Learning Economic Parameters from Revealed Preferences

Maria-Florina Balcan[1] Amit Daniely[2] Ruta Mehta[3] Ruth Urner[1]
Vijay V. Vazirani[3]

[1]Department of Machine Learning, Carnegie Mellon University
ninamf@cs.cmu.edu, rurner@cs.cmu.edu
[2]Department of Mathematics, The Hebrew University
amit.daniely@mail.huji.ac.il
[3]School of Computer Science, Georgia Institute of Technology
rmehta@cc.gatech.edu, vazirani@cc.gatech.edu

Abstract

A recent line of work, starting with Beigman and Vohra [3] and Zadimoghaddam and Roth [28], has addressed the problem of *learning* a utility function from revealed preference data. The goal here is to make use of past data describing the purchases of a utility maximizing agent when faced with certain prices and budget constraints in order to produce a hypothesis function that can accurately forecast the *future* behavior of the agent.

In this work we advance this line of work by providing sample complexity guarantees and efficient algorithms for a number of important classes. By drawing a connection to recent advances in multi-class learning, we provide a computationally efficient algorithm with tight sample complexity guarantees ($\Theta(d/\epsilon)$ for the case of d goods) for learning linear utility functions under a linear price model. This solves an open question in Zadimoghaddam and Roth [28]. Our technique yields numerous generalizations including the ability to learn other well-studied classes of utility functions, to deal with a misspecified model, and with non-linear prices.

Keywords: revealed preference, statistical learning, query learning, efficient algorithms, Linear, SPLC, CES and Leontief utility functions

1 Introduction

A common assumption in Economics is that agents are utility maximizers, meaning that the agent, facing prices, will choose to buy the bundle of goods that she most prefers among all bundles that she can afford, according to some concave, non-decreasing utility function [20]. In the classical "revealed preference" analysis [27], the goal is to produce a model of the agent's utility function that can explain her behavior based on past data. Work on this topic has a long history in economics [25, 18, 19, 13, 22, 1, 26, 9, 14, 10], beginning with the seminal work by Samuelson (1948) [23]. Traditionally, this work has focused on the "rationalization" or "fitting the sample" problem, in which explanatory utility functions are constructively generated from finitely many agent price/purchase observations. For example, the seminal work of Afriat [1] showed (via an algorithmic construction) that any finite sequence of observations is rationalizable if and only if it is rationalizable by a piecewise linear, monotone, concave utility function. Note, however, that just because a function agrees with a set of data does not imply that it will necessarily predict future purchases well.

A recent exciting line of work, starting with Beigman and Vohra [3] introduced a statistical learning analysis of the problem of learning the utility function from past data with the explicit formal goal of having predictive or forecasting properties. The goal here is to make use of the observed data describing the behavior of the agent (i.e., the bundles the agent bought when faced with certain prices and budget constraints) in order to produce a hypothesis function that can accurately predict or forecast the *future* purchases of a utility maximizing agent. [3] show that without any other assumptions on utility besides monotonicity and concavity, the sample complexity of learning (in a statistical or probably approximately correct sense) a demand and hence utility function is infinite. This shows the importance of focusing on important sub-classes since fitting just any monotone, concave function to the data will not be predictive for future events.

Motivated by this, Zadimoghaddam and Roth [28] considered specific classes of utility functions including the commonly used class of linear utilities. In this work, we advance this line of work by providing sample complexity guarantees and efficient algorithms for a number of important classical classes (including linear, separable piecewise-linear concave (SPLC), CES and Leontief [20]), significantly expanding the cases where we have strong learnability results. At a technical level, our work establishes connections between learning from revealed preferences and problems of multi-class learning, combining recent advances on intrinsic sample complexity of multi-class learning based on compression schemes [8] with a new algorithmic analysis yielding time- and sample-efficient procedures. We believe that this technique may apply to a variety of learning problems in economic and game theoretic contexts.

1.1 Our Results

For the case of linear utility functions, we establish a connection to the so-called structured prediction problem of *D-dimensional linear classes* in theoretical machine learning (see e.g., [5, 6, 15]). By using and improving very recent results of [8], we provide a computationally efficient algorithm with *tight* sample complexity guarantees for learning linear utility functions under a linear price model (i.e., additive over goods) for the statistical revealed preference setting. This improves over the bound in Zadimoghaddam and Roth [28] by a factor of d and resolves their open question concerning the right sample complexity of this problem. In addition to noting that we can actually fit the types of problems stemming from revealed preference in the structured prediction framework of Daniely and Shalev-Shwartz [8], we also provide a much more efficient and practical algorithm for this learning problem. We specifically show that we can reduce their compression based technique to a classic SVM problem which can be solved via convex programming[1]. This latter result could be of independent interest to Learning Theory.

The connection to the structured prediction problem with D-dimensional linear classes is quite powerful and it yields numerous generalizations. It immediately implies strong sample complexity guarantees (though not necessary efficient algorithms) for other important revealed preference settings. For linear utility functions we can deal with non-linear prices (studied for example in [16]), as well as with a misspecified model — in learning theoretic terms this means the agnostic setting where the target function is consistent with a linear utility function on a $1-\eta$ fraction of bundles; furthermore, we can also accommodate non-divisible goods. Other classes of utility functions including SPLC and CES can be readily analyzed in this framework as well.

We additionally study *exact* learning via revealed preference queries: here the goal of the learner is to determine the underlying utility function exactly, but it has more power since it can choose instances (i.e., prices and budgets) of its own choice and obtain the labels (i.e., the bundles the buyer buys). We carefully exploit the structure of the optimal solution (which can be determined based on the KKT conditions) in order to design query efficient algorithms. This could be relevant for scenarios where sellers/manufacturers with many different products have the ability to explicitly set desired prices for exploratory purposes (e.g., with the goal to be able to predict how demands change with change in prices of different goods, so that they can price their goods optimally).

As a point of comparison, for both statistical and the query setting, we also analyze learning classes of utility functions directly (from utility values instead of from revealed preferences). Table 1.1 summarizes our sample complexity bounds for learning from revealed preferences (RP) and from utility values (Value) as well as our corresponding bounds on the query complexity (in the table we omit log-factors). Previously known results are indicated with a $*$.

	RP, Statistical	**RP, Query**	**Value, Statistical**	**Value, Query**
Linear	$\Theta(d/\epsilon)$	$O(nd)$	$O(d/\epsilon)^*$	$O(d)^*$
SPLC (at most κ segments per good)	$O(\kappa d/\epsilon)$ (known segment lengths)	$O(n\kappa d)$	?	$O(n\kappa d)$
CES	$O(d/\epsilon)$ (known ρ)	$O(1)$	$O(d/\epsilon)$ (known ρ)	$O(d)$
Leontief	$O(1)$	$O(1)$	$O(d/\epsilon)$	$O(d)$

Table 1: Markets with d goods, and parameters of (bit-length) size n

2 Preliminaries

Following the framework of [28], we consider a market that consists of a set of agents (buyers), and a set $\mathcal{G}$ of d goods of unit amount each. The prices of the goods are indicated by a price vector $\boldsymbol{p} = (p_1, \ldots, p_d)$. A buyer comes with a budget of money, say B, and intrinsic preferences over bundles of goods. For most of the paper we focus on divisible goods. A bundle of goods is represented by a vector $\boldsymbol{x} = (x_1, \ldots, x_d) \in [0,1]^d$, where the i-th component x_i denotes the amount of the i-th good in the bundle. The price of a bundle is computed as the inner product $\langle \boldsymbol{p}, \boldsymbol{x} \rangle$. Then the preference over bundles of an agent is defined by a non-decreasing, non-negative and concave utility function $U : [0,1]^d \to \mathbb{R}_+$. The buyer uses her budget to buy a bundle of goods that maximizes her utility.

[1]Such an algorithm has been used in the context of revealed preferences in a more applied work of [16]; but we prove correctness and tight sample complexity.

In the revealed preference model, when the buyer is provided with $(\boldsymbol{p}, B)$, we observe the optimal bundle that she buys. Let this optimal bundle be denoted by $\mathcal{B}_U(\boldsymbol{p}, B)$, which is an optimal solution of the following optimization problem:

$$\begin{array}{lll} \arg\max_{\boldsymbol{x}\in[0,\ 1]^d} & : & U(\boldsymbol{x}) \\ s.t. & & \langle \boldsymbol{p}, \boldsymbol{x}\rangle \le B \end{array} \tag{1}$$

We assume that if there are multiple optimal bundles, then the buyer will choose a cheapest one, i.e., let $S = \arg\max_{\boldsymbol{x}\in[0,\ 1]^d} U(\boldsymbol{x})$ at $(\boldsymbol{p}, B)$, then $\mathcal{B}_U(\boldsymbol{p}, B) \in \arg\min_{\boldsymbol{x}\in S}\langle \boldsymbol{x}, \boldsymbol{p}\rangle$. Furthermore, if there are multiple optimal bundles of the same price, ties a broken according to some rule (e.g. , the buyer prefers lexicographically earlier bundles).

Demand functions While a utility function U, by definition, maps bundles to values, it also defines a mapping from pairs $(\boldsymbol{p}, B)$ of price vectors and budgets to an optimal bundles under U. We denote this function by $\widehat{U}$ and call it the *demand function* corresponding to the utility function U. That is, we have $\widehat{U} : \mathbb{R}^d_+ \times \mathbb{R}_+ \to [0, 1]^d$, and $\widehat{U}(\boldsymbol{p}, B) = \mathcal{B}_U(\boldsymbol{p}, B)$. For a class of utility function $\mathcal{H}$ we denote the corresponding class of demand functions by $\widehat{\mathcal{H}}$.

2.1 Classes of utility functions

Next we discuss four different types of utility functions that we analyze in this paper, namely linear, SPLC, CES and Leontief [20], and define their corresponding classes formally. Note that at given prices $\boldsymbol{p}$ and budget B, $\mathcal{B}_U(\boldsymbol{p}, B) = \mathcal{B}_{\alpha U}(\boldsymbol{p}, B)$, for all $\alpha > 0$, i.e., positive scaling of utility function doesn't affect optimal bundles. Since we are interested in learning U by asking queries to $\mathcal{B}_U$ we will make some normalizing assumptions in the following definitions. We start with the simplest and the most studied class of functions, namely linear utilities.

Definition 1 (Linear $\mathcal{H}_{lin}$) *A utility function U is called linear if the utility from a bundle $\boldsymbol{x}$ is linear in each good. Formally, for some $\boldsymbol{a} \in \mathbb{R}^d_+$, we have $U(\boldsymbol{x}) = U_{\boldsymbol{a}}(x) = \sum_{j\in\mathcal{G}} a_j x_j$. It is wlog to assume that $\sum_j a_j = 1$. We let $\mathcal{H}_{lin}$ denote the class of linear utility functions.*

Next, is a generalization of linear functions that captures decreasing marginal utility, called separable piecewise-linear concave.

Definition 2 (Separable Piecewise-Linear Concave (SPLC) $\mathcal{H}_{splc}$) *A utility function function U is called SPLC if, $U(\boldsymbol{x}) = \sum_{j\in\mathcal{G}} U_j(x_j)$ where each $U_j : \mathbb{R}_+ \to \mathbb{R}_+$ is non-decreasing piecewise-linear concave function. The number of (pieces) segments in U_j is denoted by $|U_j|$ and the k^{th} segment of U_j denoted by (j, k). The slope of a segment specifies the rate at which the agent derives utility per unit of additional good received. Suppose segment (j, k) has domain $[a, b] \subseteq \mathbb{R}_+$, and slope c. Then, we define $a_{jk} = c$ and $l_{jk} = b - a$; $l_{j|U_j|} = \infty$). Since U_j is concave, we have $a_{j(k-1)} > a_{jk}$, $\forall k \ge 2$. We can view an SPLC function, with $|U_j| \le \kappa$ for all j, as defined by to matrices $\boldsymbol{A}, \boldsymbol{L} \in \mathbb{R}^{d\times\kappa}_+$ and we denote it by $U_{\boldsymbol{AL}}$. We let $\mathcal{H}_{splc}$ denote the class of all SPLC functions.*

Linear and SPLC functions are applicable when goods are substitutes, i.e., one good can be replaced by another to maintain a utility value. The other extreme is when goods are complementary, i.e., all goods are needed in some proportions to obtain non-zero utility. Next, we describe a class of functions, used extensively in economic literature, that captures both substituteness and complementarity in different ranges.

