + \frac{\varepsilon^2}{4} < \frac{\varepsilon^2}{2}$, thus contradicting our assumption.

Let $J \subseteq [n]$ be the union of k parts in $\mathcal{P}$. Since $\mathcal{P}$ is a random partition, J is a random subset obtained by including each element of $[n]$ in J independently with probability $p = \frac{k}{r} < \frac{1}{2k+1}$. By Theorem 3.12, $\Pr_{\mathcal{P}}[I_f(\overline{J}) < \frac{\varepsilon^2}{4}] = \Pr[J \in \mathcal{G}_{\frac{1}{4}}] = \mu_{\frac{k}{r}}(\mathcal{G}_{\frac{1}{4}}) \leq (\frac{k}{r})^{2k}$. By the union bound the probability that there exists a set $J \subseteq [n]$ that is the union of k parts in $\mathcal{P}$ for which $\mathrm{Inf}_f(\overline{J}) < \frac{\varepsilon^2}{4}$ is bounded above by $\binom{r}{k}(\frac{k}{r})^{2k} \leq (\frac{er}{k})^k(\frac{k}{r})^{2k} \leq (\frac{ek}{r})^k < \frac{1}{6}$. □

3.3 Testing juntas

In this section we discuss the relevant work on testing juntas. Testing k-juntas is the problem of determining if a given function $f : \{0,1\}^n \to \{0,1\}$ is a k-junta or ϵ-far from every k-junta correctly with probability at least $2/3$. For this section the distance metric is Hamming distance unless mentioned otherwise.

The first known algorithm for testing juntas was obtained(implicitly) by combining an algorithm for learning juntas [11][38] with the technique of testing-by-learning [35] and had a query complexity of $O(k(\frac{\log k}{\epsilon} + \log n))$. The technique of testing-by-learning is discussed in Chapter 4. Testing juntas was first explicitly studied by Parnas et al. [46], they studied testing of 1-juntas and gave an algorithm with query complexity $O(\frac{1}{\epsilon})$. Later Fischer et

al. [31] studied the question of testing k-juntas, for $k \geq 1$ and gave a testing algorithm which requires $\tilde{O}(k^2/\epsilon)$ queries. This was the first constant query tester for k-juntas. Their algorithm and many algorithms for testing juntas use the following test as a subroutine.

Definition 3.14. Independence Test: Given a function f and set of variables S, choose $x \in \{0,1\}^{\overline{S}}$ and $y, y^{'} \in \{0,1\}^{S}$ uniformly at random. Accept if $f(x,y) = f(x,y^{'})$ and reject if $f(x,y) \neq f(x,y^{'})$.

Fischer et al.'s algorithms works as follows: partition the variables into $O(k^2)$ buckets uniformly at random and do independence test(few times) on blocks of k buckets picked at random, and if it passes the independence test then label all the buckets in the block as not-relevant. Do this for $O(k \log k)$ blocks, and at the end label all the unlabeled buckets as relevant. If at least half the blocks pass the independence test and less than k buckets are labeled relevant at the end of it then accept the function. In the same paper Fischer et al. also give a much simpler test where they partition the variables into $O(k^2)$ buckets uniformly at random and do independence test on each bucket a few times and if more than k buckets fail the independence test then reject it, else accept it. The query complexity of this algorithm is $\tilde{O}(k^4/\epsilon)$. The $\tilde{O}(k^2/\epsilon)$ upper bound on testing juntas was later improved by Eric Blais in [5] to $\tilde{O}(k^{\frac{3}{2}}/\epsilon)$ and further to $\tilde{O}(k/\epsilon)$ in [6]. The key insight in getting the query complexity down to linear in k was that when we partition the variables into buckets, if the original function was ϵ-far from every junta then if we pick any set of k buckets, the influence of the variables outside these k buckets is high. We use a similar lemma(Lemma 3.13) in the analysis of our algorithm too. Chockler and Gutfreund in [21] showed that testing of juntas requires at least $\Omega(k)$ queries, so tester in [6] is optimal up to a factor of $O(\log k)$. The algorithm in [6] would work for real valued functions too in Hamming setting, and Diakonikolas et al. [23] showed that Fischer et al.'s algorithms can be extended to work for real valued functions in Hamming setting.

Much less is known about the problem of tolerant testing of k-juntas. The above testing algorithms have slight tolerance. They accept functions that are poly$(\frac{\epsilon}{k})$-close to a k-junta. Chakraborty et al. [20] [19] observed that the testing algorithm of Eric Blais [6] implies a $(c\epsilon, \epsilon)$-tolerant tester for k-juntas with query complexity $\exp(k)$, for some constant $0 < c < 1$, this works for real valued functions too under Hamming setting. Subsequent to our work, Blais et al. in [9], give an algorithm that with probability $2/3$ accepts functions that are $\epsilon/16$-close to some k-junta, and with probability $2/3$ rejects functions that are ϵ-far from every $4k$ junta. The query complexity of this algorithm is poly(k) but it only works for boolean functions and it does not imply a $(c\epsilon, \epsilon)$-tolerant tester for k-juntas, for any constant $0 < c < 1$.

Due to the relationship between testing in Hamming setting and ℓ_p setting, the tester in [6] can be used to test k-juntas in ℓ_p setting with $\tilde{O}(k/\epsilon)$ queries. The same is not true for tolerant testing, a tolerant tester in Hamming setting can not be used as a tolerant tester in ℓ_p setting because two functions can be very close in ℓ_p distance and still be very far in Hamming distance, so none of the above mentioned tolerant testing algorithms imply a tolerant tester in ℓ_p setting. We are the first to study tolerant testing of k-juntas in the ℓ_p setting.

There has been some work on tolerant testing of properties of k-juntas too and it is discussed in detail in Chapter 4.

3.4 Approximation by juntas

Submodular functions are not the first natural class of functions that are shown to be close to juntas. There are theorems which say that certain kinds of functions can be approximated by juntas. Friedgut's junta theorem is one such theorem, which informally says functions with low influence can be approximated well by juntas. Bourgain's junta theorem is another such junta approximation theorem which says functions which are noise stable can be well approximated by juntas. In the work we only discuss Friedgut's theorem. Readers interested in Bourgain's theorem can refer to [13].

Below is a list of natural properties which can be shown to be well approximated by juntas using Fiedgut's theorem and Bourgain's theorem. Boolean literals (dictators), conjunctions, s-term monotone DNFs, decision lists, size-s decision trees, size-s branching programs, s-term DNFs, size-s Boolean formulas, s-sparse polynomials, size-s Boolean circuits, functions with Fourier degree $\leq$ d. Definitions of these can be found in [44].

Theorem 3.15 (Friedgut)**.** *For every boolean function $f : \{0,1\}^n \to \{0,1\}$, there exists another boolean function $g : \{0,1\}^n \to \{0,1\}$ such that $\mathrm{dist}_{\mathrm{Ham}}(f,g) \leq \epsilon$ and g is a $2^{O(\frac{I(f)}{\epsilon})}$-junta, where $I(f) = \sum_{i=1}^{n} \mathrm{Inf}_f(i)$.*

Friedgut's theorem was originally defined for boolean valued functions. Later, Feldman and Vondrák in [29] prove a generalization of Friedgut's theorem for real valued functions. This generalization gives an approximation by juntas result for self-bounding functions, which we define in the last chapter. Below is the theorem statement.

Feldman–Vondrák junta theorem. *(Self-bounding functions) Fix any $\epsilon \in (0, \frac{1}{2})$. For every self-bounding function $f : \{0,1\}^n \to [0,1]$, there exists a self-bounding function $g : \{0,1\}^n \to [0,1]$ that is a $2^{O(\frac{1}{\epsilon^2})}$-junta such that $\|f - g\|_2 \leq \epsilon$.*

Submodular functions with range $[0,1]$ are self-bounding and hence we get that every submodular function can be approximated by a $2^{O(\frac{1}{\epsilon^2})}$-junta. In the same paper they prove a much better and tighter approximation by juntas result for submodular functions using the structural properties of submodular functions. The precise statement is below.

Feldman–Vondrák junta theorem. *(Submodular functions) Fix any $\epsilon \in (0, \frac{1}{2})$. For every submodular function $f : \{0,1\}^n \to [0,1]$, there exists a submodular function $g : \{0,1\}^n \to [0,1]$ that is a $O(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon})$-junta such that $\|f - g\|_2 \leq \epsilon$.*

Feldman and Vondrák prove the above theorem with the help of an algorithm that given a submodular function f identifies a set of relevant variables J, such that $|J|$ is small. The algorithm proceeds as follows: maintain two sets S and T initialized to $\emptyset$, let S_δ represent a set obtained by including each element in S with probability δ. Add an element $i \notin S$ to a set S if $\Pr[f(S_\delta \cup \{i\}) - f(S_\delta) \geq \alpha] \geq \frac{1}{2}$, and add an element $i \notin T$ to the set T if $\Pr[f([n] \setminus T_\delta) - f([n] \setminus (T_\delta \cup \{i\})) \leq -\alpha] \geq \frac{1}{2}$. Intuitively set S consists of variables that will increase the function value by a significant amount on adding to most of the sets, and T consists of variables that decrease the function value by a significant amount on removing from most sets. The set S takes care of relevant variables in monotone submodular functions we need T to take care of the non-monotone case. The algorithm returns a set $J = S \cup T$. They argue that $|J|$ is small, if it was not small the function f would not have bounded range $[0,1]$. The submodular junta that is close to f is f_J.

