1 INFERRING EQUATIONS FROM DATA - ZENNA TAVARES

The problem of inferring equations from data is approached. We seek to learn both the model structure and parameters, and hence this problem can be viewed as a simple instane of program induction. Additionally we propose a *pattern grammar* a formalism for modifying existing expressions according to a parametric

1.1 INTRODUCTION

A strict interpretation of computationalism projects beyond the metaphorical; the mind is not merely explained by computational processes, nor just is it amenable to computer simulation, it *is* computational. Cognitive tasks such as language, planning, decision making and explanation, it is argued, are as much the execution of programs as the silicon counterpart. This very diversity is but one catalyst for a more complete theory, if the mind is a computer, then what kind of computer? This study addresses one aspect of this question: we seek a algorithmic and representational account of theories and theory construction.

Humans demonstrate a remarkable capacity for constructing theories of structural complexity. Physical theories such as the law of gravitation involve iteration over variable quantities of objects. Psychological theories of mind and game playing may involve recursively considering the intentions of agents. Generative theories of vision and physics exploit simulation. When people compose these theories, which are developed intuitively and deliberatively, it resembles the constructs and languages used to compose computer programs.

This has led to the conjecture that the resemblance is more than cosmetic. As computationalism posits *cognition as computation*, any sufficiently rich representation for theories must be computational. In other words, *theory as program*.

When investigating the nature of the relationship between theories and programs, we must first approach the nature of each component. What is a theory? Functionally, theories impose constraints on possible worlds; they declare what is possible, or probable conditioned on evidence. Variation expands across numerous dimensions: the extent to which latent variables are appealed to, whether or not relationships evoke a causal nature, the rigour through which they have been evaluated, and the formality with which they are stated or communicated. Language has partitioned these variations correspondingly, with terms such as hypothesis, explanation, and definition creasing rough conceptual boundaries. Still, there is little consensus on the extent to which these are the same, or what precisely differentiates them.

The argument I propose here is that *there is a correspondence between kinds of theory and kinds of programs*, and that this is not coincidental. The models we propose in the next section incorporate both declarative components of a theory as declarative programs, and the generative component of a theory as a stochastic generative program.

First we outline the problem of inferring symbolic equations from data. Our proposal in terms of declarative and generative models is explored. The need to detect invariant features of models follows, and we outline a number of approaches to learning invariant attributes of our data and models. Model classes as an extention to models are introduced, and following this we give a description of a grammar for modifying existing expressions.

2 DISCOVERING DECLARATIVE STRUCTURE IN DATA

Theories are often formally expressed as equalities and inequalities: logical assertions on the equivalence of two expressions. An expression is a finite combination of symbols belonging to some formal language. The representational power of this language, and probable cause of its permanance and pervasiveness throughout modern science, derives primarily from two sources: its use of syntax and symbolic form, and its use computable functions.

We synthesise the preceding ideas in application to the problem of equation discovery. Given a dataset of variables $\mathcal{D} = (x_1, x_2, ..., x_n)$, each composed of N corresponding real-valued datapoints $x_i \in \mathbb{R}^N$, we aim to discover a system of equalities and inequalities which govern the data. We represent equalities as *s-expressions*, a notation for nested lists, popularised by dialects of the programming language Lisp. No hard constraints on the class of models we are willing to consider are imposed; search is not constrained only to say polynomials, linear equations or even functional relationships.

For an equation to govern the data in this sense is subtle. We must relax the hardness of logical constraints in order to deal with real world data corrupted with noise. We must also introduce some inductive bias to prefer some equalities over the infinite number of alternatives. Additionally, we must provide some means to avoid tautologies and identities; equalities which are evidently true and may even be supported by the data, but are unconstraining and hence uninteresting.

Our objectives can be stated formally in a probabilstic framework as an inductive inference problem, we seek to maximise the posterior probability $P(e|D)$ where $e \in \mathcal{E}$ is an expression in a subset of lisp, reduced to arithmetic operations and first order functions.

2.1 MODELS

The framework proceeds genrally by extending *models*. A model has explicitly both a declarative and generative component. Declaratively, a model is an equation composed of variables, parameters and functions. The generative component is a stochastic function, composed of determinsitic and stochastic primitives. It takes an integer i as input, samples values for any parameters it has, then fixing these values, samples i values of its variables.

The following code shows our representation of a power model $y = x^n$.

```
(def power-model
  {:as-expr '(= y (Math/pow x n))
  :dists {'n #(rand) 'x #(* 10 (rand))}
  :params ['n]
  :vars ['y 'x]
  :name 'power})
```

The generative model samples parameter n from a uniform distribution on the interval $[0, 1]$, variable x from a uniform distribution $[0, 10]$ and computes the value of y according to the declarative expression.

2.2 MODEL INVARIANTS

Human hypotheses construction is directed by data. We are able to infer constraints on valid proposals by first noticing regularities and structure in data. Consider data sampled from the functional relationship $y = x^2 + sin(\frac{1}{x})$, we are able to recognise the presence of a sinusoid and power relationship despite the data being grossly transformed from any canonical or prototypical model we are likely familiar with, such as $y = sin x$, or $y = x^2$.

The features or attributes of the data are more abstract than the data. We evaluate whether the data is smooth, passes through the origin, is monotonic or appears functional. We determine the number of stationary points, whether these are local maxima or minima, whether the data is periodic, and if so, the constancy of the period. Evaluating attributes may be automatic or deliberative, procedural or parallel, and the responsibility of which appears distributed from lower level sensory areas, across to higher level cognition. For our intents it suffices to consider these evaluations as procedures; programs to be applied to the data.