Definition 3 (Constant elasticity of substitution (CES) $\mathcal{H}_{ces}$) *A utility function U is called CES if for some $-\infty < \rho \leq 1$, and $\boldsymbol{a} \in \mathbb{R}_+^d$ we have $U(\boldsymbol{x}) = U_{\boldsymbol{a}\rho}(\boldsymbol{x}) = (\sum_j a_j x_j^\rho)^{1/\rho}$. Again it is wlog to assume that $\sum_j a_j = 1$. Let $\mathcal{H}_{ces}$ be the set of all CES functions. Further, for some fixed ρ, we let $\mathcal{H}_{ces}^\rho$ denote the subclass of functions with parameter ρ.*

Note that if $\rho = 1$ for some CES function, then the function is linear, that is $\mathcal{H}_{ces}^1 = \mathcal{H}_{lin}$. Further, under CES functions with $\rho > 0$, the goods behave as substitutes. However, for $\rho \leq 0$, they behave as complements, i.e., if an $x_j = 0$ while $a_j > 0$ the utility derived remains zero, regardless of how much amounts of other goods are given. As $\rho \to -\infty$, we get Leontief function at the limit where goods are completely complementary, i.e., a set of goods are needed in a specific proportion to derive any utility.

Definition 4 (Leontief $\mathcal{H}_{leon}$) *A utility function U is called a Leontief function if $U(\boldsymbol{x}) = \min_{j \in \mathcal{G}} x_j/a_j$, where $\boldsymbol{a} \geq 0$ and (wlog) $\sum_j a_j = 1$. Let $\mathcal{H}_{leon}$ be the set of all Leontief functions on d goods.*

In order to work with finite precision, in all the above definition we assume that the parameters defining the utility functions are rational numbers of (bit-length) size at most n.

2.2 Learning models: Statistical & Query

We now introduce the formal models under which we analyze the learnability of utility functions. We start by reviewing the general model from statistical learning theory for multi-class classification. We then explain the more specific model for learning from revealed preferences as introduced in [28]. Finally, we also consider a non-statistical model of exact learning from queries, which is explained last in this section.

General model for statistical multi-class learning Let $\mathcal{X}$ denote a *domain* set and let $\mathcal{Y}$ denote a *label* set. A *hypothesis* (or *label predictor* or *classifier*), is a function $h : \mathcal{X} \to \mathcal{Y}$, and a *hypothesis class* $\mathcal{H}$ is a set of hypotheses. We assume that data is generated by some (unknown) probability distribution P over $\mathcal{X}$. This data is labeled by some (unknown) *labeling function* $l : \mathcal{X} \to \mathcal{Y}$. The quality of a hypothesis h is measured by its *error* with respect to P and l:

$$\mathrm{err}_P^l(h) = \Pr_{x \sim P}[l(x) \neq h(x)],$$

A *learning algorithm* (or *learner*) gets as input a sequence $S = ((x_1, y_1), \ldots, (x_m, y_m))$ and outputs a hypothesis.

Definition 5 (Multi-class learnability (realizable case)) *We say that an algorithm $\mathcal{A}$* learns *some hypothesis class $\mathcal{H} \subseteq \mathcal{Y}^\mathcal{X}$, if there exists a function $m : (0,1) \times (0,1) \to \mathbb{N}$ such that, for all distributions P over $\mathcal{X}$, and for all $\epsilon > 0$ and $\delta > 0$, when given a sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$ of size at least $m = m(\epsilon, \delta)$ with the x_i generated* i.i.d. *from P and $y_i = h(x)$ for some $h \in \mathcal{H}$, then, with probability at least $1 - \delta$ over the sample, $\mathcal{A}$ outputs a hypothesis $h_\mathcal{A} : \mathcal{X} \to \mathcal{Y}$ with $\mathrm{err}_P^h(h_\mathcal{A}) \leq \epsilon$.*

The complexity of a learning task is measured by its *sample complexity*, that is, informally, the amount of data with which an optimal learner can achieve low error. We call the (point-wise) smallest function $m : (0,1) \times (0,1) \to \mathbb{N}$ that satisfies the condition of Definition 5 the *sample complexity of the algorithm $\mathcal{A}$ for learning $\mathcal{H}$*. We denote this function by $m[\mathcal{A}, \mathcal{H}]$. We call the smallest function $m : (0,1) \times (0,1) \to \mathbb{N}$ such that there exists a learning algorithm $\mathcal{A}$ with $m[\mathcal{A}, \mathcal{H}] \leq m$ the *sample complexity of learning H* and denote it by $m[\mathcal{H}]$.

Statistical learning from revealed preferences As in [28], we consider a statistical learning setup where data is generated by a distribution P over pairs of price vectors and budgets (that is, P is a distribution over $\mathbb{R}_+^d \times \mathbb{R}_+$). In this model, a learning algorithm $\mathcal{A}$ gets as input a sample $S = (((\boldsymbol{p}_1, B_1), \mathcal{B}_U(\boldsymbol{p}_1, B_1)), \ldots, ((\boldsymbol{p}_m, B_m), \mathcal{B}_U(\boldsymbol{p}_m, B_m)))$, where the $(\boldsymbol{p}_i, B_i)$ are generated *i.i.d.* from the distribution P and are labeled by the optimal bundles under some utility function U. It outputs some function $\mathcal{A}(S) : \mathbb{R}_+^d \times \mathbb{R}_+ \to [0,1]^d$ that maps pairs of price vectors and budgets to bundles. A learner is considered successful if it learns to predict a bundle of value that is the optimal bundles' value.

Definition 6 (Learning from revealed preferences) *An algorithm $\mathcal{A}$ is said to* learn a class of utility functions $\mathcal{H}$ from revealed preferences*, if for all $\epsilon, \delta > 0$, there exists a sample size $m = m(\epsilon, \delta) \in \mathbb{N}$, such that, for any distribution P over $\mathbb{R}_+^d \times \mathbb{R}_+$ (pairs of price vectors and budgets) and any target utility function $U \in \mathcal{H}$, if $S = (((\boldsymbol{p}_1, B_1), \mathcal{B}_U(\boldsymbol{p}_1, B_1)), \ldots, ((\boldsymbol{p}_m, B_m), \mathcal{B}_U(\boldsymbol{p}_m, B_m)))$ is a sequence of i.i.d. samples generated by P with U, then, with probability at least $1 - \delta$ over the sample S, the output utility function $\mathcal{A}(S)$ satisfies*

$$\Pr_{(\boldsymbol{p},B)\sim P}\left[U(\mathcal{B}_U(\boldsymbol{p}, B)) \neq U(\mathcal{B}_{\mathcal{A}(S)}(\boldsymbol{p}, B))\right] \leq \epsilon.$$

Note that the above learning requirement is satisfied if the learner "learns to output the correct optimal bundles". That is, to learn a class $\mathcal{H}$ of utility functions from revealed preferences, in the sense of Definition 6, it suffices to learn the corresponding class of demand functions $\widehat{\mathcal{H}}$ in the standard sense of Definition 5 (with $\mathcal{X} = \mathbb{R}_+^d \times \mathbb{R}_+$ and $\mathcal{Y} = [0,1]^d$). This is what the algorithm in [28] and our learning algorithms for this setting actually do. The notion of sample complexity in this setting can be defined analogously to the definition above.

2.2.1 Model for exact learning from queries

In the query learning model, the goal of the learner is to determine the underlying utility function exactly. The learner can choose instances and obtain the labels of these instances from some oracle. A *revealed preference query learning algorithm* has access to an oracle that, upon given the input (query) of a price vector and a budget $(\boldsymbol{p}, B)$, outputs the corresponding optimal bundle $\mathcal{B}_U(\boldsymbol{p}, B)$ under some utility function U. Slightly abusing notation, we also denote this oracle by $\mathcal{B}_U$.

Definition 7 (Learning from revealed preference queries) *A learning algorithm* learns a class $\mathcal{H}$ from m revealed preference queries*, if for any function $U \in \mathcal{H}$, if the learning algorithm is given responses from oracle $\mathcal{B}_U$, then after at most m queries the algorithm outputs the function U.*

Both in the statistical and the query setting, we analyze a revealed preference learning model as well as a model of learning classes of utility function "directly" from utility values. Due to limited space, these latter definition and results have been moved to the Appendix, Sections D and E.

3 Efficiently learning linear multi-class hypothesis classes

We start by showing that certain classes of multi-class predictors, so-called *D-dimensional linear classes* (see Definition 8 below), can be learnt efficiently both in terms of their sample complexity and in terms of computation. For this, we make use of a very recent upper bound on their sample complexity by Daniely and Shalev-Shwartz [8]. At a high level, their result obtains strong bounds on the sample complexity of D-dimensional linear classes (roughly D/ϵ) by using an algorithm and sample complexity analysis based on a compression scheme — which roughly means that the hypothesis produced can be uniquely described by a small subset of D of the training examples. We show that their algorithm is actually equivalent to a multi class SVM formulation, and thereby obtain a computationally efficient algorithm with optimal sample complexity. In the next sections we then show how learning classes of utility functions from revealed preferences can be cast in this framework.

Definition 8 *A hypothesis classes $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ is a D-*dimensional linear class*, if there exists a function $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ such that for every $h \in \mathcal{H}$, there exists a vector $\boldsymbol{w} \in \mathbb{R}^D$ such that $h(x) \in \arg\max_{y \in \mathcal{Y}} \langle \boldsymbol{w}, \Psi(x, y) \rangle$ for all $x \in \mathcal{X}$. We then also denote the class by $\mathcal{H}_\Psi$ and its members by $h_{\boldsymbol{w}}$.*

For now, we assume that (the data generating distribution is so that) the set $\arg\max_{y \in \mathcal{Y}} \langle \boldsymbol{w}, \Psi(x, y) \rangle$ contains only one element, that is, there are no ties[2]. The following version of the multi-class support vector machine (SVM) has been introduced by Crammer and Singer [7].

Algorithm 1 Multi-class (hard) SVM [7]

Input: Sample $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m) \in \mathcal{X} \times \mathcal{Y}$
Solve: $\boldsymbol{w} = \arg\min_{\boldsymbol{w} \in \mathbb{R}^d} \|\boldsymbol{w}\|$
such that $\langle \boldsymbol{w}, \Psi(\boldsymbol{x}_i, y_i) - \Psi(\boldsymbol{x}_i, y) \rangle \geq 1 \quad \forall i \in [m], y \neq y_i$
Return: vector $\boldsymbol{w}$

Remark 9 *Suppose that given $\boldsymbol{w} \in \mathbb{R}^d$ and $x \in \mathcal{X}$, it is possible to efficiently compute some $y' \in \mathrm{argmax}_{y \neq \mathrm{argmax}_{y''} \langle \boldsymbol{w}, \Psi(x, y'') \rangle} \langle \boldsymbol{w}, \Psi(x, y) \rangle$. That is, it is possible to compute a label y in the set of "second best" labels. In that case, it is not hard to see that SVM can be solved efficiently. The reason is that this gives a separation oracle. SVM minimizes a convex objective subject to, possibly exponentially many, linear constraints. For a given $\boldsymbol{w}$, a violated constraint can be efficiently detected (by one scan over the input sample) by observing that $\langle \boldsymbol{w}, \Psi(x_i, y_i) - \Psi(x_i, y') \rangle < 1$.*

The following theorem on the sample complexity of the above SVM formulation, is based on the new analysis of linear classes by [8]. We show that the two algorithms (the SVM and the one in [8]) are actually the same.