Chapter 4

Testing by implicit learning

In this chapter we discuss testing-by-implicit learning, which is a testing technique used to obtain constant query testers for lots of natural properties. Before we jump into testing-by-implicit learning we briefly discuss the relation between testing and learning and see why testing by proper learning can not achieve constant query testers. We end the chapter by discussing the limitations of the testing-by-implicit learner of Diakonikolas et al. [23] and also discuss other testing-by-implicit learners.

4.1 Testing by proper learning

In this section we discuss the connections between property testing and learning. We mention how to use a learning algorithm to get a testing algorithm. In this section when we say learning or proper learning we are referring to proper PAC learning defined in Definition 2.1.

Testing by learning: There is a strong connection between property testing and learning theory that goes back to the seminal work of Goldreich, Goldwasser, and Ron [35]. As they first observed, any proper learning algorithm for the class of functions $\mathcal{P}$ can also be used to test $\mathcal{P}$: run the learning algorithm on f with error parameter $\frac{\epsilon}{2}$ and samples drawn from the uniform distribution, and verify whether the resulting hypothesis function h is $\frac{3}{4}\epsilon$-close to the function f or not. If yes then accept h otherwise reject it. If $f \in \mathcal{P}$ then the learning algorithm will return a $h \in \mathcal{P}$ such that $\text{dist}_{\text{Ham}}(f, h) \leq \frac{\epsilon}{2}$, hence h is $\frac{3}{4}\epsilon$-close to f and we accept it. If f was ϵ-far from $\mathcal{P}$ then $\text{dist}_{\text{Ham}}(f, h) \geq \epsilon$, hence f is rejected. Note that proper learning algorithms have the promise that h always belongs to $\mathcal{P}$. The

query complexity of the tester is equal to the sample complexity of the learner plus the number of queries needed to approximate distance between f, h. Distance approximation up to an additive error of $\frac{\epsilon}{4}$ can be done efficiently so the query complexity of the tester is asymptotically the same as the sample complexity of the learner. The same holds for the time complexity too. Hence we can always get a tester with the same query complexity and time complexity of a learner.

This approach yields good bounds on the number of queries required to test many properties of functions but using simple information theory arguments it can be shown that it cannot yield query complexity bounds that are smaller than $O(\log n)$ for almost all natural properties of functions over $\{0,1\}^n$. Intuitively if the function class has N functions that are reasonably far apart from each other then we would need $O(\log N)$ queries, as each query could eliminate half the functions, to narrow down to one we would need $O(\log N)$ queries. Most function classes have at least n such functions so the lower bound for the query complexity would be $\Omega(\log n)$. For example: consider the function class which consists of functions with one literal, this has n functions and each function differs from every other function on at least half the inputs. Hence to learn this we would need at least $\log n$ queries.

The relation between learning and testing means, the PAC learning algorithm of [29] gives a tester for submodularity in ℓ_1 testing model with query complexity $\text{poly}(\frac{1}{\epsilon}) \log n + 2^{\tilde{O}(1/\epsilon^2)}$. This is essentially the testing algorithm of Feldman and Vondrák [29]. They improve the dependence on ϵ from $2^{\tilde{O}(1/\epsilon^2)}$ to $2^{\tilde{O}(1/\epsilon)}$ using techniques in [49].

The testing by proper learning algorithm for submodular functions vaguely looks something like this:

- Identify the $\tilde{O}(\frac{1}{\epsilon^2})$ relevant variables.
- Generate samples $\{(x^i, f(x^i))\}_{1 \leq i \leq m}$.
- Check if there is a function $h \in \mathcal{P}_{core}$ which is close to f on the samples generated in second step. If yes accept the function f otherwise reject it.

The $\log n$ factor in the query complexity comes from the step of identifying the relevant variables and it can be shown that for finding the relevant variables you need at least $\log n$ samples.

4.2 Testing by implicit learning

Diakonikolas et al. [23] bypassed the barrier of $\log n$ queries for the special case when every function that has some property $\mathcal{P}$ is close to a junta. Every k-junta f has corresponding "core" functions $f_{\text{core}} : \{0,1\}^k \to \{0,1\}$ that define its value based on the value of the k relevant coordinates of its input. Diakonikolas et al.'s key insight is that for testing properties whose functions are (very) close to juntas, it suffices to learn the core of the input function—without having to identify the relevant coordinates.

If we look closely at the testing by proper learning algorithm for submodularity we can see that the only place where we are using the identified relevant variables is to figure out what values the relevant variables are taking on the samples generated. If we can somehow achieve this without identifying the relevant variables then we can get rid of the $\log n$ term. This is exactly what testing-by-implicit learning does, it generates samples uniformly at random and tells what values the relevant variables are taking in the samples.

4.2.1 Brief description of Diakonikolas et al.'s algorithm

Let every function in $\mathcal{P}$ be τ-close to a k-junta in $\mathcal{P}$. Let $\mathcal{F}$ be the set of all k-juntas in $\mathcal{P}$. The testing by implicit learning algorithm is inspired from Fischer et al.'s [31] junta testing algorithm and proceeds in three main steps:

- **Identifying relevant subsets** In this step we partition our n variables into roughly k^2 buckets(subsets) uniformly at random. After this step with very high probability every subset will have at most one relevant variable. We refer to a variable as relevant if its influence is over a threshold θ, which is a function of τ and k. We want to identify the subsets having relevant variables. For this we pick $O(k \log k)$ blocks of k buckets, and do the independence test on each block a few times and if a block passes the independence test every time we label all the buckets in the block to not relevant. At the end we label all the unlabeled buckets to relevant, let them be $I = \{I_1, ..., I_l\}$. If there are more than k relevant buckets then reject the function. If the function in fact belonged to the class $\mathcal{P}$ then this scenario will occur with very low probability.

- **Generating samples** Once we identify the relevant subsets, $I_1, ..., I_l$, we construct a set of m labeled samples $(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})$, where $x^{(i)}$ is uniformly random over $\{0,1\}^k$ such that if $f \in \mathcal{P}$ with high probability there exists a function $f' \in \mathcal{F}_{\text{core}}$ which agrees on the samples. We first generate m labeled samples, $(z^{(1)}, y^{(1)}), ..., (z^{(m)}, y^{(m)})$ on f, where $z^{(i)}$ is picked uniformly at random from

$\{0,1\}^n$ and $f(z^{(i)}) = y^{(i)}$. To construct the set of samples we want, in each $z^{(i)}$ we identify what values the relevant variables are set to. The value of $x_j^{(i)}$ can be fixed as follows: do independence test on $I_j^0 = \{r : (r \in I_j) \cap (z_r^{(i)} = 0)\}$ and $I_j^1 = \{r : (r \in I_j) \cap (z_r^{(i)} = 1)\}$ and check which of the sets has the relevant variable. If both of them fail the independence test reject the function, if both of them pass the test then choose the value arbitrarily. But if just I_j^0 fails the test then $x_j^{(i)} = 0$ similarly if just I_j^1 fails the test then $x_j^{(i)} = 1$. By doing this for all $j \in [l]$ we get values of the first l coordinates in $x^{(i)}$, set the values of the rest $k - l$ coordinates uniformly at random.

- **Checking consistency** In this step we check if there is a function $f^{'} \in \mathcal{F}_{\text{core}}$ that agrees on the samples generated in the previous step or not. If there exists such a function then we accept f otherwise we reject f.

The proof proceeds as follows: Any function close to a k-junta will pass the step of identifying relevant subsets with high probability and the identified buckets will have the following properties 1) None of the relevant variables are eliminated 2) Every bucket chosen has at most one relevant variable and 3) The influence $[n] \setminus I$ is low. If $f \in \mathcal{P}$ then there exists a function $f^{'} \in \mathcal{F}_{core}$, such that $f(x^{(i)}) = y^{(i)}$, $\forall i \in [m]$. This proves that every function $f \in \mathcal{P}$ is accepted. If the function, f, is ϵ-far from $\mathcal{P}$, then it is rejected while identifying the crucial subsets or generating samples and if it is not then none of the functions in $\mathcal{F}_{core}$ will match with all the generated samples.

The above algorithm would work for small τ(e.g. $O(2^{-k})$). The bottleneck on tolerance of the above algorithm is the second step of generating samples. We need at least $m = 2^{O(k)}$ samples to learn the core, as the size of the core function class could be up to 2^{2^k}. If $f \in \mathcal{P}$ when generating the samples whenever we do the independence test on I_j^0 and I_j^1 we want the set that does not contain the relevant variable to pass the test, for every bucket and every sample. The probability that it does not happen is upper bounded by $mk\tau$, and we want this value to be less than a constant. This gives us that $\tau \leq \frac{1}{m} = 2^{-O(k)}$. But most of the properties they are trying to test are very close to juntas so they are still okay with it.

4.2.2 Other testing by implicit learning algorithms

In this section we briefly mention other testing by implicit learning algorithms. We start the section by defining things we need for this section.