Our first step is to construct a first good guess. This is achived by evaluating a set of attributes which are predicates designed to capture abstract features of the data. Much in line with the theme of this work, an attribute is a program, again composed explicitly of a declarative and procedural component.

Attributes are evaluated against a data set to produce an attribute vector $A = (a_1, ..., a_n)$ where $a_i \in prop$ is a proposition. Then as a first approximation towards computing $P(e|D)$, we seek to compute $P(model|A)$, the probability of a model given an attribute vector.

Our approach to this problem is generative. We generate transformations to our model, sample parameter values from the distributions defined, and then generate entire new data sets. We then evaluate the attributes on this new dataset.

The simple example above demonstrates a hurdle, clearly our ability to generate a sinusoid is not invariant under all transformations. Hence we propose a propsoe a new kind of grammar, which we call a pattern grammar to overcome this problem.

2.3 ATTRIBUTE LEARNING

What set of attributes, or features to use is an critical choice. An attribute set must be chosen that is (approximately) optimal to some criterion. There are numerous criterons of optimality suggested in the literature, primaril based on information theoretic arguments.

We can phrase our feature selecton problem as a supervised learning problem. Each model defined a label, and data generated from that model is assigned that label. Our supervised learning task is given some data, find the most appropraite label, i.e. model. One goal for feature learning then is to learn a set of features which will minimise our classification error. We can evaluate this directly by incorporating into a cost function of a feature set, some kind of cross-validation check, and using the error directly as the cost of the features. These methods tend to lead to low error rates but can be prone to fitting and generalise poorly. An alternative is to use attempt to find features which maximally separate the classes in informationt theoretic terms

Additionally we must decide what space of features we are going to consider, whether we are selecting from a predefined set of features, modifying a feature weight vector, or perhaps even generating new features from some generative model.

2.4 MODEL CLASSES

Although the above definition of a model is very general, there are a variety of data and statistical models which it can not accommodate. Mixture model or sets of expressions such as polynomials are prime examples. In particular, any nonparametric model where hyperparameters control the structure of the model can not be represented. It should be noted that that while individual models are parametric, the method as a whole is nonparametric.

To account for this we outline a proposal for *model classes*. A model class C is a set of expressions, defined declaratively as a set of rules. For instance the model class of polynomials are a set of constraints which assert that an expression must be composed only of variables and constants, must use only addition, substraction, multiplication and exponents which must be non-negative integers.

Model classes are used in a similar way to models, they possess features which we hope to recognise in order to prune our search space, at an even coarser level than win models. To do feature recognition we again take a generative approach, this time generating models which generate data. The models generated must of course adhere to the rules of the model class. This is a difficult problem, and a number of approaches based on program transformations are being attempted.

A further use of a model is in terms of its *hyperparameters*. A model class has a set of associated hyperparameters H, where $h \in H : C \to T$, where T is an arbitrary range. For instance a polynomial model class may have a hyper parameter for the degree of a polynomial, or the number of non zero terms. By learning a mapping from the data to values of the hyperparameters, we can again help constrain our space. For instance, if our data presents a small number of modes, we should use this information to constrain the degree on the polynomials we consider.

2.5 PATTERN GRAMMAR

To reiterate, we choose good candidate models for our data, and modify them to account for residual errors. Hence, we would like some means of parameterising the modification of an existing string, and to this end we propose a new kind of grammar.

A *pattern grammar* combines *formal grammars* with *pattern matching*. Pattern matching is the process of checking a perceived sequence of tokens for the presence of the constituents of some pattern. The pattern '(I need a ?X) when matched with the string '(I need a vacation), would yield a match, and return the assignment ?X = vacation. Formal grammars are sets of production rules for strings, and define formal languages. Patterns also define formal languages in the recognition sense; they are functions that determines whether a given string belongs to the language or otherwise. They are distinguished from grammars in their admission of variables, parts of the matched expression to be extracted (or replaced) in the occurance of a match.

Intuitively we can think of patterns providing a language in which to express modifications to a string, and when combined with a formal grammar we have a means of defining, and subsequently generating, well formed modifications to any particular string.

Formally a pattern grammar is a (P, N, Σ, R):

- A finite set P of patterns, each consisting of a fixed non-zero number of matching variables.
- A finite set N of nonterminal symbols.
- A finite set Σ of terminal symbols that is disjoint from N
- A finite set R of parsing rules.

Given a pattern grammar and a *base* sentence, one can evaluate whether any other string is an extension of the base with respect to the grammar. The complexity of such a decision problem will of course depend of the matching complexity of the pattern. Our objectives in such a grammar are generative however, and we focus our concerns to constructing modifications to a given string.

To this end we can define a probabilstic pattern grammar, which extends rules with probability weights, and a probability distribution over the pattern to be selected. A probabilstic pattern grammar extends existing strings, and hence can be expressed functionally as $f : \mathcal{E} \to \mathcal{E}$.

An asignment of weights determines a probability distribution on string modifications, which is precisely is required to bias the model transformations when evalauting the likelihood function.

RESULTS AND CONCLUSION

The system described above was implemented and is able to find simple symbolic equations, similar to the form given in the study. Polynomials, and compound relations such as $y = x^2 + sinx$ can be found. We are yet to find a good feature space, as the algorithm currently makes a large number of seemingly bad initial choies. Furthermore, we are not yet able to generate models which adhere to the constraints of a model class robustly. Realising this will go a long way to the general use of this method.