Theorem 10 *Let $\mathcal{H}_\Psi$ be some D-dimensional linear class. Then the sample complexity of SVM for $\mathcal{H}_\Psi$ satisfies $m[SVM, \mathcal{H}_\Psi](\epsilon, \delta) = O\left(\frac{D \log(1/\epsilon) + \log(1/\delta)}{\epsilon}\right)$.*

Proof : Let $S = (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)$ be a sample that is realized by $\mathcal{H}_\Psi$. That is, there exists a vector $\boldsymbol{w} \in \mathbb{R}^d$ with $\langle \boldsymbol{w}, \Psi(\boldsymbol{x}_i, y_i) \rangle > \langle \boldsymbol{w}, \Psi(\boldsymbol{x}_i, y) \rangle$ for all $y \neq y_i$. Consider the set $Z = \{\Psi(x_i, y_i) - \Psi(x_i, y) \mid i \in [m],\ y \neq y_i\}$. The learning algorithm for $\mathcal{H}_\Psi$ of [8] outputs the minimal norm vector $\boldsymbol{w}' \in \mathrm{conv}(Z)$. According to Theorem 5 in [8] this algorithm successfully learns $\mathcal{H}_\Psi$ and has sample complexity $O\left(\frac{D \log(1/\epsilon) + \log(1/\delta)}{\epsilon}\right)$. We will show that the hypothesis returned by that algorithm is the same hypothesis as the one returned by SVM. Indeed, let $\boldsymbol{w}$ be the vector that solves the SVM program and let $\boldsymbol{w}'$ be the vector found by the algorithm of [8]. We will show that $\boldsymbol{w} = \frac{\|\boldsymbol{w}\|}{\|\boldsymbol{w}'\|} \cdot \boldsymbol{w}'$. This is enough since in that case $h_{\boldsymbol{w}} = h_{\boldsymbol{w}'}$.

[2]The work of [8] handled ties using a "don't know" label; to remove technicalities, we make this distributional assumption in this version of our work

We note that $\boldsymbol{w}$ is the same vector that solves the *binary* SVM problem defined by the sample $\{(z, 1)\}_{z\in Z}$. It well known (see, e.g. , [24], Lemma 15.2) that the hyperplane defined by $\boldsymbol{w}$ has maximal margin. That is, the unit vector $\boldsymbol{e} = \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$ maximizes the quantity

$$\mathrm{mar}(\boldsymbol{e}'') := \min\{\langle \boldsymbol{e}'', \boldsymbol{z}\rangle \mid \boldsymbol{z} \in Z\}$$

over all unit vectors $\boldsymbol{e}'' \in S^d$. The proof of the theorem now follows from the following claim:

Claim 11 *Over all unit vectors,* $\boldsymbol{e}' = \frac{\boldsymbol{w}'}{\|\boldsymbol{w}'\|}$ *maximizes the margin.*

Proof : Let $\boldsymbol{e}'' \neq \boldsymbol{e}'$ be a unit vector. We must show that $\mathrm{margin}(\boldsymbol{e}'') < \mathrm{margin}(\boldsymbol{e}')$. Note that $\mathrm{margin}(\boldsymbol{e}') > 0$, since $\boldsymbol{w}'$ is shown in [8] to realize the sample S (that is $\langle \boldsymbol{w}, \boldsymbol{z}\rangle > 0$ for all $\boldsymbol{z} \in Z$ and thus also for all $\boldsymbol{z} \in \mathrm{conv}Z$). Therefore, we can assume w.l.o.g. that $\mathrm{margin}(\boldsymbol{e}'') > 0$. In particular, since $\mathrm{margin}(-\boldsymbol{e}') = -\mathrm{margin}(\boldsymbol{e}') < 0$, we have that $\boldsymbol{e}'' \neq -\boldsymbol{e}'$.

Since, $\mathrm{margin}(\boldsymbol{e}'') > 0$, we have that $\mathrm{margin}(\boldsymbol{e}'')$ is the distance between the hyperplane $H'' = \{\boldsymbol{x} \mid \langle \boldsymbol{e}'', \boldsymbol{x}\rangle = 0\}$ and $\mathrm{conv}(Z)$. Since $\boldsymbol{e}'' \notin \{\boldsymbol{e}', -\boldsymbol{e}'\}$, there is a vector in $\boldsymbol{v} \in H''$ with $\langle \boldsymbol{e}', \boldsymbol{v}\rangle \neq 0$. Now, consider the function

$$t \mapsto \|t \cdot \boldsymbol{v} - \boldsymbol{w}'\|^2 = t^2 \cdot \|\boldsymbol{v}\|^2 + \|\boldsymbol{w}'\|^2 - 2t\langle \boldsymbol{v}, \boldsymbol{w}'\rangle.$$

Since the derivative of this function at 0 is not 0, for some value of t we have $\mathrm{dist}(t \cdot \boldsymbol{v}, \boldsymbol{w}') < \mathrm{dist}(0, \boldsymbol{w}')$. Therefore, $\mathrm{margin}(\boldsymbol{e}'') = \mathrm{dist}(H'', Z) \leq \mathrm{dist}(t \cdot \boldsymbol{v}, \boldsymbol{w}') < \mathrm{dist}(0, \boldsymbol{w}') = \mathrm{margin}(\boldsymbol{e}')$. $\square$

$\square$

4 Statistical learning from revealed preferences

In the next section, we show that learning utility functions from revealed preferences can in many cases be cast as learning a D-dimensional linear class $\mathcal{H}_\Psi$ for a suitable encoding function Ψ and D. Throughout this section, we assume that the data generating distribution is so that there are no ties for the optimal bundle with respect to the agents' utility function (with probability 1). This is, for example, the case if the data-generating distribution has a density function.

4.1 Linear

Learnability of $\mathcal{H}_{lin}$ from revealed preferences is analyzed in [28]. They obtain a bound of (roughly) d^2/ϵ on the sample complexity. We show that the quadratic dependence on the number of goods is not needed. The sample complexity of this problem is (roughly) d/ϵ.

We will show that the corresponding class of demand functions $\widehat{\mathcal{H}_{lin}}$ is actually a d-dimensional linear class. Since learnability of a class of utility functions in the revealed preference model (Definition 6) is implied by learnability of the corresponding class of demand functions (in the sense of Definition 5), Theorem 10 then implies the upper bound in the following result:

Theorem 12 *The class* $\mathcal{H}_{lin}$ *of linear utility functions is efficiently learnable in the revealed preference model with sample complexity* $O\left(\frac{d\log(1/\epsilon)+\log(1/\delta)}{\epsilon}\right)$. *Moreover, the sample complexity is lower bounded by* $\Omega\left(\frac{(d-1)+\log(1/\delta)}{\epsilon}\right)$.

Proof : Let $U_{\boldsymbol{a}}$ be a linear utility function. By definition, the optimal bundle given a price vector $\boldsymbol{p}$ and a budget B is $\arg\max_{\boldsymbol{x}\in[0,1]^n, \langle \boldsymbol{p},\boldsymbol{x}\rangle\le B}\langle \boldsymbol{a},\boldsymbol{x}\rangle$. Note that, for a linear utility function, there is always an optimal bundle $\boldsymbol{x}$ where all (except at most one) of the x_i are in $\{0,1\}$ (this was also observed in [28]; see also Section 5.1). Essentially, given a price vector $\boldsymbol{p}$, in an optimal bundle, the goods are bought greedily in decreasing order of a_i/p_i (value per price).

Thus, given a pair of price vector and budget $(\boldsymbol{p}, B)$, we call a bundle $\boldsymbol{x}$ *admissible*, if $|\{i : x_i \notin \{0,1\}\}| \le 1$ and $\langle \boldsymbol{p},\boldsymbol{x}\rangle = B$. In case $\langle \boldsymbol{p},\mathbf{1}_d\rangle = \sum_{i\in\mathcal{G}} p_i \le B$, we also call the all 1-bundle $\mathbf{1}_d$ admissible (and in this case, it is the only admissible bundle). We now define the function Ψ as follows:

$$\Psi((\boldsymbol{p},B),\boldsymbol{x}) = \begin{cases} \boldsymbol{x} & \text{if } \boldsymbol{x} \text{ admissible} \\ \mathbf{0}_d & \text{otherwise} \end{cases}$$

where $\mathbf{0}_d$ denotes the all-0 vector in $\mathbb{R}^d$. With this, we have $\mathcal{H}_\Psi = \widehat{\mathcal{H}_{lin}}$.

We defer the proof of the lower bound to the Appendix, Section B. To outline the argument, we prove that the Natarajan dimension of $\widehat{\mathcal{H}_{lin}}$ is at least $d-1$ (Lemma 28). This implies a lower bound for learning $\widehat{\mathcal{H}_{lin}}$ (see Theorem 27 in the Appendix). It is not hard to see that the construction also implies a lower bound for learning $\mathcal{H}_{lin}$ in the revealed preference model.

To prove computational efficiency, according to Remark 9, we need to show that for a linear utility function, we can efficiently compute some

$$y' \in \mathrm{argmax}_{y' \notin \mathrm{argmax}_y \langle \boldsymbol{w}, \Psi(x,y)\rangle} \langle \boldsymbol{w}, \Psi(x,y')\rangle;$$

that is a second best bundle with respect to the mapping Ψ. This will be shown in Theorem 16 of the next subsection. □

4.1.1 Efficiently computing the second best bundle under linear utilities

It is known and easy to show (for example using KKT conditions for (1), see Section 5.1) that an optimal bundle for the case of linear utility functions can be computed as follows: Sort the goods in decreasing order of $\frac{a_j}{p_j}$, and keep buying in order until the budget runs out. The number of partially allocated goods in such a bundle is at most one, namely the last one bought in the order.

In this section show how to compute a second best *admissible bundle* (with respect to the mapping Ψ) efficiently. Recall that *admissible bundles* at prices $\boldsymbol{p}$ and budget B are defined (in the proof of the above theorem) to be the bundles that cost exactly B with at most one partially allocated good (or the all-1 bundle $\mathbf{1}_d$, in case it is affordable). Note that, in case $\langle \boldsymbol{p},\mathbf{1}_d\rangle \le B$, *any* other bundle is second best with respect Ψ. For the rest of this section, we assume that $\langle \boldsymbol{p},\mathbf{1}_d\rangle > B$.

At any given $(\boldsymbol{p}, B)$ the optimal bundle is always admissible. We now design an $O(d)$-time algorithm to compute the second best admissible bundle, i.e., $\boldsymbol{y} \in \arg\max_{\boldsymbol{x} \text{ admissible }, \boldsymbol{x}\neq\boldsymbol{x}^*}\langle \boldsymbol{a},\boldsymbol{x}\rangle$, where $\boldsymbol{x}^*$ is the optimal bundle.

At prices $\boldsymbol{p}$, let $\frac{a_1}{p_1} \ge \frac{a_2}{p_2} \ge \cdots \ge \frac{a_d}{p_d}$, and let the first k goods be bought at the optimal bundle, i.e., $k = \max_{j:\, x_j^*>0} j$. Then, clearly $\forall j<k,\ x_j^* = 1$ and $\forall j>k,\ x_j^*=0$ as $\boldsymbol{x}^*$ is admissible.

Note that, to obtain the second best admissible bundle $\boldsymbol{y}$ from $\boldsymbol{x}^*$, amounts of only first k goods can be lowered and amounts of only last k to d goods can be increased. Next we show that the number of goods whose amounts are lowered and increased at exactly one each. In all the proofs we crucially use the fact that if $\frac{a_j}{p_j} > \frac{a_k}{p_k}$, then transferring money from good k to good j gives a better bundle, i.e., $a_j\frac{m}{p_j} - a_k\frac{m}{p_k} > 0$.

Lemma 13 *There exists exactly one $j \ge k$, such that $y_j > x_j^*$.*

Proof : To the contrary suppose there are more than one goods with $y_j > x^*_j$. Consider the last such good, let it be l. Clearly $l > k$, because the first good that can be increased is k. If $y_l < 1$ then there exists $j < l$ with $y_j = 0$, else if $y_l = 1$ then there exists $j < l$ with $y_j < 1$. In either case transfer money from good l to good j such that the resulting bundle is admissible. Since, $\frac{a_j}{p_j} > \frac{a_l}{p_l}$ it is a better bundle different from $\boldsymbol{x}^*$. The latter holds because there is another good whose amount still remains increased. A contradiction to $\boldsymbol{y}$ being second best. □

Lemma 14 *There exists exactly one $j \leq k$, such that $y_j < x^*_j$.*

Proof : To the contrary suppose there are more than one goods with $y_j < x^*_j$. Let l be the good with $y_l > x^*_l$; there is exactly one such good due to Lemma 13. Let i be the first good with $y_i < x^*_i$ and let p be the good that is partially allocated in y. If p is undefined or $p \in \{i, l\}$, then transfer money from l to i. to get a better bundle. Otherwise, $p < l$ so transfer money from l to p. In either case we can do the transfer so that resulting bundle is admissible and is better than $\boldsymbol{y}$ but different from $\boldsymbol{x}^*$. A contradiction. □

Lemmas 13 and 14 gives an $O(d^2)$ algorithm to compute the second best admissible bundle, where we can check all possible way of transferring money from a good in $\{1, \ldots, k\}$ to a good in $\{k, \ldots, d\}$. Next lemma will help us reduce the running time to $O(d)$.