We define fourier spectrum of $f : \{0,1\}^n \to \{0,1\}$, $Spec(f)$, as $\{\alpha \in \{0,1\}^n : \hat{f}(\alpha) \neq 0\}$. A function $f : \{0,1\}^n \to \{0,1\}$ is said to be k-dimensional if $Spec(f)$ is a k-dimensional space of $\{0,1\}^n$. And f is said to be s-sparse if $|Spec(f)| \leq s$. And the spectral norm of f is $\sum_{\alpha \in \{0,1\}^n} |\hat{f}(\alpha)|$.

We say a property, $\mathcal{P}$, of boolean functions $\{0,1\}^n \to \{0,1\}$ linear invariant if for every $f \in \mathcal{P}$, $f \circ A \in \mathcal{P}$ for every square matrix A. Where $f \circ A(x) = f(Ax)$ for any $x \in \{0,1\}^n$. We define a linear invariant extension of a property $\mathcal{P}$, $L(P)$, as $L(\mathcal{P}) = \{f \circ A | f \in \mathcal{P}, A \text{ is any linear transformation matrix}\}$.

In [24], Diakonikolas et al. achieve an efficient tester for a class of functions called $GF(2)$ polynomials in terms of both query complexity and time complexity compared to [23] which gives a polynomial query tester but the running time is exponential. They achieve this by replacing the naive learning step at the end of Diakonikolas et al.'s [23] algorithm(comparing samples with all the functions in the core function class), with a more efficient learning algorithm.

In [20], Chakraborty et al. improve the query complexity of the tester in [23], by developing an efficient way to generate samples from the core of a k-junta $f^{'} : \{0,1\}^n \to \{0,1\}$, when only given oracle access to a function $f : \{0,1\}^n \to \{0,1\}$ that is close to $f^{'}$. Though this improves the query complexity, it works only when the functions in $\mathcal{P}$ are ϵ-close to k-juntas for some $k < O(\epsilon^{-\frac{1}{4}})$.

In [36], Gopalan et al. try to give a general characterization of boolean functions that are efficiently testable. They give efficient testers for functions having low fourier dimensionality or low fourier sparsity. They extend this to give an implicit learning algorithm which tests any sub-property of having concise fourier representation.

In [51] Wimmer and Yoshida gave a testing by implicit learner that can test any linear-invariant property which can be approximated by functions with small spectral norm using constant number of queries. This implies that $L(submodular)$(linear-invariant closure of submodular functions) can be tested using constant number of queries. It is not immediately clear if or how the algorithm can be extended to test submodular functions itself but it is an interesting open question to explore.

4.2.3 Limitations of Diakonikolas et al.'s [23] testing by implicit learning algorithm

The Feldman-Vondrák junta theorem suggests a natural approach for obtaining a constant query complexity for the same problem by combining it with a testing-by-implicit-learning

algorithm. In order to implement this approach, however, new testing-by-implicit-learning techniques are required to overcome two obstacles.

The first obstacle is that most existing testing-by-implicit-learning algorithms [23, 24, 20, 36] are designed for properties that contain functions which are close to juntas in Hamming distance, not ℓ_p distance. This is a stronger condition, and enables the analysis of these algorithms to assume that with large probability, when f is very close to a k-junta f', the queries x made by the algorithm all satisfy $f(x) = f'(x)$. In the ℓ_p distance model, however, we can have a function f that is extremely close to a k-junta but still has $f(x) \neq f'(x)$ for many (or even every!) input x.

The second (related) obstacle that we encounter when considering submodular functions is that most of the current testing-by-implicit-learning algorithm only works in the regime where the functions in $\mathcal{P}$ are ϵ-close to k-juntas for some $k \ll \epsilon^{-1/2}$. (See for example the discussion in §2.5 of [48].) This condition is satisfied by the properties of Boolean functions that have been studied previously, but the bounds in the Feldman–Vondrák junta theorem, however, do not satisfy this requirement.

Chapter 5

Our implicit learning tester

In this section we present our new testing by implicit learning algorithm which overcomes the limitations mentioned at the end of the previous section and we also provide proofs of Theorem 1.7 and Theorem 1.8 .

We first show how to get Theorem 1.7 from Theorem 1.8 and rest of the chapter will be dedicated to proving Theorem 1.8.

Proof of Theorem 1.7. Consider any property $\mathcal{P}$ that contains only functions which are $\frac{\epsilon}{10^7}$-close to some k-junta. Let $\mathcal{F}$ be the set of all k-juntas that are $\frac{\epsilon}{10^7}$-close to some function in $\mathcal{P}$. Run the tolerant tester from Theorem 1.8 on $\mathcal{F}$, with error parameter $\frac{\epsilon}{10}$, accept if the tolerant tester accepts and reject if the tolerant tester rejects. This tester accepts every function that is $\frac{\epsilon}{10^7}$-close and rejects functions that are $\frac{\epsilon}{10}$-far from $\mathcal{F}$. If $f \in \mathcal{P}$ then it is $\frac{\epsilon}{10^7}$-close to $\mathcal{F}$ hence is accepted with $\frac{2}{3}$ probability and if the function f is ϵ-far from $\mathcal{P}$ it is $(\epsilon - \frac{\epsilon}{10^7}) > \frac{\epsilon}{10}$-far from $\mathcal{F}$ hence is rejected with probability $\frac{2}{3}$. □

The proof of Theorem 1.8 is established by analyzing the Implicit Learning Tester algorithm presented in the next section.

5.1 Algorithm

In this section we present our testing by implicit learning algorithm.

Our algorithm differs from the previous algorithms in the following sense:

The current testing-by-implicit-learning algorithms proceed in two main stages. In the first stage, the coordinates in $[n]$ are randomly partitioned into poly(k) parts, and an influence test is used to identify the (at most k) parts that contain relevant variables of an unknown input function f that is very close to being a k-junta. In the second stage, inputs $x \in [n]$ are drawn at random according to some distribution, the value $f(x)$ is observed, and the value of the relevant coordinate in each of the parts identified in the second stage is determined using more calls to the influence test.

The Implicit Learning Tester algorithm that we introduce in this paper reverses the order of the two main stages. In the first stage, it draws a sequence of q queries $X = (x^{(1)}, \ldots, x^{(q)})$ at random and queries the value of f on each of these queries. It also uses X to partition the coordinates in $[n]$ into 2^q random parts according to the values of the coordinate on the q queries. In the second stage, the algorithm then uses an influence estimator to identify the k parts that contain the relevant coordinates of a k-junta that is close to f and, since all the coordinates in a common part have the same value on each of the q queries, learn the value of the k relevant coordinates on each of these initial queries. The algorithm then checks whether the core function thus learned is consistent with those of functions in the property being tested.

The main advantage of the Implicit Learning Tester algorithm is that its analysis does not require the assumption that our samples are exactly consistent with those of an actual k-junta (instead of those of a function that is only promised to be *close* to a k-junta). This feature enables us to overcome the obstacles listed in the previous section, at the cost of adding a few complications to the analysis, as described below.

Outline of the algorithm:

1. Draw a sequence of q queries $X = (x^{(1)}, \ldots, x^{(q)})$ at random and query the value of f on each of these queries. Use X to partition the coordinates in $[n]$ into 2^q random parts, $S_1, \ldots, S_{2^q}$ according to the values of the coordinate on the q queries. From now on through the algorithm we refer to $S_i's$ as buckets.

2. In the second stage, the algorithm then uses an influence estimator to identify the k buckets that contain the relevant coordinates of a k-junta that is close to f. This is done as follows: Partition the 2^q buckets into $100k^4$ parts, $P_1, \ldots, P_{100k^4}$, uniformly at random and find the best k parts, $P_{0,1}, \ldots, P_{0,k}$, such that the estimated influence of rest of the variables is lowest.

3. For all i partition $P_{0,i}$ into random equi-partition $P_{0,i,0}, P_{0,i,1}$. For each i pick one of $P_{0,i,0}, P_{0,i,1}$ such that the estimated influence of the parts left out is lowest, i.e of the

2^k options pick the best option. Repeat this step till the partitions can not be split any further. At the end we will be left with k buckets. Let the buckets left at the end be $\{b_1, \dots b_k\}$. If the influence of the variables $[n] \setminus \cup_{i=1}^{k} b_i$ is greater than $\frac{\epsilon^2}{1000}$ we reject the function. We assume our k relevant variables are in $\{b_1, \dots b_k\}$ and since all the variables in a bucket take the same value on all the samples we know what value our relevant variables take on our samples.

4. Check if there exists a function in the core which is close to f on the samples $x^{(1)}, \dots, x^{(q)}$. If yes accept f otherwise reject f.

Algorithm 2: IMPLICIT LEARNING TESTER($\mathcal{F}, k, \epsilon$)

Data: $q = \frac{2^{O(k)}}{\epsilon^5}$, $m = O(\frac{k^6}{\epsilon^5})$, $r = \log \frac{2^k}{100k^4}$

Draw $x^{(1)}, \dots, x^{(q)} \in \{0,1\}^n$ independently and uniformly at random;

For each $c \in \{0,1\}^q$, define $S_c \leftarrow \left\{ i \in [n] : \left(x_i^{(1)}, \dots, x_i^{(q)}\right) = c \right\}$;

Let $P_1, \dots, P_{100k^4}$ be a random equi-partition of $\{0,1\}^q$;

for *each* $J \subseteq [100k^4]$ *of size* $|J| = k$ **do**

  $S_J \leftarrow \bigcup_{j \in J} \bigcup_{c \in P_j} S_c$;

  $\eta_J \leftarrow \text{ESTIMATEINF}(f, [n] \setminus S_J, m)$;

$\{j_1^*, \dots, j_k^*\} \leftarrow \text{argmin}_J \, \eta_J$;

$(P_{0,1}, \dots, P_{0,k}) \leftarrow (P_{j_1^*}, \dots, P_{j_k^*})$;

for $\ell = 1, \dots, r$ **do**

  Let $P_{\ell,i,0}, P_{\ell,i,1}$ be a random equi-partition of $P_{\ell-1,i}$ for each $i \le k$;

  for *every* $z \in \{0,1\}^k$ **do**

    $S_z \leftarrow \bigcup_{i \le k} \bigcup_{c \in P_{\ell,i,z_i}} S_c$;

    $\eta_z \leftarrow \text{ESTIMATEINF}(f, [n] \setminus S_z, m)$;

  $z_\ell^* \leftarrow \text{argmin}_z \, \eta_z$;

  For each $i \le k$, update $P_{\ell,i} \leftarrow P_{\ell,i,z_\ell^*}$;

Let $B = \{b_1, ..., b_k\} \leftarrow \bigcup_{i \le k} P_{r,i}$;

If $\text{ESTIMATEINF}(f, [n] \setminus S_B, m) > \epsilon^2/1000$, reject;

Let $\phi : \{0,1\}^n \to \{0,1\}^k$ be any projection that satisfies $\phi(x)_i \in S_{b_i}$ for each $i \le k$;

for $h \in \mathcal{F}_{\text{core}}^{(\frac{\epsilon}{1000})}$ **do**

  If $\frac{1}{q} \sum_{i=1}^{q} \left(f(x^{(i)}) - h(\phi(x^{(i)})) \right)^2 \le 0.35\epsilon$, accept and return h;

Reject;

5.2 Analysis

In this section we provide a proof of Theorem 1.8 by analyzing the algorithm we presented in the previous section. We show that our testing by implicit learner is a $(\frac{\epsilon}{10^6}, \epsilon)$ tolerant tester for $\mathcal{F}$ with the desired properties.

We prove Theorem 1.8 by showing that every function f that is $\frac{\epsilon}{10^6}$-close to $\mathcal{F}$ will be accepted with probability $\frac{5}{6}$(Claim 5.3) and every function f that is ϵ-far from $\mathcal{F}$ is rejected with probability $\frac{5}{6}$. We prove that every function ϵ-far from $\mathcal{F}$ will be rejected in two parts, in first part we deal with functions that are $\frac{\epsilon}{100}$-far from being a k-junta(Claim 5.4) and in the second part we deal with functions $\frac{\epsilon}{100}$-close to a k-junta but are ϵ-far from $\mathcal{F}$(Claim 5.5).

The proofs of the above mentioned claims depend on two technical lemmas Lemma 5.1 and Lemma 5.2. Lemma 5.1 shows that when the input function f is close to a k-junta, then with reasonably large probability, the function f is close to a junta on the set B of k parts that is identified by the algorithm. We prove this lemma by showing that the influence of the variables left out at each step is very low so influence of union of the variables left out at is still low. This is true because the function is assumed to be a k-junta and at every step we are leaving out the set which has the lowest estimated influence.

Lemma 5.1. *For any $\varepsilon > 0$, if the function $f : \{0,1\}^n \to [0,1]$ is ε-close to a k-junta and every call to* EstimateInf *returns an influence estimate with additive error at most $\frac{\varepsilon^2}{100k^2}$, then the set B obtained by the* Junta-Property Tester *satisfies* $\Pr\left[\mathrm{Inf}_f([n] \setminus S_B) > 100\varepsilon^2\right] \leq \frac{1}{20}$.

The second lemma shows that the estimate in Step 20 provides a good estimate of the distance between f and the functions in $\mathcal{P}$.

Lemma 5.2. *Fix $\varepsilon > 0$. Let $f : \{0,1\}^n \to [0,1]$ be a function that satisfies* $\mathrm{dist}_2(f,g) \leq \varepsilon$ *for some function g that is a junta on $J \subseteq [n]$, $|J| \leq k$. Then for every $h_{\mathrm{core}} \in \mathcal{F}_{\mathrm{core}}^{(\frac{\varepsilon}{1000})}$, the mapping $\psi : \{0,1\}^n \to \{0,1\}^k$ defined in the* Implicit Learning Tester *and the function $h = h_{\mathrm{core}} \circ \psi$ satisfy*

$$\left| \left(\frac{1}{q} \sum_{i}^{q} \left(f(x^{(i)}) - h(x^{(i)}) \right)^2 \right)^{\frac{1}{2}} - \mathrm{dist}_2(g,h) \right| \leq 3\varepsilon$$

except with probability at most $2e^{-16q\varepsilon^4} + \frac{5k^2}{2^q}$.

The proofs of these lemmas are presented in Sections 5.2.1 and 5.2.2. We now show how they are used to complete the proof of Theorem 1.8.

As a first observation, we note that by Hoeffding's inequality and the union bound, all of the calls to EstimateInf have additive error at most $\frac{\epsilon^2}{10^6 k^2}$ except with probability at most $\frac{1}{6}$. In the following, we assume that this condition holds and show how, when it does, the algorithm correctly accepts or rejects with probability with probability at least $\frac{5}{6}$.

We begin by establishing the completeness of the Implicit Learning Tester.

Claim 5.3 (Completeness)**.** *When f is $\frac{\epsilon}{10^6}$-close to the property $\mathcal{F}$ of k-juntas, the* Implicit Learning Tester *accepts with probability at least $\frac{5}{6}$.*

Proof. First, by Lemma 5.1, the probability that f is rejected on step 17 is at most $\frac{1}{18}$. In the rest of the proof, we will show that except with probability at most $\frac{1}{9}$, there is a function $h_{\text{core}} \in \mathcal{F}_{\text{core}}^{(\frac{\epsilon}{1000})}$ for which the algorithm accepts on line 20.

Let $g \in \mathcal{F}$ be a function that satisfies $\text{dist}_2(f, g) \leq \frac{\epsilon}{10^6}$. Without loss of generality, we can assume that g is a junta on $[k]$. Let $J = [k] \cap S_B$ be the set of the junta variables of g that are contained in the final parts selected by the algorithm. Again without loss of generality (by relabeling the input variables once again if necessary), we can assume that $J = [j]$ for some $j \leq k$, and $i \in S_{b_i}$, for $i \leq j$.

Define $\psi : \{0,1\}^n \to \{0,1\}^k$ to be the mapping defined by $\psi(x) = (x_1, \ldots, x_j, x_{i_1}, \ldots, x_{i_{k-j}})$ where $i_1, \ldots, i_{k-j} \in [n] \setminus [k]$ are representative coordinates from the remaining parts $b \in B$ for which $P_b \cap [k] = \emptyset$.

Let $g_{\text{core}} \in \mathcal{F}_{\text{core}}$ be the core of g corresponding to the projection $\psi(x) = (x_1, \ldots, x_k)$, and let $h_{\text{core}} \in \mathcal{F}_{\text{core}}^{(\frac{\epsilon}{10^6})}$ be the discretized approximation to g_{core}. Define $h = h_{\text{core}} \circ \psi$. By our choice of g, we have $\text{dist}_2(f, g) \leq \frac{\epsilon}{10^6}$. In order to invoke Lemma 5.2, we now want to bound $\text{dist}_2(g, h)$.

Let $h^* \in \mathcal{F}^{(\frac{\epsilon}{10^6})}$, be the discretized approximation of g. Then $\text{dist}_2(g, h^*) \leq \frac{\epsilon}{10^6}$ and the triangle inequality implies that

$$\text{dist}_2(f, h^*) \leq \text{dist}_2(f, g) + \text{dist}_2(g, h^*) \leq \tfrac{2\epsilon}{10^6}$$

and that

$$\text{dist}_2(g, h) \leq \text{dist}_2(g, h^*) + \text{dist}_2(h^*, h) \leq \text{dist}_2(h^*, h) + \tfrac{\epsilon}{10^6}.$$

Furthermore, since $h_{\text{core}} = h^*_{\text{core}}$,

$$\begin{aligned}\text{dist}_2(h^*, h) &= \mathop{\mathrm{E}}_x \left[\left(h^*_{\text{core}}(x_1, \ldots, x_k) - h^*_{\text{core}}(x_1, \ldots, x_j, x_{i_1}, \ldots, x_{i_{k-j}}) \right)^2 \right]^{\frac{1}{2}} \\ &= 2\,\text{Inf}_{h^*_{\text{core}}}([k] \setminus [j])^{\frac{1}{2}} = 2\,\text{Inf}_{h^*}([n] \setminus [j])^{\frac{1}{2}}.\end{aligned}$$

By Proposition 3.10 and Lemma 5.1, except with probability at most $\frac{1}{18}$,

$$\mathrm{Inf}_{h^*}([n] \setminus [j])^{\frac{1}{2}} \leq \mathrm{Inf}_f([n] \setminus [j])^{\frac{1}{2}} + \mathrm{dist}_2(f, h^*) \leq \mathrm{Inf}_f([n] \setminus S_B)^{\frac{1}{2}} + \tfrac{2\epsilon}{10^6} \leq \tfrac{12\epsilon}{10^6}$$

and the distance between g and h is bounded by $\mathrm{dist}_2(g, h) \leq \frac{13}{10^6}\epsilon$. When this bound holds, by Lemma 5.2 with $\varepsilon = \frac{\epsilon}{100}$, the algorithm accepts f for this h except with probability at most $\frac{1}{18}$. □

The soundness of the Implicit Learning Tester is established in two steps. The first step is to show that it rejects functions that are far from being k-juntas.