Lemma 15 *If $x^*_k < 1$, and for $j > k$ we have $y_j > x^*_j$, then $y_k < x^*_k$. Further, if $x^*_k = 1$ and $y_j > x^*_j$ then $j = k + 1$.*

Proof : To the contrary suppose, $y_k = x^*_k < 1$ and for a unique $i < k$, $y_i < x^*_i$ (Lemma 14). Clearly, $y_i = 0$ and $y_j = 1$ because $0 < y_k < 1$. Thus, transferring money from j to k until either $y_j = 0$ or $y_k = 1$ gives a better bundle different from $\boldsymbol{x}^*$, a contradiction.

For the second part, note that there are no partially bought good in $\boldsymbol{x}^*$ and $y_{k+1} = 0$. To the contrary suppose $j > k + 1$, then transferring money from good j to good $k + 1$ until either $y_j = 0$ or $y_{k+1} = 1$ gives a better bundle other than $\boldsymbol{x}^*$, a contradiction. □

The algorithm to compute second best bundle has two cases. First is when $x^*_k < 1$, then from Lemma 15 it is clear that if an amount of good in $\{k + 1, \ldots, d\}$ is increased then the money has to come from good k. This leaves exactly $d - 1$ bundles to be checked, namely when money is transferred from good k to one of $\{k+1, \ldots, d\}$, and when it is transferred from one of $\{1, \ldots, k-1\}$ good k.

The second case is when $x^*_k = 1$, then we only need to check k bundles namely, when money is transferred from one of $\{1, \ldots, k\}$ to good $k + 1$. Thus, the next theorem follows.

Theorem 16 *Given prices $\boldsymbol{p}$ and budget B, the second best bundle with respect to the mapping Ψ for a utility function $U \in \mathcal{H}_{lin}$ at $(\boldsymbol{p}, B)$ can be computed in $O(d)$ time.*

4.2 Other classes of utility functions

By designing appropriate mappings Ψ as above, we also obtain bounds on the sample complexity of learning other classes of utility functions from revealed preferences. In particular, we can employ the same technique for the class of SPLC functions with known segment lengths and the class of CES functions with known parameter ρ. See Table 1.1 for an overview on the results and Section C in the appendix for the technical details.

4.3 Extensions

Modeling learning tasks as learning a D-dimensional linear class is quite a general technique. We now discuss how it allows for a variety of interesting extensions to the results presented here.

Agnostic setting In this work, we mostly assume that the data was generated by an agent that has a utility function that is a member of some specific class (for example, the class of linear utilities). However, this may not always be a realistic assumption. For example, an agent may sometimes behave irrationally and deviate from his actual preferences. In learning theory, such situations are modeled in the *agnostic learning* model. Here, we do not make any assumption about membership of the agents' utility function in some fixed class. The goal then is, to output a function from some class, say the class of linear utility functions, that predicts the agents' behavior with error that is at most ϵ worse than the best linear function would.

Formally, the requirement on the output classifier h in Definition 5 then becomes $\mathrm{err}^l_P(h) \leq \eta + \epsilon$ (instead of $\mathrm{err}^l_P(h) \leq \epsilon$), where η is the error of the best classifier in the class. Since our sample complexity bounds are based on a compression scheme, and compression schemes also imply learnability in the agnostic learning model (see Section A.2 in the appendix), we get that the classes of utility functions with D-dimensional linear classes as demand functions that we have analyzed are also learnable in the agnostic model. That is, we can replace the assumption that the data was generated exactly according to a linear (or SPLC or CES) function with an assumption that the agent behaves according to such a function at least a $1 - \eta$ fraction of the time.

Non-linear prices and indivisible goods So far, we looked at a market where pricing is always linear and goods are divisible (see Section 2). We note that the sample complexity results for $\mathcal{H}_{lin}, \mathcal{H}_{splc}$, and $\mathcal{H}_{ces}$ that we presented here actually apply in a broader context. Prices per unit could vary with the amount of a good in a bundle (e.g. [16]). For example, there may be discounts for larger volumes. Also, goods may not be arbitrarily divisible (e.g. [11]). Instead of one unit amount of each good in the market, there may then be a number of non-divisible items of each good on offer. Note that we can still define the functions Ψ to obtain a D-dimensional linear demand function class and the classes of utility functions discussed above are learnable with the same sample complexity (though not necessarily efficiently).

Learning preference orderings Consider the situation where we would like to not only learn the preferred choice (over a number d of options) of an agent, but the complete ordering of preferences given some prices over the options.

We can model this seemingly more complex task as a learning problem as follows: Let $\mathcal{X} = \mathbb{R}^d_+$ be our instance space of price vectors. Denote by $\mathcal{Y} = S_d$ the group of permutations on d elements. Let a vector $\boldsymbol{w} \in \mathbb{R}^d$ represent the unknown valuation of the agent, that is w_i indicates how much the agent values option i. Consider the functions $h_{\boldsymbol{w}} : \mathbb{R}^d_+ \to S_d$ such that $h_{\boldsymbol{w}}(\boldsymbol{p})$ is the permutation corresponding to the ordering over the values w_i/p_i (i.e. $\pi(1)$ is the index with the largest value per money w_i/p_i and so on).

Finally, consider the hypothesis class $\mathcal{H}_\pi = \{h_{\boldsymbol{w}} \ : \ \boldsymbol{w} \in \mathbb{R}^d_+\}$. We show below hat $\mathcal{H}_\pi$ is a d-dimensional linear class. Therefore, this class can also be learned with sample complexity $O\left(\frac{d\log(1/\epsilon)+\log(1/\delta)}{\epsilon}\right)$. With the same construction as for linear demand functions (see Lemma 28 in the appendix), we can also show that the Natarajan dimension of $\mathcal{H}_\pi$ is lower bounded by $d-1$, which implies that this bound on the sample complexity is essentially optimal.

To see that $\mathcal{H}_\pi$ is d-dimensional linear, consider the map $\Psi : \mathcal{X} \times S_d \to \mathbb{R}^d$ defined by $\Psi(\boldsymbol{p}, \pi) = \sum_{1 \le i < j \le d} \pi_{ij} \cdot ((1/p_j)e_j - (1/p_i)e_i)$, where, π_{ij} is 1 if $\pi(i) < \pi(j)$ and else -1; $e_1, \ldots, e_d$ is the standard basis of $\mathbb{R}^d$

5 Learning via Revealed Preference Queries

In this section we design algorithms to learn classes $\mathcal{H}_{lin}$, $\mathcal{H}_{splc}$, $\mathcal{H}_{leon}$ or $\mathcal{H}_{ces}$ using poly(n, d) revealed preference queries. (Recall that we have assumed all defining parameters of a function to be rationals of size (bit length) at most n.)

5.1 Characterization of optimal bundles

In this section we characterize optimal bundles for linear, SPLC, CES and Leontief utility functions. In other words, given $(\boldsymbol{p}, B)$ we characterize $\mathcal{B}_U(\boldsymbol{p}, B)$ when U is in $\mathcal{H}_{lin}$, $\mathcal{H}_{splc}$, $\mathcal{H}_{ces}$, or $\mathcal{H}_{leon}$. Since function U is concave, formulation (1) is a convex formulation, and therefore Karush-Kuhn-Tucker (KKT) conditions characterize its optimal solution [4, 2]. For a general formulation $\min\{f(\boldsymbol{x}) \mid g_i(\boldsymbol{x}) \le 0, \ \forall i \le n\}$, the KKT conditions are as follows, where μ_i is the dual variable for constraint $g_i(\boldsymbol{x}) \le 0$.

$$\begin{array}{c} L(\boldsymbol{x}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \sum_{i \le n} \mu_i g_i(\boldsymbol{x}); \quad \forall i \le n: \ \frac{dL}{dx_i} = 0 \\ \forall i \le n: \ \mu_i g_i(\boldsymbol{x}) = 0, \quad g_i(\boldsymbol{x}) \le 0, \quad \mu_i \ge 0 \end{array}$$

In (1), let μ, μ_j and μ'_j be dual variables for constraints $\langle \boldsymbol{p}, \boldsymbol{x} \rangle \le B$, $x_j \le 1$ and $-x_j \le 0$ respectively. Then its optimal solution $\boldsymbol{x}^* = \mathcal{B}_U(\boldsymbol{p}, B)$ satisfies the KKT conditions: ${}^{dL}/_{dx_j}|_{\boldsymbol{x}^*} = -{}^{dU}/_{dx_j}|_{\boldsymbol{x}^*} + \mu p_j + \mu_j - \mu'_j = 0$, $\mu'_j x^*_j = 0$, and $\mu_j(x^*_j - 1) = 0$. Simplifying these gives us:

$$\begin{array}{lll} \forall j \ne k, \quad x^*_j > 0, \ x^*_k = 0 & \Rightarrow & \frac{dU/dx_j|_{x^*}}{p_j} \ge \frac{dU/dx_k|_{x^*}}{p_k} \\ \forall j \ne k, \quad x^*_j = 1, \ 0 \le x^*_k < 1 & \Rightarrow & \frac{dU/dx_j|_{x^*}}{p_j} \ge \frac{dU/dx_k|_{x^*}}{p_k} \\ \forall j \ne k, \quad 0 < x^*_j, x^*_k < 1 & \Rightarrow & \frac{dU/dx_j|_{x^*}}{p_j} = \frac{dU/dx_k|_{x^*}}{p_k} \end{array} \tag{2}$$

Linear functions: Given prices $\boldsymbol{p}$, an agent derives ${}^{a_j}/_{p_j}$ utility per unit money spent on good j (bang-per-buck). Thus, she prefers the goods where this ratio is maximum. Characterization of optimal bundle exactly reflects this,

$$\begin{array}{lll} \forall j \ne k, \quad x^*_j > 0, \ x^*_k = 0 & \Rightarrow & \frac{a_j}{p_j} \ge \frac{a_k}{p_k} \\ \forall j \ne k, \quad x^*_j = 1, \ 0 \le x^*_k < 1 & \Rightarrow & \frac{a_j}{p_j} \ge \frac{a_k}{p_k} \\ \forall j \ne k, \quad 0 < x^*_j, x^*_k < 1 & \Rightarrow & \frac{a_j}{p_j} = \frac{a_k}{p_k} \end{array} \tag{3}$$

SPLC functions: At prices $\boldsymbol{p}$, the utility per unit money (bang-per-buck) on segment (j, k) is ${}^{a_{jk}}/_{p_j}$. Clearly, the agent prefers segments with higher bang-per-buck and therefore, if allowed, will buy segments in order of decreasing bang-per-buck. Let x^*_j in optimal bundle be be ending at t^{th} segment. Then clearly segments 1 to $t-1$ are completely allocated, and segments $t+1$ to $|U_j|$ are not allocated at all. Accordingly define $\forall k < t, \ x^*_{jk} = l_{jk}$, $x^*_{jt} = x^*_j - \sum_{k<t} l_{jk}$, and $\forall k > t, \ x^*_{jk} = 0$, then similar to the conditions for linear function, these satisfy,

$$\begin{array}{lll} \forall (j,k) \ne (j',k'), \quad x^*_{jk} > 0, \ x^*_{j'k'} = 0 & \Rightarrow & \frac{a_{jk}}{p_j} \ge \frac{a_{j'k'}}{p_{j'}} \\ \forall (j,k) \ne (j'k'), \quad x^*_{jk} = l_{jk}, \ 0 \le x^*_{j'k'} < l_{jk} & \Rightarrow & \frac{a_{jk}}{p_j} \ge \frac{a_{j'k'}}{p_{j'}} \\ \forall (j,k) \ne (j'k'), \quad 0 \le x^*_{jk}, x^*_{j'k'} < l_{jk} & \Rightarrow & \frac{a_{jk}}{p_j} = \frac{a_{j'k'}}{p_{j'}} \end{array} \tag{4}$$

CES utility functions: Since $\frac{dU}{dx_j} = \frac{a_j U(x)^{1-\rho}}{x_j^{1-\rho}}$ and $-\infty < \rho < 1$, we have $\lim_{x_j \to 0} \frac{dU}{dx_j} = \infty$. Therefore, conditions (2) gives the following. $\forall j,\ x_j^* > 0$,

$$
\begin{array}{llll}
\forall j \neq k, & x_k^* < x_j^* = 1 & \Rightarrow \ \frac{a_j}{p_j} \geq \frac{a_k}{p_k}\left(\frac{1}{x_k^*}\right)^{1-\rho} & \Rightarrow \ \frac{a_j}{p_j} > \frac{a_k}{p_k} \\
\forall j \neq k, & 0 < x_k^* \leq x_j^* < 1 & \Rightarrow \ \frac{a_j}{a_k} = \frac{p_j}{p_k}\left(\frac{x_j^*}{x_k^*}\right)^{1-\rho} & \Rightarrow \ \frac{a_j}{p_j} \geq \frac{a_k}{p_k}
\end{array} \tag{5}
$$

Leontief utility functions: An optimal bundle at Leontief is essentially driven by a_js and not so much by prices. Note that to achieve unit amount of utility the buyer has to buy at least a_j amount of each good j, and therefore has to spend $\sum_j a_j p_j$ money. Thus from money B she can obtain at most $\frac{B}{\sum_j a_j p_j}$ units of utility. Further, since she will always buy the cheapest optimal bundle, we get,

$$\forall j, x_j^* = \beta a_j, \quad \text{where} \ \ \beta = \min \left\{ \frac{B}{\sum_j a_j p_j}, \frac{1}{\max_j a_j} \right\} \tag{6}$$

The next theorem follows using the KKT conditions of (2) for each class of utility functions.