Claim 5.4 (Soundness I)**.** *If f is $\frac{\epsilon}{100}$-far from being a k-junta, then the* Implicit Learning Tester *rejects with probability at least $\frac{5}{6}$.*

Proof. The initial partition $S_{P_1}, \ldots, S_{P_{100k^4}}$ is a random partition of $[n]$ with more than $20k^2$ parts so, by Lemma 3.13, with probability at least $\frac{5}{6}$, for any union $J \subseteq [n]$ of at most k of these parts we have $\mathrm{Inf}_f([n] \setminus J) \geq \frac{\epsilon^2}{400}$. When this is the case, the inclusion $S_B \subseteq \overline{L_0}$ and the fact that L_0 is the complement of the union of some set of k parts in the random partition imply that

$$\mathrm{Inf}_f([n] \setminus S_B) \geq \mathrm{Inf}_f(L_0) \geq \frac{\epsilon^2}{400}$$

and, under the assumed accuracy of EstimateInf calls, the algorithm rejects f in Step 17. □

We now complete the soundness analysis of the Implicit Learning Tester by showing that it also rejects functions that are far from $\mathcal{F}$ but close to being k-juntas.

Claim 5.5 (Soundness II)**.** *If f is $\frac{\epsilon}{100}$-close to a k-junta, but is $\frac{99\epsilon}{100}$-far from $\mathcal{F}$, then the* Implicit Learning Tester *rejects with probability at least $\frac{5}{6}$.*

Proof. Let g be any k-junta that satisfies $\mathrm{dist}_2(f, g) \leq \frac{\epsilon}{100}$. For any $h_{\mathrm{core}} \in \mathcal{F}_{\mathrm{core}}^{(\frac{\epsilon}{1000})}$ and any injective mapping $\psi : \{0,1\}^n \to \{0,1\}^k$, the function $h = h_{\mathrm{core}} \circ \psi$ is in $\mathcal{F}^{(\frac{\epsilon}{1000})}$ and so by the triangle inequality,

$$dist_2(f, \mathcal{F}^{(\frac{\epsilon}{1000})}) \geq dist_2(f, \mathcal{F}) - \tfrac{\epsilon}{1000}$$

and

$$\mathrm{dist}_2(g, h) \geq \mathrm{dist}_2(f, h) - \mathrm{dist}_2(f, g) \geq \tfrac{99}{100}\epsilon - \tfrac{\epsilon}{1000} - \tfrac{\epsilon}{100} \geq \tfrac{97}{100}\epsilon.$$

Then, by Proposition 3.10 and the union bound over all $|\mathcal{F}_{\text{core}}^{(\frac{\epsilon}{1000})}| \leq (1000/\epsilon)^{2^k}$ functions in $\mathcal{F}_{\text{core}}^{(\frac{\epsilon}{1000})}$, with probability at least $\frac{5}{6}$, the condition in Step 20 is never satisfied and the algorithm rejects. □

Claim 5.3 shows that Implicit Learning Tester accepts every function that is $\frac{\epsilon}{10^6}$ close to $\mathcal{F}$ with the desired probability, and Claims 5.4 and 5.5 shows that it rejects all functions that are ϵ-far from $\mathcal{F}$. Finally, we note that the query complexity of the algorithm is at most $q + 2m(2^{O(k\log(k))} + 2^k q = \frac{2^{\tilde{O}(k\log k)}}{\epsilon^{10}}$, as claimed. Finally, the general result for ℓ_p testing when $p \neq 2$ follows from Fact 6.4.

5.2.1 Proof of Lemma 5.1

Let f be any function ε-close to a k-junta and assume without loss of generality (by relabeling the input variables if necessary) that f is close to a junta on $[k]$. The definition of $P_1, \ldots, P_{100k^4}$ in step 3, means that $S_{P_1}, \ldots, S_{P_{100k^4}}$ is a random partition of $[n]$. So by the union bound, the probability that any two of the coordinates in $[k]$ land in the same part is at most $\frac{1}{100k^2}$.

For each $\ell = 0, 1, 2, \ldots, r$, let $L_\ell = [n] \setminus \bigcup_{i=1}^{k} S_{P_{\ell,i}}$ denote the set of variables that have been "eliminated" after ℓ iterations of the loop. Then $[n] \setminus S_B = L_r$ and

$$\mathrm{Inf}_f([n] \setminus S_B) = \mathrm{Inf}_f(L_0) + \sum_{\ell=1}^{r} \Big(\mathrm{Inf}_f(L_\ell) - \mathrm{Inf}_f(L_{\ell-1})\Big). \tag{5.1}$$

We bound both terms on the right-hand side of the expression separately.

By Proposition 3.10, we have $\mathrm{Inf}_f([n]\setminus[k]) \leq \varepsilon^2$ and so by the monotonicity of influence there is a choice of $J \subseteq [k^2]$ of size $|J| \leq k$ for which $\mathrm{Inf}_f([n] \setminus S_J) \leq \varepsilon^2$. The guaranteed accuracy on EstimateInf then implies that

$$\mathrm{Inf}_f(L_0) \leq (1 + \tfrac{2}{100k^2})\varepsilon^2. \tag{5.2}$$

Define $\mathcal{E} = \{\ell \leq r : (L_\ell \setminus L_{\ell-1}) \cap [k] \neq \emptyset\}$ to be the set of rounds for which the algorithm eliminated at least one of the coordinates in $[k]$. By this definition, each $\ell \in [r] \setminus \mathcal{E}$ satisfies $(L_\ell \setminus L_{\ell-1}) \cap [k] = \emptyset$ and

$$\begin{aligned}\sum_{\ell \in [r]\setminus\mathcal{E}} \mathrm{Inf}_f(L_\ell) - \mathrm{Inf}_f(L_{\ell-1}) &= \sum_{\ell\in[r]\setminus\mathcal{E}} \sum_{T:\, T\cap L_\ell \neq \emptyset \wedge T \cap L_{\ell-1} = \emptyset} \hat{f}(T)^2 \\ &\leq \sum_{T \subseteq [n]\setminus[k]} \hat{f}(T)^2 \leq \mathrm{Inf}_f([n] \setminus [k]) \leq \varepsilon^2. \end{aligned} \tag{5.3}$$

For each $\ell \in \mathcal{E}$, define $X_\ell = \{\cup_{i=1}^k S_{P_{\ell,i,1-(z^*_\ell)_i}} : S_{P_{\ell,i,1-(z^*_\ell)_i}} \cap [k] \neq \emptyset\}$ to be the set of coordinates in the parts that contain a coordinate in $[k]$ that was eliminated in the ℓth iteration of the loop. Let also $Y_\ell = \{\cup_{i=1}^k S_{P_{\ell,i,(z^*_\ell)_i}} : S_{P_{\ell,i,1-(z^*_\ell)_i}} \cap [k] \neq \emptyset\}$ be the coordinates in the parts that were kept instead. Then the guaranteed accuracy of EstimateInf and the choice of z^*_ℓ implies that

$$\mathrm{Inf}_f(L_\ell) \leq \mathrm{Inf}_f\big((L_\ell \setminus X_\ell) \cup Y_\ell\big) + 2\,\tfrac{\varepsilon^2}{100k^2}$$

and, therefore,

$$\begin{aligned}\sum_{\ell\in\mathcal{E}} \mathrm{Inf}_f(L_\ell) - \mathrm{Inf}_f(L_{\ell-1}) &\leq \frac{2\varepsilon^2}{1000k} + \sum_{\ell\in\mathcal{E}} \mathrm{Inf}_f\big((L_\ell \setminus X_\ell) \cup Y_\ell\big) - \mathrm{Inf}_f(L_{\ell-1}) \\ &\leq \frac{2\varepsilon^2}{1000k} + \sum_{\ell\in\mathcal{E}} \sum_{T:\, T\cap(L_\ell\setminus X_\ell)\neq\emptyset\wedge T\cap L_{\ell-1}=\emptyset} \hat{f}(T)^2 + \sum_{\ell\in\mathcal{E}} \sum_{T:\, T\cap Y_\ell\neq\emptyset\wedge T\cap L_{\ell-1}=\emptyset} \hat{f}(T)^2. \end{aligned} \tag{5.4}$$

As above, since $(L_\ell \setminus X_\ell) \cap [k] = \emptyset$,

$$\sum_{\ell\in\mathcal{E}} \sum_{T:\, T\cap(L_\ell\setminus X_\ell)\neq\emptyset\wedge T\cap L_{\ell-1}=\emptyset} \hat{f}(T)^2 \leq \sum_{T\subseteq[n]\setminus[k]} \hat{f}(T)^2 \leq \varepsilon^2. \tag{5.5}$$

It remains to bound the last sum on the right-hand side of (5.4). By splitting up the terms in this sum according to whether $|T| \leq k$ or not, we obtain

$$\sum_{T:T\cap Y_\ell\neq\emptyset\wedge T\cap L_{\ell-1}=\emptyset} \hat{f}(T)^2 \leq \sum_{|T|\leq k} \hat{f}(T)^2 \cdot \mathbf{1}[T\cap Y_\ell \neq \emptyset] + \sum_{|T|>k} \hat{f}(T)^2 \cdot \mathbf{1}[T\cap L_{\ell-1} = \emptyset].$$

Let $Z \subseteq [n]\setminus[k]$ denote the set of coordinates that occur in one of the the original parts $S_{P_1}, \ldots, S_{P_{100k^4}}$ that also contains one of the elements in $[k]$. Then $Y_\ell \subseteq Z$ and

$$\sum_{\ell\in\mathcal{E}} \sum_{|T|\leq k} \hat{f}(T)^2 \cdot \mathbf{1}[T\cap Y_\ell \neq \emptyset] \leq \sum_{\ell\in\mathcal{E}} \sum_{|T|\leq k} \hat{f}(T)^2 \cdot \mathbf{1}[T\cap Z \neq \emptyset] \leq k \cdot \sum_{|T|\leq k} \hat{f}(T)^2 \cdot \mathbf{1}[T\cap Z \neq \emptyset].$$

The probability, over the choice of $P_1, \ldots, P_{100k^4}$, that $T \cap Z \neq \emptyset$ is at most $|T|/100k^3$, so the expected value of the last expression (again over the choice of the initial partition) is bounded above by

$$\mathrm{E}\Big[\sum_{\ell\in\mathcal{E}} \sum_{|T|\leq k} \hat{f}(T)^2 \cdot \mathbf{1}[T\cap Y_\ell \neq \emptyset]\Big] \leq k \cdot \sum_{|T|\leq k, T\setminus[k]\neq\emptyset} \hat{f}(T)^2 \cdot \Big(\frac{k}{100k^3}\Big) \leq \frac{1}{100k} \cdot \mathrm{Inf}_f([n]\setminus[k]) \leq \frac{\varepsilon^2}{100k}. \tag{5.6}$$

Lastly, since $L_0 \subseteq L_{\ell-1}$ for each $\ell \geq 1$,

$$\sum_{|T|>k} \hat{f}(T)^2 \cdot \mathbf{1}[T \cap L_{\ell-1} = \emptyset] \leq \sum_{|T|>k} \hat{f}(T)^2 \cdot \mathbf{1}[T \cap L_0 = \emptyset].$$

A set T can be disjoint from L_0 only when its elements are contained in at most k of the parts of the initial random partition, which happens with probability at most $\frac{1}{100k^2}$ when $|T| > k$, so

$$\mathrm{E}\left[\sum_{\ell \in \mathcal{E}} \sum_{|T|>k} \hat{f}(T)^2 \cdot \mathbf{1}[T \cap L_{\ell-1} = \emptyset]\right] \leq \mathrm{E}\left[k \sum_{|T|>k} \hat{f}(T)^2 \cdot \mathbf{1}[T \cap L_0 = \emptyset]\right] \leq \frac{1}{100k} \sum_{|T|>k} \hat{f}(T)^2 \leq \frac{\varepsilon^2}{100k}, \tag{5.7}$$

where the last inequality uses the fact that $\sum_{|T|>k} \hat{f}(T)^2 \leq \mathrm{Inf}_f([n] \setminus [k])$.

Combining the inequalities (5.1)–(5.7), we obtain that the expected value of $\mathrm{Inf}_f([n] \setminus S_B)$ is bounded above by

$$\mathrm{E}\left[\mathrm{Inf}_f([n] \setminus S_B)\right] \leq (1 + \tfrac{2}{100k^2})\varepsilon^2 + \tfrac{2\varepsilon^2}{100k} + (1 + \tfrac{2}{100k})2\varepsilon^2 \leq 4\varepsilon^2.$$

Applying Markov's inequality and adding the probability that the junta variables are completely separated in the partition $P_1, \ldots, P_{100k^4}$ completes the proof of the lemma.

5.2.2 Proof of Lemma 5.2

For any $X = (x^{(1)}, \ldots, x^{(q)})$, let $\mathrm{dist}_X(f_1, f_2) = \left(\frac{1}{q}\sum_{i=1}^{q}\left(f_1(x^{(i)}) - f_2(x^{(i)})\right)^2\right)^{1/2}$ denote the empirical distance between f_1 and f_2 according to X. To prove the lemma, we want to show that $\mathrm{dist}_X(f, h)$ is within the specified bounds.

The function dist_X is a metric, so we can apply the triangle inequality to obtain

$$\mathrm{dist}_X(f, h) \leq \mathrm{dist}_X(f, g) + \mathrm{dist}_X(g, h).$$

By Hoeffding's inequality, when $x^{(1)}, \ldots, x^{(q)}$ are drawn independently and uniformly at random, the upper bound

$$\mathrm{dist}_X(f, g) \leq \mathrm{dist}_2(f, g) + \varepsilon \leq 2\varepsilon$$

holds except with probability at most $e^{-16q\varepsilon^4}$.

We now want to show that $\mathrm{dist}_X(g, h)$ is also close to $\mathrm{dist}_2(g, h)$. This analysis is a bit more subtle, however, because the choice of samples $x^{(1)}, \ldots, x^{(q)}$ is *not* independent of h

(as it affects what mapping ψ will be chosen by the algorithm). So before we can apply concentration inequalities, we must "decouple" X and h. To do so, we introduce a new random process for generating X. Let $\lambda : [n] \to \{0,1\}^q$ be chosen uniformly at random. This function corresponds to a random partition of the set $[n]$ of coordinates into 2^q parts. Let $\pi : \{0,1\}^q \to \{0,1\}^q$ be a random permutation. Then the random variable X obtained by setting $x_j^{(i)} = \pi(\lambda(j))_i$ has the desired uniform distribution over sequences of q vectors in $\{0,1\}^n$.

This random process is designed so that the choice of ψ in the algorithm (and therefore also h) is *independent* of π; the only information about X used in determining it is the identity of the parts defined by λ, not what values the coordinates in each parts receive on the q queries. Then

$$\mathop{\mathrm{E}}_{X}[\mathrm{dist}_X(g,h)] = \mathop{\mathrm{E}}_{\lambda,r}[\mathop{\mathrm{E}}_{\pi}[\mathrm{dist}_X(g,h)]]$$

where r represents the internal randomness of the algorithm outside of that used to generate X. With probability at least $k^2/2^q$, the partition λ completely separates the indices in J. Fix such a partition λ. Define $J^* = J \cup \mathrm{supp}(\psi)$. Then $|J^*| \leq 2k$. Define $Y = (y^{(1)}, \ldots, y^{(q)})$ by setting $y^{(i)} = x^{(i)}_{J^*}$. Since $\mathrm{dist}_X(g,h)$ only depend on the coordinates in J^*, we can write it equivalently as $\mathrm{dist}_Y(g,h)$.

Let D denote the distribution on Y induced by π. The distribution D is close to but not equal to the uniform distribution U on $\{0,1\}^{q \times |J^*|}$, since D is equivalent to the distribution obtained by making drawing $(y_i^{(1)}, \ldots, y_i^{(q)})$ for each $i \in J^*$ without replacement from $\{0,1\}^q$. Then

$$\begin{aligned}\Pr_{Y\sim D}[|\mathrm{dist}_Y(g,h) - \mathop{\mathrm{E}}_{Y\sim U}\mathrm{dist}_Y(g,h)| \geq \varepsilon] &\leq \mathrm{d_{TV}}(D,U) + \Pr_{Y\sim U}[|\mathrm{dist}_Y(g,h) - \mathop{\mathrm{E}}_{Y\sim U}\mathrm{dist}_Y(g,h)| \geq \varepsilon] \\ &\leq \tfrac{4k^2}{2^q} + e^{-16q\varepsilon^4}.\end{aligned}$$

In the last inequality, the bound $\mathrm{d_{TV}}(D,U) \leq \frac{(2k)^2}{2^q}$ is by the standard total variation bound between sampling with and without replacement [32] and the other bound on the other term is by Hoeffding's inequality.

Chapter 6

Applications and conclusion

In this chapter we discuss how the general theorem we proved in the previous chapter can be used to obtain a tester for submodularity and also a constant query tester for many other properties of valuation functions.

6.1 Testing properties of other valuation functions

Natural properties of bounded real-valued Boolean functions have been studied extensively in the context of valuation functions in algorithmic game theory [43]. For a sequence of n goods labeled with the indices $1, \ldots, n$, we can encode the value of each subset of these goods to some agent with a function $f : \{0,1\}^n \to [0,1]$ by setting $f(x)$ to be the (possibly normalized) value of the subset $\{i \in [n] : x_i = 1\}$ to the agent. Such a valuation function f is

Additive if there are weights $w_1, \ldots, w_n$ such that $f(x) = \sum_{i:x_i=1} w_i$;

a **Coverage function** if there exists a universe U, non-negative weights $\{w_u\}_{u \in U}$, and subsets $A_1, ..., A_n \subseteq U$ such that $f(x) = \sum_{u \in \bigcup_{i:x_i=1} A_i} w_u$.