Theorem 17 *Given prices $\boldsymbol{p}$ and budget B, conditions (3), (4), (5) and (6) together with feasibility constraints of (1) exactly characterizes $\boldsymbol{x}^* = \mathcal{B}_U(\boldsymbol{p}, B)$ for $U \in \mathcal{H}_{lin}$, $U \in \mathcal{H}_{splc}$, $U \in \mathcal{H}_{ces}$ and $U \in \mathcal{H}_{leon}$ respectively.*

5.2 Linear functions

Recall that, if $U \in \mathcal{H}_{lin}$ then $U(\boldsymbol{x}) = \sum_j a_j x_j$, where $\sum a_j = 1$. First we need to figure out which a_js are non-zero.

Lemma 18 *For $p_j = 1,\ \forall j$ and $B = n$, if $\boldsymbol{x} = \mathcal{B}_U(\boldsymbol{p}, B)$, then $x_j = 0 \Rightarrow a_j = 0$.*

Proof : Since $B = \sum_j p_j$, the agent has enough money to buy all the good completely, and the lemma follows as the agent buys cheapest optimal bundle. □

Lemma 18 implies that one query is enough to find the set $\{j \mid a_j > 0\}$. Therefore, wlog we now assume that $\forall j \in \mathcal{G},\ a_j > 0$.

Note that, it suffices to learn the ratios $\frac{a_j}{a_1},\ \forall j \neq 1$ exactly in order to learn U, as $\sum_j a_j = 1$. Since the bit length of the numerator and the denominator of each a_j is at most n, we have that $1/2^{2n} \leq a_j/a_1 \leq 2^{2n}$. Using this fact, next we show how to calculate each of these ratios using $O(n)$ revealed preference queries, and in turn the entire function using $O(dn)$ queries.

Recall the optimality conditions (3) for linear functions. Algorithm 2 determines a_j/a_1 when called with $H = 2^{2n}$, $q = 1$ and $x_j^e = 0$.[3] The basic idea is to always set budget B so low that the agent can by only the most preferred good, and then do binary search by varying p_j appropriately. Correctness of the algorithm follows using (3) and the fact that bit length of a_j/a_1 is at most $2n$.

Theorem 19 *The class $\mathcal{H}_{lin}$ is learnable from $O(nd)$ revealed preference queries.*

[3]These three inputs are irrelevant for learning linear functions, however they will be used to learn SPLC functions in Appendix 5.3.

Algorithm 2 Learning Linear Functions: Compute a_j/a_1

Input: Good j, upper bound H, quantity q of goods, extra amount x_j^e.
Initialize: $L \leftarrow 0$; $p_1 \leftarrow 1$; $p_k \leftarrow 2^{10n}$, $\forall k \in \mathcal{G} \setminus \{j, 1\}$; $i \leftarrow 0$; flag $\leftarrow nil$
while $i <= 4n$ **do**
  $i \leftarrow i+1$; $p_j \leftarrow \frac{H+L}{2}$; $B \leftarrow x_j^e * p_j + \frac{\min\{p_1, p_j\}}{q}$; $\boldsymbol{x} \leftarrow \mathcal{B}_U(\boldsymbol{p}, B)$
  if $x_j > 0$ & $x_1 > 0$ **then** **Return** p_j;
  if $x_j > 0$ **then** $L \leftarrow p_j$; flag $\leftarrow 1$; **else** $H \leftarrow p_j$; flag $\leftarrow 0$;
end while
if flag $= 1$ **then** Round up p_j to nearest rational with denominator at most 2^n
else Round down p_j to nearest rational with denominator at most 2^n
Return p_j.

5.3 Separable piecewise-linear concave (SPLC) functions

In this section we design a learning mechanism for a function of class $\mathcal{H}_{splc}$, which requires us to learn slops as well as lengths of each of the segment (j, k). As discussed in Section 2.1 that for any $\alpha > 0$, $\mathcal{B}_U(\boldsymbol{p}, B) = \mathcal{B}_{\alpha U}(\boldsymbol{p}, B)$, $\forall (\boldsymbol{p}, B)$, it is impossible to distinguish between functions U and αU, and that is why we made normalizing assumptions while defining $\mathcal{H}_{lin}$, $\mathcal{H}_{ces}$ and $\mathcal{H}_{leon}$. Similarly for $U \in \mathcal{H}_{splc}$ we wlog assume that $a_{11} = 1$ now on.

As the size of each a_{jk} and l_{jk} is at most n, we have $\frac{1}{2^n} \leq a_{jk} \leq 2^n$, $\forall (j, k)$ and $\frac{1}{2^n} \leq l_{jk} \leq 2^n$, $\forall j$, $\forall k < |U_j|$; recall that length of the last segment for each good is infinity, i.e., $l_{j|U_j|} = \infty$. Therefore, slop of first segments a_{j1} of good j, can be learned by calling Algorithm 2 with $H = 2^n$, $q = \frac{1}{2^{n+1}}$ and $B^e = 0$; extra budget B^e will be used to learn slops of second segment onward. This will make sure that no segment can be bought fully during the algorithm, and therefore when a good is bought we know that the allocation is on its first segment.

Next we show how to learn length l_{j1} of this segment. Suppose we fix prices p_1 and p_j such that agent is prefers segment $(j, 1)$ before $(1, 1)$ before $(j, 2)$, i.e., $\frac{a_{1j}}{p_j} > \frac{a_{11}}{p_1} > \frac{a_{j2}}{p_j}$, then Algorithm 3 outputs l_{j1} when provided with p_1, p_j and $x_j^e = 0$. The basic idea is to do binary search by varying the budget appropriately.

Algorithm 3 Learning Linear Functions: Compute l_{jk}

Input: Good j, prices p_j and p_1, extra amount x_j^e.
Initialize: $H \leftarrow 2^{n+1}$; $L \leftarrow 0$; $p_k \leftarrow 2^{10n}$, $\forall k \in \mathcal{G} \setminus \{j, 1\}$; $i \leftarrow 0$; flag $\leftarrow nil$;
$B \leftarrow (H + x_j^e) * p_j$; $\boldsymbol{x} \leftarrow \mathcal{B}_U(\boldsymbol{p}, B)$; **if** $x_1 = 0$ **then Return** ∞;
while $i <= 2n+1$ **do**
  $i \leftarrow i+1$; $T \leftarrow \frac{H+L}{2}$; $B \leftarrow (T + x_j^e) * p_j$; $\boldsymbol{x} \leftarrow \mathcal{B}_U(\boldsymbol{p}, B)$;
  if $x_1 = 0$ **then** $L \leftarrow T$; flag $\leftarrow 1$; **else** $H \leftarrow T$; flag $\leftarrow 0$;
end while
if flag $= 1$ **then** Round up T to nearest rational with denominator at most 2^n;
else Round down T to nearest rational with denominator at most 2^n;
Return T;

The next question is what should be p_1 and p_j so that $\frac{a_{1j}}{p_j} > \frac{a_{11}}{p_1} > \frac{a_{j2}}{p_j}$ is ensured. Setting $p_j = a_{j1}$ and $p_1 = a_{11} + \epsilon > 1$ ensures $\frac{a_{11}}{p_1} < \frac{a_{1j}}{p_j}$. Further, $\epsilon = \frac{1}{2^{2n+1}}$ ensures $\frac{a_{11}}{p_1} > \frac{a_{j2}}{p_j}$ using the next claim.

Claim 20 *If* $\epsilon = \frac{1}{2^{2n+1}}$, $p_1 = a_{11} + \epsilon$, *and* $p_j = a_{jk}$ *then* $\frac{a_{j(k+1)}}{p_j} < \frac{a_{11}}{p_1}$.

Proof : As $a_{jk} > a_{j(k+1)} \geq \frac{1}{2^n}$ with bit length of both being at most n, we have $a_{jk} - a_{j(k+1)} \geq \frac{1}{2^n}$.

$$\begin{aligned} a_{jk} - a_{j(k+1)} \geq \tfrac{1}{2^n} \;\Leftrightarrow\; 1 - \tfrac{a_{j(k+1)}}{a_{jk}} \geq \tfrac{1}{2^{2n}} \;\Leftrightarrow\; \tfrac{a_{j(k+1)}}{a_{jk}} \leq 1 - \tfrac{1}{2^{2n}} \\ \Leftrightarrow\; \tfrac{a_{j(k+1)}}{p_j} < \tfrac{1}{1+\epsilon} \;\Leftrightarrow\; \tfrac{a_{j(k+1)}}{p_j} < \tfrac{a_{11}}{p_1} \end{aligned}$$

□

Induction. Once we learn slops and lengths of up to k^{th} segment of good j, can learn $a_{j(k+1)}$ by calling Algorithm 2 with $H = a_{jk}$, $q = \frac{1}{2^{n+1}}$ and $x^e_j = \sum_{t \leq k} l_{jt}$. And then learn $l_{j(k+1)}$ by calling Algorithm 3 with $p_j = a_{j(k+1)}$, $p_1 = 1 + \epsilon$, and $x^e_j = l_{jk}$ (it works using Claim 20). We stop when Algorithm 3 returns ∞.

For each good $j \neq 1$, think of a hypothetical 0^{th} segment with $a_{j0} = 2^{n+1}$ and $l_{j0} = 0$, and apply the above inductive procedure to learn a_{jk} and l_{jk} for all $1 \leq k \leq |U_j|$. To learn the parameters for good 1, we can swap its identity with some other good, and repeat the above procedure. The number of calls to oracle $\mathcal{B}_U$ in Algorithms 2 and 3 are of $O(n)$, and if there are at most κ segments in each U_j, then total sample complexity for learning such an SPLC function is $O(nd\kappa)$.

Theorem 21 *The class $\mathcal{H}_{splc}$ is learnable from $O(nd\kappa)$ revealed preference queries.*

5.4 CES and Leontief functions

In this section we show that surprisingly constantly many queries are enough to learn a CES or a Leontief function. The reason behind this is that the optimal bundles of these functions are well behaved, e.g., a buyer buys all the goods of non-zero amount, and in a fixed proportion in case of Leontief functions.

CES. Let $U \in \mathcal{H}_{ces}$ be a function defined as $U_{\boldsymbol{a}\rho}(\boldsymbol{x}) = (\sum_j a_j x_j^\rho)^{1/\rho}$, where $\sum_j a_j = 1$ and $\rho < 1$. Since an optimal bundle for such a $U_{\boldsymbol{a}\rho}$ contains non-zero amount of good j only if $a_j > 0$, wlog we assume that $a_j > 0,\ \forall j$. We show that two queries, with prices $\boldsymbol{p} > 0$ and budget $B < \min_j p_j$ are enough to learn $\mathcal{H}_{ces}$ from revealed preference queries.