Unit demand if there are weights $w_1, \ldots, w_n$ such that $f(x) = \max\{w_i : x_i = 1\}$;

OXS if there are $k \geq 1$ unit demand functions $g_1, \ldots, g_k$ such that $f(x) = \max\{g_1(x^{(1)}), \ldots, g_k(x^{(k)})\}$ where the maximum is taken over all $x^{(1)}, \ldots, x^{(k)}$ such that for every $i \in [n]$, $x_i = \sum_{j=1}^{k} x_i^{(j)}$;

Gross Substitutes if for any $p' \leq p \in \mathbb{R}^n$ and any x, x' that maximize $f(x) - \sum_{i:x_i=1} p_i$ and $f(x') - \sum_{i:x'_i=1} p'_i$, respectively, every $j \in [n]$ for which $x_j = 1$ and $p_j = p'_j$ also satisfies $x'_j = 1$;

Submodular if $f(x) + f(y) \geq f(x \wedge y) + f(x \vee y)$ for every $x, y \in \{0,1\}^n$, where $\wedge$ and $\vee$ are the bitwise AND and OR operations;

Fractionally subadditive (XOS) iff there are non-negative real valued weights $\{w_{ij}\}_{i,j \leq n}$ such that $f(x) = \max_i \sum_j w_{ij} \cdot x_j$;

Self-bounding if $f(x) \geq \sum_i (f(x) - \min_{x_i} f(x))$, where $\min_{x_i} f(x) = \min\{f(x), f(x \oplus e_i)\}$ and $\oplus$ is the bitwise XOR operator; and

Subadditive if $f(x \cup y) \leq f(x) + f(y)$ for every $x, y \in \{0,1\}^n$.

Each of these properties enforces some structure on valuation functions, and much work has been devoted to better understanding these structures (and their algorithmic implications) by studying the properties through the lenses of learning theory [3, 2, 28], optimization [26, 27], approximation [30, 29], and sketching [1]. The problem of testing whether an unknown valuation function satisfies one of these properties offers another angle from which we can learn more about the structure imposed on the functions that satisfy these properties.

The following is the hierarchy of the above mentioned properties. (See, e.g., [41].)

Lemma 6.1. *The properties of $\mathcal{F}_n$ defined in the introduction satisfy the inclusion hierarchy*

$$\textit{Additive} \subseteq \textit{Coverage} \subseteq \textit{Unit demand} \subseteq \textit{OXS} \subseteq \textit{Gross substitute} \subseteq \textit{Submodularity} \subseteq \textit{XOS} \subseteq \textit{Self-bounding}.$$

Other then submodularity another property that has been considered in the (standard Hamming distance) testing model is that of being a coverage function. Chakrabarty and Huang [17] showed that for constant values of $\epsilon > 0$, $O(nm)$ queries suffice to ϵ-test whether a function f is a coverage function on some universe U of size $|U| \leq m$. Note that, unlike in the learning and approximation settings, bounds on the number of queries required to test some property $\mathcal{P}$ do not imply anything about number of queries required to test properties $\mathcal{P}' \subset \mathcal{P}$, so even though coverage functions are submodular, results on testing submodularity do not imply any bounds on the query complexity for testing coverage

functions. Nonetheless, our next result shows that this property—along with most of the other properties of valuation functions listed above—can also be tested with a number of queries that is independent of n.

Theorem 6.2. *For any $\epsilon > 0$ and any $p \geq 1$, there are ϵ-testers in the ℓ_p testing model for additive functions, coverage functions, unit demand functions, OXS functions, and gross substitute functions that each have query complexity $2^{\tilde{O}(1/\epsilon^{\max\{2,p\}})}$, and there are ϵ-testers in the ℓ_p testing model for fractional subadditivity and self-bounded functions that have query complexity $2^{2^{\tilde{O}(1/\epsilon^{\max\{2,p\}})}}$.*

6.2 Relationship between different testing models

Before we present the proofs for theorems 1.6 and 6.2 we discuss the relation between different ℓ_p testing models. This relation along with the general theorem we proved in the previous chapter will give us the proofs.

There is a relationship between different ℓ_p distance metrics which we will later use to get a relationship between query complexities of ℓ_2 and ℓ_p, $p \geq 1$ testing models.

Proposition 6.3. *Let $f, g \in \mathcal{F}_n$. For all $p \geq q \geq 1$*

1. $\text{dist}_q(f, g) \leq \text{dist}_p(f, g)$
2. $\text{dist}_q^q(f, g) \geq \text{dist}_p^p(f, g)$

Proof. The first one follows from Jensen's inequality and the second one is true because the range of the functions is $[0, 1]$. □

We denote the worst case query complexity of testing for a property, $\mathcal{P}$, in ℓ_p testing model with proximity parameter ϵ by $Q_p(\mathcal{P}, \epsilon)$. The proposition below gives the relationship between different query models.

Proposition 6.4 (c.f. Fact 5.2 in [4])**.** *For any $\mathcal{P} \subseteq \mathcal{F}_n$, any $\epsilon > 0$, and any $p \geq 3$, the number $Q_p(\mathcal{P}, \epsilon)$ of queries required to ϵ-test $\mathcal{P}$ in the ℓ_p testing model satisfies*

1. $Q_1(\mathcal{P}, \epsilon) \leq Q_2(\mathcal{P}, \epsilon)$
2. $Q_p(\mathcal{P}, \epsilon) \leq Q_2(\mathcal{P}, \epsilon^{\frac{p}{2}})$.

Proof. By Proposition 6.3, $\text{dist}_2^2(f,g) \geq \text{dist}_p^p(f,g)$ for $p \geq 3$ and $\text{dist}_2(f,g) \geq \text{dist}_1(f,g)$. This implies that $\text{dist}_2^2(f,\mathcal{P}) \geq \text{dist}_p^p(f,\mathcal{P})$ for $p \geq 3$ and $\text{dist}_2(f,\mathcal{P}) \geq \text{dist}_1(f,\mathcal{P})$. We will later prove that given an ϵ-tester for property $\mathcal{P}$ in ℓ_2 testing model it can be used as an ϵ-tester for $\mathcal{P}$ in ℓ_1 testing model and an $\epsilon^{\frac{2}{p}}$-tester for $\mathcal{P}$ in ℓ_p testing model for $p \geq 3$. As every ϵ-tester in ℓ_2 testing model is also an ϵ-tester in ℓ_1 testing model, the worst case query complexity is worse in ℓ_2 testing model, $Q_1(\mathcal{P},\epsilon) \leq Q_2(\mathcal{P},\epsilon)$. As every ϵ-tester in ℓ_2 model is also a $\epsilon^{\frac{2}{p}}$-tester in ℓ_p model, if we use the ℓ_2 tester with $\epsilon = \epsilon^{\frac{p}{2}}$ this gives an ϵ-tester in ℓ_p model, hence worst case query complexity for ℓ_p model with proximity parameter ϵ is better than the worst case query complexity for the ℓ_2 model with proximity parameter $\epsilon^{\frac{p}{2}}$, $Q_p(\mathcal{P},\epsilon) \leq Q_2(\mathcal{P},\epsilon^{\frac{p}{2}})$.

Let W be a ϵ tester for $\mathcal{P}$ in ℓ_2 testing model. The algorithm accepts every function $f \in \mathcal{P}$ with probability at least $\frac{2}{3}$ and rejects every function f that satisfies $\text{dist}_2(f,\mathcal{P}) \geq \epsilon$ with probability $\frac{2}{3}$. The algorithm W is also a ϵ-tester in ℓ_1 testing model because it accepts every function $f \in \mathcal{P}$ with probability at least $\frac{2}{3}$ and rejects every function f that satisfies $\text{dist}_2(f,\mathcal{P}) \geq \text{dist}_1(f,\mathcal{P}) \geq \epsilon$ with probability $\frac{2}{3}$. Similarly, W is also a $\epsilon^{\frac{2}{p}}$-tester for $\mathcal{P}$ in ℓ_p testing model for $p \geq 3$ because it accepts every function $f \in \mathcal{P}$ with probability at least $\frac{2}{3}$ and rejects every function f that satisfies $\text{dist}_2(f,\mathcal{P}) \geq \text{dist}_p^{\frac{p}{2}}(f,\mathcal{P}) \geq (\epsilon^{\frac{2}{p}})^{\frac{p}{2}} = \epsilon$ with probability $\frac{2}{3}$. □

6.3 Applications

In this short section, we show how Theorems 1.6 and 6.2 both follow directly from Theorem 1.7, the junta theorem of Feldman and Vondrák, and Proposition 6.4.

Proof of Theorem 1.6. By the first part of the Feldman–Vondrák junta theorem, every submodular function $f \in \mathcal{F}_n$ is $\frac{\epsilon}{10^6}$-close to a k-junta for some $k = O(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})$. Therefore, by Theorem 1.7, submodularity can be tested with $2^{O(k\log k)}/\epsilon^{10} = 2^{\tilde{O}(1/\epsilon^2)}$ queries in the ℓ_2 testing model. By Proposition 6.4, the number of queries for testing submodularity in the ℓ_p testing model for any $1 \leq p < 2$ is also $2^{\tilde{O}(1/\epsilon^2)}$ and for any $p > 2$ it is $2^{\tilde{O}(1/(\epsilon^{p/2})^2)} = 2^{\tilde{O}(1/\epsilon^p)}$. □

Proof of Theorem 6.2. By Lemma 6.1, additive functions, coverage functions, unit demand functions, OXS functions, and gross substitute functions are all also submodular. Therefore, the first part of the Feldman–Vondrák junta theorem also applies to these functions and the rest of the proof is identical to that of Theorem 1.6.

Lemma 6.1 also implies that fractionally subadditive functions are self-bounding, so the second part of the Feldman–Vondrák junta theorem shows that every function f that has either of these properties is $\frac{\epsilon}{10^6}$-close to a k-junta for some $k = 2^{O(\frac{1}{\epsilon^2})}$. Therefore, by Theorem 1.7, fractional subadditivity and self-boundedness can both be tested with $2^{O(k \log k)}/\epsilon^{10} = 2^{2^{\tilde{O}(1/\epsilon^2)}}$ queries in the ℓ_2 testing model; the general result for the ℓ_p testing model again follows directly from Proposition 6.4. □

6.4 Discussion and open problems

Theorems 1.6–1.7 raise a number of intriguing questions. The most obvious question left open is whether we can also test subadditivity of real-valued functions with a constant number of queries: subadditive functions need not be close to juntas, so such a result would appear to require a different technique.

It is also useful to compare our bounds for submodularity testing with those for testing monotonicity: in the Hamming distance testing model, Seshadhri and Vondrák [49] showed that the query complexity for testing submodularity is at least as large as that for testing monotonicity. However, the best current bounds for testing monotonicity in the ℓ_p testing model have a linear dependence on n [4]. Is it also possible to test monotonicity with a constant number of queries? Or is it the case that testing submodularity is strictly easier than testing monotonicity in the ℓ_p testing setting?

Our tester for submodularity would work for functions having range $[0, 1]$, it can be made to work for functions with range $[0, M]$ by scaling everything by M. But this would blow up the query complexity in ℓ_2 setting to $2^{\tilde{O}(M^2/\epsilon^2)}$. It will be good to have a tester for submodularity whose query complexity has a polynomial dependence on $\frac{1}{\epsilon}$ so that the blow up will not be that much for functions with range $[0, M]$.

References

[1] Ashwinkumar Badanidiyuru, Shahar Dobzinski, Hu Fu, Robert Kleinberg, Noam Nisan, and Tim Roughgarden. Sketching valuation functions. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1025–1035, 2012.

[2] Maria-Florina Balcan, Florin Constantin, Satoru Iwata, and Lei Wang. Learning valuation functions. In *COLT 2012 - The 25th Annual Conference on Learning Theory, June 25-27, 2012, Edinburgh, Scotland*, pages 4.1–4.24, 2012.

[3] Maria-Florina Balcan and Nicholas J. A. Harvey. Learning submodular functions. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 793–802, 2011.

[4] Piotr Berman, Sofya Raskhodnikova, and Grigory Yaroslavtsev. L_p-testing. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 164–173, 2014.

[5] Eric Blais. Improved bounds for testing juntas. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008, Boston, MA, USA, August 25-27, 2008. Proceedings*, pages 317–330, 2008.

[6] Eric Blais. Testing juntas nearly optimally. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 151–158, 2009.

[7] Eric Blais and Abhinav Bommireddi. Testing submodularity and other properties of valuation functions. *CoRR*, abs/1611.07879, 2016.

[8] Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. *Computational Complexity*, 21(2):311–358, 2012.

[9] Eric Blais, Clément L. Canonne, Talya Eden, Amit Levi, and Dana Ron. Tolerant junta testing and the connection to submodular optimization and function isomorphism. *CoRR*, abs/1607.03938, 2016.

[10] Eric Blais, Amit Weinstein, and Yuichi Yoshida. Partially symmetric functions are efficiently isomorphism testable. *SIAM J. Comput.*, 44(2):411–432, 2015.

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

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

[13] J. Bourgain. On the distribution of the fourier spectrum of boolean functions. *Israel Journal of Mathematics*, 131(1):269–276, Dec 2002.

[14] Niv Buchbinder and Moran Feldman. Deterministic algorithms for submodular maximization problems. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 392–403, 2016.

[15] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM J. Comput.*, 44(5):1384–1402, 2015.

[16] Deeparnab Chakrabarty and Zhiyi Huang. Testing coverage functions. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, pages 170–181, 2012.

[17] Deeparnab Chakrabarty and Zhiyi Huang. Recognizing coverage functions. *SIAM Journal on Discrete Mathematics*, 29(3):1585–1599, 2015.

[18] Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. Subquadratic submodular function minimization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1220–1231, 2017.

[19] Sourav Chakraborty, Eldar Fischer, David García-Soriano, and Arie Matsliah. Junto-symmetric functions, hypergraph isomorphism and crunching. In *Proceedings of the 27th Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012*, pages 148–158, 2012.

[20] Sourav Chakraborty, David García-Soriano, and Arie Matsliah. Efficient sample extractors for juntas with applications. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, pages 545–556, 2011.

[21] Hana Chockler and Dan Gutfreund. A lower bound for testing juntas. *Inf. Process. Lett.*, 90(6):301–305, 2004.

[22] William H. Cunningham. On submodular function minimization. *Combinatorica*, 5(3):185–192, 1985.

[23] Ilias Diakonikolas, Homin K. Lee, Kevin Matulef, Krzysztof Onak, Ronitt Rubinfeld, Rocco A. Servedio, and Andrew Wan. Testing for concise representations. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 549–558, 2007.

[24] Ilias Diakonikolas, Homin K. Lee, Kevin Matulef, Rocco A. Servedio, and Andrew Wan. Efficiently testing sparse GF(2) polynomials. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, pages 502–514, 2008.

[25] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162:2005, 2004.

[26] Uriel Feige. On maximizing welfare when utility functions are subadditive. *SIAM J. Comput.*, 39(1):122–142, 2009.

[27] Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM J. Comput.*, 40(4):1133–1153, 2011.

[28] Vitaly Feldman and Pravesh Kothari. Learning coverage functions and private release of marginals. In *Proceedings of The 27th Conference on Learning Theory, COLT 2014, Barcelona, Spain, June 13-15, 2014*, pages 679–702, 2014.

[29] Vitaly Feldman and Jan Vondrák. Tight bounds on low-degree spectral concentration of submodular and XOS functions. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 923–942, 2015.

[30] Vitaly Feldman and Jan Vondrák. Optimal bounds on approximation of submodular and XOS functions by juntas. *SIAM J. Comput.*, 45(3):1129–1170, 2016.

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

[32] David Freedman. A remark on the difference between sampling with and without replacement. *Journal of the American Statistical Association*, 72(359):681–681, 1977.

[33] Ehud Friedgut. On the measure of intersecting families, uniqueness and stability. *Combinatorica*, 28(5):503–528, 2008.

[34] Michel X. Goemans, Nicholas J. A. Harvey, Satoru Iwata, and Vahab S. Mirrokni. Approximating submodular functions everywhere. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 535–544, 2009.

[35] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

[36] Parikshit Gopalan, Ryan O'Donnell, Rocco A. Servedio, Amir Shpilka, and Karl Wimmer. Testing fourier dimensionality and sparsity. *SIAM J. Comput.*, 40(4):1075–1100, 2011.

[37] Martin Grötschel, László Lovász, and Alexander Schrijver. Corrigendum to our paper "the ellipsoid method and its consequences in combinatorial optimization". *Combinatorica*, 4(4):291–295, 1984.

[38] David Guijarro, Jun Tarui, and Tatsuie Tsukiji. Finding relevant variables in PAC model with membership queries. In *Algorithmic Learning Theory, 10th International Conference, ALT '99, Tokyo, Japan, December 6-8, 1999, Proceedings*, page 313, 1999.

[39] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. *SIAM J. Comput.*, 22(6):1331–1348, 1993.

[40] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1049–1065, 2015.

[41] Benny Lehmann, Daniel J. Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, 2006.

[42] Kevin Matulef, Ryan O'Donnell, Ronitt Rubinfeld, and Rocco A. Servedio. Testing halfspaces. *SIAM J. Comput.*, 39(5):2004–2047, 2010.

[43] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.

[44] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

[45] Michal Parnas, Dana Ron, and Ronitt Rubinfeld. On testing convexity and submodularity. *SIAM J. Comput.*, 32(5):1158–1184, 2003.

[46] Michal Parnas, Dana Ron, and Alex Samorodnitsky. Testing basic boolean formulae. *SIAM J. Discrete Math.*, 16(1):20–46, 2002.

[47] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[48] Rocco A. Servedio. Testing by implicit learning: A brief survey. In *Property Testing - Current Research and Surveys*, pages 197–210, 2010.

[49] C. Seshadhri and Jan Vondrák. Is submodularity testable? In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 195–210, 2011.

[50] Leslie G. Valiant. A theory of the learnable. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, pages 436–445, 1984.

[51] Karl Wimmer and Yuichi Yoshida. Testing linear-invariant function isomorphism. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 840–850, 2013.