Let $p^1_j = 1,\ \forall j$, $p^2_j = j,\ \forall j$, $B = 0.5$, $\boldsymbol{x}^1 = \mathcal{B}_U(\boldsymbol{p}^1, B)$ and $\boldsymbol{x}^2 = \mathcal{B}_U(\boldsymbol{p}^2, B)$. Since there is not enough budget to buy any good completely in either query, we have $0 < x^i_j < 1,\ i = 1, 2, \forall j$, using (5). Like for linear functions, it is enough to learn ratios $\frac{a_j}{a_1},\ \forall j \neq 1$. Using Equation (5), Section 5.1, we get the following.

$$\begin{aligned} i = 1, 2, \forall j \neq 1,\ \frac{a_j}{a_1} = \frac{p^i_j}{p^i_1}\left(\frac{x^i_j}{x^i_1}\right)^{1-\rho} \quad &\Rightarrow \quad \left(\frac{x^1_j}{x^1_1}\right)^{1-\rho} = j\left(\frac{x^2_j}{x^2_1}\right)^{1-\rho} \\ &\Rightarrow \quad (1-\rho)\log\frac{x^1_j}{x^1_1} = \log j + \log\frac{x^2_j}{x^2_1} \end{aligned}$$

Since $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ are known, we can evaluate the above to get ρ and a_j/a_1.

Leontief. Consider a Leontief function $U_{\boldsymbol{a}} \in \mathcal{H}_{leon}$ such that $U_{\boldsymbol{a}}(\boldsymbol{x}) = \min_j x_j/a_j$, where $\sum_j a_j = 1$. Wlog, we assume that $a_j > 0,\ \forall j$; if $a_j = 0$ then $x_j = 0$ in an optimal bundle at any given prices and budget. We show that one query, with prices $\boldsymbol{p} > 0$ and budget $B < \min_j p_j$, is enough to determine $U_{\boldsymbol{a}}$ and thus one query suffices to learn the class $\mathcal{H}_{leon}$ from revealed preference queries.

Suppose, $\boldsymbol{x} = \mathcal{B}_U(\boldsymbol{p}, B)$ where $p_j = 1,\ \forall j$ and $B = 0.5$. Then using (6), we get $\beta = B/\sum_j a_j = B = 0.5$ and $a_j = x_j/\beta = 2x_j$.

Theorem 22 *The classes $\mathcal{H}_{ces}$ and $\mathcal{H}_{leon}$ are learnable from $O(1)$ revealed preference queries.*

Acknowledgments

This work was supported in part by AFOSR grant FA9550-09-1-0538, ONR grant N00014-09-1-0751, NSF grants CCF-0953192 and CCF-1101283, a Microsoft Faculty Fellowship, and a Google Research Award.

References

[1] S. N. Afriat. The construction of utility functions from expenditure data. *International Economic Review*, 1967.

[2] M. S. Bazarra, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons, 2006.

[3] E. Beigman and R. Vohra. Learning from revealed preference. In *EC*, pages 36–42, 2006.

[4] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2009.

[5] M. Collins. Discriminative reranking for natural language parsing. In *ICML*, pages 175–182, 2000.

[6] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[7] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[8] A. Daniely and S. Shalev-Shwartz. Optimal learners for multiclass problems. In *COLT*, pages 287–316, 2014.

[9] E. Diewert. Afriat and revealed preference theory. *Review of Economic Studies*, 40:419–426, 1973.

[10] A. R. Dobell. A comment on A. Y. C. Koo's an empirical test of revealed preference theory. *Econometrica*, 33(2):451–455, 1965.

[11] F. Echenique, D. Golovin, and A. Wierman. A revealed preference approach to computational complexity in economics. In *EC*, pages 101–110, 2011.

[12] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. In *Symposium on Computational Geometry*, pages 61–71, 1986.

[13] H. S. Houthakker. Revealed preference and the utility function. *Economica*, 17:159–174, 1950.

[14] A. Y. C. Koo. An empirical test of revealed preference theory. *Econometrica*, 31(4):646–664, 1963.

[15] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[16] S. Lahaie. Kernel methods for revealed preference analysis. In *European Conference on Artificial Intelligence*, pages 439–444, 2010.

[17] N. Littlestone and M. K. Warmuth. Relating data compression and learnability. Unpulished manuscript, 1986.

[18] A. Mas-Colell. The recoverability of consumers' preferences from market demand. *Econometrica*, 45(6):1409–1430, 1977.

[19] A. Mas-Colell. On revealed preference analysis. *The Review of Economic Studies*, 45(1):121–131, 1978.

[20] A. Mas-Colell, M. D. Whinston, and J. R. Green. *Microeconomic Theory*. New York: Oxford university press, 1995.

[21] B. Natarajan. On learning sets and functions. *Machine Learning*, 4(1):67–97, 1989.

[22] M. Richter. Revealed preference theory. *Econometrica*, 34(3):635–645, 1966.

[23] P. Samuelson. Consumption theory in terms of revealed preference. *Econometrica*, 15, 1948.

[24] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning*. Cambridge University Press, 2014.

[25] H. Uzawa. Preference and rational choice in the theory of consumption. *Mathematical Models in Social Science, eds. K. J. Arrow, S. Karlin, and P. Suppes*, 1960.

[26] H. R. Varian. The non-parametric approach to demand analysis. *Econometrica*, 50:945–974, 1982.

[27] H. R. Varian. Revealed preference. In *Samuelsonian Economics and the 21st Century. M. Szenberg, L. Ramrattand, and A. A. Gottesman, editors*, pages 99–115, 2005.

[28] M. Zadimoghaddam and A. Roth. Efficiently learning from revealed preference. In *WINE*, pages 114–127, 2012.

Appendix

A Multi-class learning background

Here we review previously established results on multiclass learnability, that are relevant to the results in our paper.

A.1 The new bound for linear classes

In our work, we employ the following recent upper bound by [8] on the sample complexity of *D-dimension linear* hypothesis classes (Definition 8).

Theorem 23 ([8], Theorem 5, part 1) *For every $\Psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$, the (PAC) sample complexity of learning $\mathcal{H}_\Psi$ is*

$$m[\mathcal{H}_\Psi](\epsilon, \delta) = O\left(\frac{D\log(1/\epsilon) + \log(1/\delta)}{\epsilon}\right).$$

The upper bound in the above Theorem is achieved by a compression scheme based algorithm. That is, the authors show that there always exists a compression scheme for linear classes, which yields learnability for both the realizable and the agnostic case as we outline next.

A.2 Compression scheme based learning

Definition 24 (Compression scheme) *Let $\mathcal{H} \subseteq \mathcal{Y}^\mathcal{X}$ be a hypothesis class. A compression scheme of size d for the class $\mathcal{H}$ consists of two functions $C : \bigcup_{n\in\mathbb{N}}(\mathcal{X} \times \mathcal{Y})^n \to (\mathcal{X} \times \mathcal{Y})^d$ and $D : (\mathcal{X} \times \mathcal{Y})^d \to \mathcal{H}$ satisfying the following condition:*

- *Let $S = ((x_1, y_1), \ldots, (x_n, y_n))$ with $y_i = h(x_i)$ for some $h \in \mathcal{H}$ and all i. Then $C(S)$ is a subsequence of S and for the function $h_D = D(C(S)) \in \mathcal{H}$ we have $h_D(x_i) = y_i$ for all x_i in S.*

If a class admits a compression scheme, then it is learnable both in the realizable and in the agnostic case with the following sample complexity bounds (also see [24], Chapter 30):

Theorem 25 (Based on [17]) *Assume that class $\mathcal{H} \subseteq \mathcal{Y}^\mathcal{X}$ has a compression scheme (C, D) of size d. Then it is learnable in the realizable case (by the algorithm $D \circ C$) with sample complexity satisfying*

$$m[\mathcal{H}](\epsilon, \delta) = O\left(\frac{d\log(1/\epsilon) + 1/\delta}{\epsilon}\right).$$

Moreover, the class is also learnable in the the agnostic case with sample complexity satisfying

$$m[\mathcal{H}](\epsilon, \delta) = O\left(\frac{d\log(d/\epsilon) + 1/\delta}{\epsilon^2}\right).$$

A.3 Lower bounds

The following measure of complexity of a hypothesis class yields a lower bound for multi-class learnability:

Definition 26 (N-shattering; Natarajan dimension) *A set $\{x_1, \ldots, x_n\}$ is N-shattered by a class of functions $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ if there exists two functions $f_1, f_2 \in \mathcal{Y}^{\mathcal{X}}$ with $f_1(x_i) \neq f_2(x_i)$ for all $i \in [n]$, such that, for any binary vector $v \in \{0,1\}^n$ of indices, there exists an $h_v \in H$ with*

$$h_v(x_i) \begin{cases} = f_1(x_i) & \text{if} \quad v_i = 1 \\ = f_2(x_i) & \text{if} \quad v_i = 0 \end{cases}$$

We call the size of a largest N-shattered set the Natarajan-dimension *of the class $\mathcal{H}$.*

Theorem 27 ([21]) *The sample complexity of learning a multi-class hypothesis class $\mathcal{H}$ satisfies*

$$m[\mathcal{H}](\epsilon, \delta) = \Omega\left(\frac{d_N(H) + \ln(1/\delta)}{\epsilon}\right)$$

B The lower bound in Theorem 12

We show a lower bound on the Natarajan dimension of $\widehat{\mathcal{H}_{lin}}$:

Lemma 28 *The Natarajan dimension of the class $\widehat{\mathcal{H}_{lin}}$ is at least $d-1$.*

Proof : We show that there is a set of pairs of price vectors and budgets of size $d-1$ that is N-shattered by $\widehat{\mathcal{H}_{lin}}$. Consider the set $\{(\boldsymbol{p}^1, 1) \ldots (\boldsymbol{p}^{d-1}, 1)\}$ with all budgets set to 1 and with the price vectors defined by:

$$p_i^j = \begin{cases} 1 & \text{if} \quad i = 1 \\ 1 & \text{if} \quad i = j \\ 10 & \text{otherwise} \end{cases}$$

We consider the following functions f_0 and and f_1 that map the pairs $(\boldsymbol{p}^j, 1)$ to bundles. We set $f_0(\boldsymbol{p}^j, 1) = (1, 0, \ldots, 0)$ for all j; that is, f_0 maps all pairs to the bundle where only the first good is bought. Now we define f_1 by setting the i-th coordinate of the bundle $f_1(\boldsymbol{p}^j, 1)$ to

$$(f_1(\boldsymbol{p}^j, 1))_i = \begin{cases} 1 & \text{if} \quad i = j \\ 0 & \text{otherwise} \end{cases}$$

That is, f_1 maps $(\boldsymbol{p}^j, 1)$ to the bundle where only the j-th good is bought.

Now, given a vector $\boldsymbol{v} \in \{0,1\}^{d-1}$, the demand function that is defined by the utility vector $\boldsymbol{w}$ with

$$w_i = \begin{cases} 1 & \text{if} \quad i = 1 \\ 2 & \text{if} \quad v_{i-1} = 1 \\ 1 & \text{if} \quad v_{i-1} = 0 \end{cases}$$

yields $\widehat{U_{\boldsymbol{w}}}(\boldsymbol{p}^i, 1) = f_{v_j}(\boldsymbol{p}^i, 1)$ for all i, j. Thus, the set $\{(\boldsymbol{p}^1, 1) \ldots (\boldsymbol{p}^{d-1}, 1)\}$ is N-shattered. □

According to Theorem 12 above, this lower bound on the Natarajan dimension yields the lower bound for learning $\widehat{\mathcal{H}_{lin}}$ stated in the Theorem. It is not difficult to see that the shattering construction in the above lemma also yields the same lower bound for learning $\mathcal{H}_{lin}$ in the revealed preference model. For this, observe that the two functions f_0 and f_1 in the construction not only yield different optimal bundles on the $(\boldsymbol{p}^j, 1)$, but these optimal bundles also have different utility values.

C Statistical learning from revealed preferences

C.1 SPLC functions

Recall that an SPLC utility function $U_{\boldsymbol{AL}}$ can be defined by two $d \times \kappa$ matrices. Entry a_{ij} of $\boldsymbol{A}$ stands for the slope of the j-th segment of U_i (the piecewise linear function for the marginal utility over good i). Entry l_{ij} of $\boldsymbol{L}$ is the length of that same segment. If the maximum number of segments and their lengths are known a priori, we can employ the same technique as for learning linear utility functions from revealed preferences. That is, let $\mathcal{H}_{splc}^{\boldsymbol{L}}$ denote the subclass of all SPLC functions where number and lengths of the segments are fixed (defined by matrix $\boldsymbol{L}$).

As for linear utility functions, we can identify admissible candidates for optimal bundles. Note that, an agent will greedily buy segments according to an order of a_{ij}/p_i and in an optimal bundle all, but at most one, segments are bought fully (see also Section 5.1). Thus, here we call a bundle $\boldsymbol{x}$ admissible for some $(\boldsymbol{p}, B)$ if $|\{j \ : \ x_j \neq \sum_{g \leq h} l_{jg}$ for some $h \in [d]\}| \leq 1$ and $\langle \boldsymbol{p}, \boldsymbol{x} \rangle = B$. As in the linear case, if $\langle \boldsymbol{p}, \boldsymbol{1}_d \rangle = \sum_{i \in \mathcal{G}} p_i \leq B$, we also call the all 1-bundle $\boldsymbol{1}_d$ admissible (and in this case, it is the only admissible bundle).

Then the corresponding class of demand functions $\widehat{\mathcal{H}_{splc}^{\boldsymbol{L}}}$ is a κd-dimensional linear class as witnessed by the mapping

$$\Psi((\boldsymbol{p}, B), \boldsymbol{x}) = \begin{cases} \boldsymbol{x}^{\kappa d} & \text{if admissible} \\ \boldsymbol{0}_{\kappa d} & \text{otherwise} \end{cases},$$

where $\boldsymbol{x}^{\kappa d}$ is the "split" of $\boldsymbol{x}$ into κd dimensions according to the matrix $\boldsymbol{L}$ as follows:

$$x_i^{\kappa d} = \begin{cases} l_{hj} & \text{if} \quad \sum_{g \leq j} l_{hg} \leq x_h \quad \text{and} \quad h = \lceil i/d \rceil \quad \text{and} \quad i = j \mod d \\ x_j - \sum_{g \leq j-1} l_{hg} & \text{if} \quad \sum_{g \leq j-1} l_{hg} \leq x_h < \sum_{g \leq j} l_{hg} \quad \text{and} \quad h = \lceil i/d \rceil \quad \text{and} \quad i = j \mod d \\ 0 & \text{if} \quad x_h < \sum_{g \leq j-1} l_{hg} \quad \text{and} \quad h = \lceil i/d \rceil \quad \text{and} \quad i = j \mod d \end{cases}$$

Therefore, we immediately get the sample complexity result:

Theorem 29 *The classes $\mathcal{H}_{splc}^{\boldsymbol{L}}$ of linear utility functions with known segments are learnable efficiently in the revealed preference model with sample complexity*

$$O\left(\frac{\kappa d \log(1/\epsilon) + \log(1/\delta)}{\epsilon}\right).$$

In order to argue for the computational efficiency in the above theorem, according to Remark 9, we need to show how to compute the second best *admissible* bundle in polynomial-time. As in the linear case, if $\boldsymbol{1}_d$ is an admissible bundle (that is, if $\langle \boldsymbol{p}, \boldsymbol{1}_d \rangle \leq B$), then any other bundle is second best (with respect to the mapping Ψ).

Otherwise, for given $(\boldsymbol{p}, B)$, we design an $O(d)$-time algorithm to compute the second best admissible bundle, i.e., $\boldsymbol{y} \in \arg\max_{\boldsymbol{x} \text{ admissible }, \boldsymbol{x} \neq \boldsymbol{x}^*} \boldsymbol{a} \cdot \boldsymbol{x}$, where $\boldsymbol{x}^*$ is the optimal bundle.

Similar to an optimal bundle for a function of $\mathcal{H}_{lin}$, an optimal bundle for $U_{\boldsymbol{a}} \in \mathcal{H}_{splc}^{\boldsymbol{L}}$ can be computed by sorting segments (j, k) in decreasing order of $\frac{a_{jk}}{p_j}$ and buying them in order (Section 5.1); $a_{jk} > a_{j(k+1)}$ ensures that segments of a good are bought from first to last. Thus, the second best admissible optimal bundle can also be computed in similar way as done in Section 4.1.1 for $\mathcal{H}_{lin}$.

Corresponding to the optimal bundle $\boldsymbol{x}^*$, let x^*_{jk} denote the allocation on segment (j, k). Let k_j be the last segment bought of good j. Then, clearly there exists exactly one good, say t, such that $x^*_{tk_t} < l_{tk_t}$. Let $\boldsymbol{y}$ be the second best admissible bundle. Like in Lemma 13 it follows that $y_l > x{*}_l$

for exactly one good l. Further, the extra allocation has to be on segment $(l,(k_l+1)$ if $l \neq t$ else (t,k_t). Next like Lemma 14 $y_i < x_i^*$ for exactly one good i, and decrease in allocation is on segment (i,k_i). Finally, similar to Lemma 15, if $l \neq t$ and $x^*_{tk_t} < l_{tk_t}$ then $y_t < x_t^*$, and if $x^*_{tk_t} = l_{tk_t}$ then l has to be the good whose was the first to be not allocated.

Thus, the algorithm to compute second best bundle will have check only $O(d)$ bundles, namely if $x^*_{tk_t} < l_{tk_t}$ then either money is transferred from one of segments (i,k_i) to one of segments (t,k_t) or from (t,k_t) to one of $(l,(k_l+1))$, $\forall l \neq t$, and otherwise from (i,k_i) to the best unallocated segment in $\boldsymbol{x}^*$.

C.2 CES known ρ

We show that the classes of demand functions $\widehat{\mathcal{H}^{\rho}_{ces}}$ are also d-dimensional linear classes, for any $\rho \in \mathbb{R}_+, \rho \leq 1$ (the case $\rho = 1$ yields linear utility functions whose demand functions were shown to be d-dimensional linear above).

Recall that a CES function is defined by a parameter $\rho \in \mathbb{R}_+, \rho \leq 1, \rho \neq 0$ and a vector $\boldsymbol{a} \in \mathbb{R}^d_+$. Note that for some price vector $\boldsymbol{p}$ and budget B, we have

$$\operatorname*{arg\,max}_{\boldsymbol{x}\in[0,1]^n, \langle \boldsymbol{p},\boldsymbol{x}\rangle \leq B} \Big(\sum_j a_j x_j^{\rho}\Big)^{1/\rho} = \operatorname*{arg\,max}_{\boldsymbol{x}\in[0,1]^n, \langle \boldsymbol{p},\boldsymbol{x}\rangle \leq B} \sum_j a_j x_j^{\rho}$$

Thus, we can employ the following mapping:

$$\Psi((\boldsymbol{p},B),\boldsymbol{x}) = \begin{cases} \boldsymbol{x}^{\rho} \text{ if } \boldsymbol{p}\cdot\boldsymbol{x} \leq B \\ \mathbf{0}_d \text{ if } \boldsymbol{p}\cdot\boldsymbol{x} > B, \end{cases} \tag{7}$$

where $\boldsymbol{x}^{\rho} = (x_1^{\rho}, \ldots, x_d^{\rho})$. This yields:

Theorem 30 *The classes $\mathcal{H}^{\rho}_{ces}$ of linear utility functions with known parameter ρ are learnable in the revealed preference model with sample complexity*

$$O\left(\frac{d\log(1/\epsilon) + \log(1/\delta)}{\epsilon}\right).$$

C.3 Leontief

Learning the class of Leontief functions from revealed preferences in a statistical setting is trivial, since observing *one* optimal bundle reveals all the relevant information (see Section 5.1).

D Statistical learning of utility functions

As a point of comparison, we also analyze the learnability of classes of utility functions in the standard statistical multi-class learning model (Definition 5). That is, here the input to the learner is a sample $S = ((\boldsymbol{x}_1, U(\boldsymbol{x}_1)), \ldots, (\boldsymbol{x}_n, U(\boldsymbol{x}_n)))$ of pairs of bundles and values generated by a distribution P over bundles and labeled by a utility function U from a class $\mathcal{H}$. The learner outputs a function from bundles to values $\mathcal{A}(S) : [0,1]^d \to \mathbb{R}_+$.

D.1 Linear

Linear functions are learnable in the multi-class learning framework. The following result has been implicit in earlier works. For completeness, we provide a proof here.

Theorem 31 *The class of linear functions $H = \{\boldsymbol{x} \mapsto \langle \boldsymbol{x}, \boldsymbol{w} \rangle \;:\; \boldsymbol{w} \in \mathbb{R}^d\}$ on $\mathbb{R}^d$ is learnable with sample complexity $O\left(\frac{d\log(d/\epsilon)+\log(1/\delta)}{\epsilon}\right)$.*

Proof : [Sketch] Note that for any two linear functions $\boldsymbol{w}$ and $\boldsymbol{w}'$, the set of points on which $\boldsymbol{w}$ and $\boldsymbol{w}'$ have the same value forms a linear subspace. Thus, the the set $H\Delta H$ of subsets of $\mathcal{X}$ where two linear functions $\boldsymbol{w}$ and $\boldsymbol{w}'$ disagree is exactly the collection of all complements of linear subspaces. The set of all linear subspaces of a vector space of dimension d has VC-dimension d. Since a collection of subsets has the same VC-dimension as the collection of corresponding complements of subsets, $H\Delta H$ has VC-dimension d for the class H of linear functions.

An *i.i.d.* sample of size $O\left(\frac{d\log(d/\epsilon)+\log(1/\delta)}{\epsilon}\right)$ is an ϵ-net for $H\Delta H$ with probability at least $1-\delta$ [12]. This guarantees that (with probability at least $1-\delta$) every function that is consistent with the sample has error at most ϵ. Note that, to find a function $\boldsymbol{w}$ that is consistent with a sample, it suffices to find a maximal linearly independent set of vectors $\boldsymbol{x}_i$ in the sample. The value on a new example can then be inferred by solving a linear system. □

D.2 SPLC and CES

It is straightforward to see that learning the class $\mathcal{H}_{splc}^{\boldsymbol{L}}$ of SPLC utility functions where the number and lengths of the segments are known reduces to learning κd-dimensional linear functions, where κ is the maximum number of segments per good. For this, given a sample S, create a new sample $S^{\kappa d}$ by mapping every example $(\boldsymbol{x}, U(\boldsymbol{x})) \in S$ to an example $(\boldsymbol{x}^{\kappa d}, U(\boldsymbol{x}))$ for $S^{\kappa d}$, where $\boldsymbol{x}^{\kappa d} \in [0,1]^{\kappa d}$ is defined coordinate-wise as follows:

$$x_i^{\kappa d} = \begin{cases} l_{hj} & \text{if} \quad \sum_{g\leq j} l_{hg} \leq x_h \quad \text{and} \quad h = \lceil i/d \rceil \quad \text{and} \quad i = j \mod d \\ x_j - \sum_{g\leq j-1} l_{hg} & \text{if} \quad \sum_{g\leq j-1} l_{hg} \leq x_h < \sum_{g\leq j} l_{hg} \quad \text{and} \quad h = \lceil i/d \rceil \quad \text{and} \quad i = j \mod d \\ 0 & \text{if} \quad x_h < \sum_{g\leq j-1} l_{hg} \quad \text{and} \quad h = \lceil i/d \rceil \quad \text{and} \quad i = j \mod d \end{cases}$$

Now, we can just learn a linear function $\boldsymbol{w} \in \mathbb{R}^d$ on $S^{\kappa d}$ and predict according to this function (employing the same mapping on a test example).

Similarly, we can reduce learning $\mathcal{H}_{ces}^{\rho}$ of learning CES functions with fixed parameter ρ to learning linear utility functions. For this, given a sample S, create a new sample S^{ρ} by mapping every example $(\boldsymbol{x}, U(\boldsymbol{x})) \in S$ to an example $(\boldsymbol{z}, (U(\boldsymbol{x}))^{\rho})$ for S^{ρ}, where $\boldsymbol{z} \in [0,1]^d$ is defined coordinate-wise by setting $z_i = (x_i)^{\rho}$.

D.3 Leontief

We now show that the class of Leontief functions is learnable. Recall that, a Leontief utility function is defined by a vector $\boldsymbol{a} = (a_1, \ldots, a_d)$ by $U_{\boldsymbol{a}}(\boldsymbol{x}) = \min_{j\in\mathcal{G}} x_j/a_j$.

Note that, given an example $(\boldsymbol{x}, y) = (\boldsymbol{x}, U_{\boldsymbol{a}}(\boldsymbol{x}))$, we have

$$U_{\boldsymbol{a}}(\boldsymbol{x}) \;\leq\; \frac{x_j}{a_j}$$

for all $j \in [d]$ with equality for at least one index j. Equivalently, we have

$$a_j \;\leq\; \frac{x_j}{U_{\boldsymbol{a}}(\boldsymbol{x})}$$

for all $j \in [d]$ with equality for at least one index j. That is, each example provides us with upper bounds on all the (unknown) parameters a_j of the utility function. This suggests the following

Algorithm 4 Learning Leontief

Input: Sample $S = ((\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^m, y^m))$
$b_j \leftarrow \min\{b_j, x_j^1/y^1\}$
for all $i \in [m]$ **do**
  for all $j \in [d]$ **do**
    $b_j \leftarrow \min\{b_j, x_j^i/y^i\}$
  end for
end for
Return: vector $\boldsymbol{b} = (b_1, \ldots, b_d)$

learning procedure: Going over all training examples, we maintain estimates b_i of the a_i, by using the above inequalities (see Algorithm 4).

On a new example, we predict with the Leontief utility function defined by $\boldsymbol{b}$.

In order to prove that the above algorithm is a successful learner, we use the following claim, that characterizes the cases where an estimate $\boldsymbol{b}$ of a target Leontief function $\boldsymbol{a}$ errs on an example $\boldsymbol{x}$.

Claim 32 *Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two vectors (defining Leontief utility functions) with $b_i \geq a_i$ for all $i \in [d]$. Then, $U_{\boldsymbol{b}}(\boldsymbol{x}) \neq U_{\boldsymbol{a}}(\boldsymbol{x})$ implies*

$$\frac{x_k}{U_{\boldsymbol{a}}(\boldsymbol{x})} < b_k$$

for the index k that defines $U_{\boldsymbol{b}}(\boldsymbol{x})$ (that is, the k that minimizes x_k/b_k).

Proof : Let $\boldsymbol{x}$ be some bundle with $U_{\boldsymbol{b}}(\boldsymbol{x}) \neq U_{\boldsymbol{a}}(\boldsymbol{x})$, that is $\min_{j \in \mathcal{G}} x_j/b_j \neq \min_{j \in \mathcal{G}} x_j/a_j$. Let k be the index that minimizes the left hand side (that defines $U_{\boldsymbol{b}}(\boldsymbol{x})$) and let i be the index that minimizes the left hand side (that defines $U_{\boldsymbol{a}}(\boldsymbol{x})$). Then the above inequality implies that either $i \neq k$ or $i = k$ and $a_i = a_k \neq b_k$.

If $i = k$ and $a_i \neq b_k$, then we get

$$\frac{x_k}{U_{\boldsymbol{a}}(\boldsymbol{x})} = \frac{x_k}{x_k/a_k} = a_k < b_k,$$

by the assumption that $b_i \geq a_i$ for all $i \in [d]$. If $i \neq k$, we have

$$\frac{x_k}{b_k} < \frac{x_i}{b_i} \leq \frac{x_i}{a_i} = U_{\boldsymbol{a}}(\boldsymbol{x}),$$

and thus

$$\frac{x_k}{U_{\boldsymbol{a}}(\boldsymbol{x})} < b_k.$$

□

Theorem 33 *The class of Leontief utility functions is learnable with sample complexity $O\left(\frac{d \log(d/\delta)}{\epsilon}\right)$.*

Proof : We show that Algorithm 4 is a successful learner for the class of Leontief utility functions. Let $\boldsymbol{a}$ be the vector that defines the target Leontief function. For each $j \in [d]$, we define consider an interval $[a_j, B_j]$, where B_j is defined by

$$B_j := \min\{B \in \mathbb{R} \ : \ \Pr_{x \sim P}[(x_j/U_{\boldsymbol{a}}(\boldsymbol{x})) \in [a_j, B]] \geq \epsilon/d\}.$$

Note that we may have $B_j = a_j$, in which case the interval contains only one point. Claim 32 implies that any Leontief utility function defined by a vector $\boldsymbol{b}$ with $b_j \in [a_j, B_j]$ for all j has error at most ϵ since for any $b_j \leq B_j$ we have

$$\Pr_{x \sim P}[(x_j / U_{\boldsymbol{a}}(\boldsymbol{x})) < b_j] \;\leq\; \epsilon/d$$

by definition of B_j. Thus, it suffices to show that the vector $\boldsymbol{b}$ that is returned by Algorithm 4 satisfies this requirement (with high probability).

Consider a sample $S = ((\boldsymbol{x}^1, y^1), \ldots, (\boldsymbol{x}^m, y^m))$, with instances generated *i.i.d.* by the distribution P over bundles and labeled by Leontief function $\boldsymbol{a}$ (that is $y^i = U_{\boldsymbol{a}}(\boldsymbol{x}^i)$). The output vector $\boldsymbol{b}$ satisfies $b_j \in [a_j, B_j]$ for all j if, for every index j, there exists an example $\boldsymbol{x}^i$ in the sample with $x^i_j / y^i = x^i_j / U_{\boldsymbol{a}}(\boldsymbol{x}) \leq B_j$, that is if the sample S hits all the intervals $[a_j, B_j]$. By definition of B_j, the probability that an *i.i.d.* sample from P of size m does not hit all the intervals is bounded by

$$n(1 - \epsilon/n)^m \;\leq\; \mathrm{e}^{\frac{\epsilon m}{d}}.$$

If $m \geq \frac{d \ln(d/\delta)}{\epsilon}$, this probability is bounded by δ. Thus, we have shown that with probability at least $1-\delta$ over the training sample S algorithm 4 outputs a Leontief function of error at most ϵ. $\square$

E Learning Utility Functions via Value Queries

In this section we show how to learn each of utility functions $\mathcal{H}_{lin}$, $\mathcal{H}_{splc}$, $\mathcal{H}_{ces}$ and $\mathcal{H}_{leon}$ efficiently from value queries. In the value query learning setting, a learning algorithm has access to an oracle that, upon given the input of a bundle $\boldsymbol{x}$, outputs the corresponding value $U(\boldsymbol{x})$ of some utility function U. Slightly abusing notation, we also denote this oracle by U.

Definition 34 (Learning from value queries) *A learning algorithm* learns a class $\mathcal{H}$ from m value queries*, if for any function $U \in \mathcal{H}$, if the learning algorithm is given responses from oracle U, then after at most m queries the algorithm outputs the function U.*

The complexity of a query learning algorithm is measured in terms of the number of queries it needs to learn a class $\mathcal{H}$. It is considered efficient if this number is polynomial in the size of the target function. Since we assume that all defining parameters in the classes of Section 2.1 are numbers of bit-length at most n, we will show that poly(n, d) queries suffice to learn these classes.

Linear function. For a function $U_{\boldsymbol{a}} \in \mathcal{H}_{lin}$, where $U_{\boldsymbol{a}}(\boldsymbol{x}) = \sum_j a_j x_j$, d queries are enough to determine it. Define $\forall k \leq d,\ x^k_j = 0,\ \forall j \neq k$ and $x^k_k = 1$. Then clearly, $a_k = U(\boldsymbol{x}^k)$.

SPLC function. Given a function $U \in \mathcal{H}_{splc}$ it can be decomposed as $U(\boldsymbol{x}) = \sum_j U_j(x_j)$, where each U_j is a piecewise-linear concave function. As described in Section 2.1, each U_j constitutes of a set of pieces with slopes and lengths. We will learn each such U_j separately. Let a_{jk} be the slope of segment k, and l_{jk} be its length. Let r be the number of segment in function U_j, then except for l_{jr} (which is ∞) let n be the maximum bit length of any a_{jk} or l_{jk}, then $1/2^n \leq a_{jk}, l_{jk} \leq 2^n$. Note that r is unknown.

Given lengths and slopes of segments $1, \ldots k-1$ determining the slope of segment k is easy: let $L = \sum_{s<k} l_{js}$ and ask for $x_j = L + \epsilon$, where $\epsilon < 1/2^n$. Then $U_j(x_j) = \sum_{s<k} a_{js} l_{js} + a_{jk}\epsilon$ (as $\epsilon < l_{jk}$) gives the value of a_{jk}. Let $u_L = \sum_{s<k} a_{js} l_{js}$.

Next is to learn the length l_{jk} of k^{th} segment. Note that, k is the last segment of function U_j if and only if $U_j(L + 2^{n+1}) = u_L + a_{jk} 2^{n+1}$. This is because if it is not the last segment then $l_{jk} \leq 2^n$. Thus, one query is enough to check this. Suppose k is not the last segment, then we will compute l_{jk} through a binary search, as follows:

S_1 Let $l_l = 0$ and $l_h = 2^{n+1}$. Set $i = 0$.

S_2 Set $l = \frac{l_l + l_h}{2}$ and $x_j = L + l$.

S_3 If $U_j(x_j) < u_l + u_{jk} l$ then set $l_h = l$, else set $l_l = l$.

S_4 Set $i = i + 1$. If $i > 2n$ then output l and exit. Else go to S_2.

In the above procedure we maintain the invariant that $l_l \le l \le l_h$. In step S_3 of an iteration, the inequality holds only if $l_{jk} < l$, and therefore the l_h is reset to l. The correctness of the procedure follows from the fact that bit length of l_{jk} is at most n.

We learn each U_j separately starting from first to the last segment. This requires n queries to learn each l_{jk}, and one query to learn a_{jk}, thus total of $O(n|U_j|)$ queries. Function $U \in \mathcal{H}_{splc}$ can be learned by making $O(n\kappa d)$ queries to its value oracle, where $\kappa = \max_j |U_j|$.

CES function with known ρ. Let $U \in \mathcal{H}_{ces}^{\rho}$ such that $U(\boldsymbol{x}) = (\sum_j a_j x_j^{\rho})^{1/\rho}$, where ρ is given. Learning such a function is equivalent to learning a linear function. Thus for $\boldsymbol{x}^k$ as defined in case of Linear functions, we get $a_k = U(\boldsymbol{x}^k)^{1/\rho}$.

Leontief function. Let $U \in \mathcal{H}_{leon}$ such that $U(\boldsymbol{x}) = \min_j {}^{x_j}/_{a_j}$, where every bit length of every a_j is at most n. In other words, if $a_j > 0$ then $\frac{1}{2^n} \le a_j \le 2^n$. Therefore, given that $a_j, a_k > 0$, we have $\frac{1}{2^{2n}} \le \frac{a_k}{a_j} \ge 2^{2n}, \ \forall j, k$.

Since $\frac{0}{0}$ is considered as ∞, for $x_j = 0$ and $\forall k \ne j, \ x_k = 1$, $U(\boldsymbol{x}) > 0$ if and only if $a_j = 0$. Thus we can figure out all the non-zero a_js using d queries, and therefore wlog assume that $a_j > 0, \ \forall j$. Consider a bundle $\boldsymbol{x}^k$, where $x_j^k = 1, \ \forall j \ne k$, and $x_k^k < {}^1/_{2^{2n}}$.

$$\forall j \ne k, \ \frac{a_k}{a_j} \ge \frac{1}{2^{2n}} \Rightarrow \frac{1}{a_j} \ge \frac{1}{2^{2n} a_k} \Rightarrow \frac{x_j^k}{a_j} > \frac{x_k^k}{a_k}$$

The above conditions imply that $U(\boldsymbol{x}) = \min_j \frac{x_j^k}{a_j} = \frac{x_k^k}{a_k} \Rightarrow a_k = \frac{x_k^k}{U(\boldsymbol{x}^k)}$. Thus, $2d$ queries are enough to learn U.

Theorem 35 *We can learn*

- $\mathcal{H}_{lin}$ *from* $O(d)$
- $\mathcal{H}_{slpc}$ *from* $O(n\kappa d)$ *(where* $\kappa = \max_j |U_j|$*)*
- $\mathcal{H}_{ces}^{\rho}$ *from* $O(d)$
- $\mathcal{H}_{leon}$ *from* $O(d)$

value queries.