A. More Related Work

Older work for set functions is *support distribution machines* (Muandet et al., 2012; Poczos et al., 2012), which we note can be expressed in the form of (1) as follows:

$$\sum_{i=1}^{N} \alpha_i y^i \left(\frac{1}{M(x)} \sum_{m=1}^{M(x)} \left(\frac{1}{M(x^i)} \sum_{m'=1}^{M(x^i)} k(x_m, x^i_{m'}) \right) \right),$$

where $\{(x^i, y^i)\}$ are training examples, $k(\cdot,\cdot)$ is a kernel, and parameters $\{\alpha_i\}$.

B. Proof of Proposition 1

Proposition 1 follows by the fact that monotonicity is preserved under function composition.

C. Satisfying the Edgeworth Constraint for Composed Functions

Ensuring the Edgeworth constraint for a multi-layer function, such as a set function, is challenging. To illustrate that, here we show that even for the simple case of $K = 1$ aggregation and $M(x) = 1$ element in the set, the simplest sufficient condition to ensure the Edgeworth constraint holds after composition with ρ requires monotonicity of $f(x)$ w.r.t the conditioning feature w and convexity of ρ:

Proposition 2. *Suppose $K = 1$ and $M(x) = 1$, such that the set function is a standard function, and suppose f is differentiable, such that the trust constraint can be expressed using derivatives. For a composition $\rho(\phi(x))$ to satisfy the sensitivity trust shape constraint, it is sufficient that:* (i) *ϕ satisfies the Edgeworth constraint,* (ii) $\frac{\partial \rho}{\partial \phi} \geq 0$, *and* (iii) *that* $\frac{\partial^2 \rho}{\partial \phi^2} \geq 0$.

Proof. It is straightforward to show that in this special case the set function reduces to a standard function such that the derivative is well-defined, and that in the limit the Edgeworth constraint can be written $\frac{\partial^2 \rho(\phi(x))}{\partial x[w] \partial x[d]} \geq 0$. By Faà di Bruno's formula: $\frac{\partial^2 \rho(\phi(x))}{\partial x[w] \partial x[d]} \geq 0$ requires $\frac{\partial \rho}{\partial \phi} \frac{\partial^2 \phi}{\partial x[w] \partial x[d]}$ $+ \frac{\partial^2 \rho}{\partial \phi^2} \frac{\partial \phi}{\partial x[w]} \frac{\partial \phi}{\partial x[d]} \geq 0$, which is satisfied by the assumptions. □

D. Edgeworth Constraints without Monotonicity Constraints

We give an example of a function f where an Edgeworth shape constraint holds but f is not monotonically increasing in either d or w. Suppose a firm x has $M(x)$ employees, and chooses whether to buy each employee a new software program $d_m \in \{0, 1\}$ and whether to pay for up to 16 hours of recommended training on that new software for each employee, $w_m \in [0, 16]$, and f is the net value to the firm. Without any training, buying the mth employee software is a loss of money. And paying for training is a loss of money if the employee doesn't have the software. Thus the net value to the firm f is not always increasing in d_m or w_m. But the more training w_m the employee gets, the the bigger value there is in having $d_m = 1$ vs $d_m = 0$. Thus the function f satisfies the Edgeworth shape constraint.

E. Satisfying Trapezoid Constraint with a Composed Function

Proof of Lemma 1. Since ρ is monotonic w.r.t each of its inputs and each ϕ_k is monotonic w.r.t input d, the resulting f is monotonic w.r.t input d. Since both ρ and each ϕ_k are continuous so is the resulting f. Finally let x, $x^{\min-}, x^{\min+}, x^{\max-}, x^{\max+}$ be as defined in the lemma. Then for each $k \in \{1, \ldots, K\}$, since ϕ_k satisfies the trapezoid constraint, we have:

$$\frac{1}{M(x)} \sum_{m'=1}^{M(x)} \phi_k(x^{\min-}_{m'}) \geq \frac{1}{M(x)} \sum_{m'=1}^{M(x)} \phi_k(x^{\min+}_{m'}), \quad (12)$$

and

$$\frac{1}{M(x)} \sum_{m'=1}^{M(x)} \phi_k(x^{\max-}_{m'}) \leq \frac{1}{M(x)} \sum_{m'=1}^{M(x)} \phi_k(x^{\max+}_{m'}).$$

Since ρ is monotonic w.r.t each of its inputs, it follows that

$$\rho\left(\frac{1}{M(x)} \sum_{m'=1}^{M(x)} \phi(x^{\min-}_{m'})\right) \geq \rho\left(\frac{1}{M(x)} \sum_{m'=1}^{M(x)} \phi(x^{\min+}_{m'})\right),$$

and

$$\rho\left(\frac{1}{M(x)} \sum_{m'=1}^{M(x)} \phi(x^{\max-}_{m'})\right) \leq \rho\left(\frac{1}{M(x)} \sum_{m'=1}^{M(x)} \phi(x^{\max+}_{m'})\right).$$

Thus (3) holds. □

F. Edgeworth Shape Constraint for Lattice Models

Throughout this section, we index D-dimensional vectors by $\{1, 2, \ldots, D\}$, and use the notation $\mathbf{v}[i]$ to denote the entry with index i of such a vector $\mathbf{v}$. As before, we index 2^D-dimensional vectors by $\{0, 1\}^D$, and use the notation $\mathbf{w}_\mathbf{p}$ to denote the entry with index $\mathbf{p}$, of such a vector $\mathbf{w}$.

Proof of Lemma 2. We first show the "only if" direction. Assume that g satisfies the Edgeworth's Shape Constraint, and let $\mathbf{p}$ be as defined in the lemma. It's easy to verify that since the calibrators are surjections on $[0,1]$ and $c[d]$ and $c[w]$ are monotonically increasing, there must exist $x^{++}, x^{--}, x^{+-}, x^{-+} \in \mathbb{R}^D$ as specified in Definition 2 such that: $c_\alpha(x^{--}) = \mathbf{p}$, $c_\alpha(x^{+-}) = \mathbf{p} + \mathbf{e}_d$, $c_\alpha(x^{-+}) = \mathbf{p} + \mathbf{e}_w$, and $c_\alpha(x^{++}) = \mathbf{p} + \mathbf{e}_d + \mathbf{e}_w$. By our assumption on g, $g(x^{++}) - g(x^{-+}) \geq g(x^{+-}) - g(x^{--})$. Inequality (7) now follows from the fact that for every $\mathbf{q} \in \{0,1\}^D$, $\theta_\mathbf{q} = \theta^T \psi(\mathbf{q})$.

We next prove the "if" direction. We first show that the Edgeworth's Shape Constraint is preserved when one adds monotonic calibration to the function's inputs. Let $L : \mathbb{R}^D \to \mathbb{R}$ denote the lattice function given by: $L(t) = \theta^T \psi(t)$. So $g(x) = L(c[1](x[1]), \ldots, c[D](x[D]))$. Let $x \in \mathbb{R}^D$ be a vector, a^{-*}, ,a^{+*}, a^{*-}, a^{*+}, be real numbers satisfying $a^{-*} \leq a^{+*}$, and $a^{*-} \leq a^{*+}$.

For $\gamma, \delta \in \{+,-\}$, denote by $x^{\gamma\delta}$ the vector obtained from x by setting its entry with index d to $a^{\gamma *}$ and its entry with index w to $a^{*\delta}$, as per Definition 2.

Similarly, denote by $c_\alpha(x)^{\gamma\delta}$ the vector obtained from $c_\alpha(x)$ by setting its entries with index d to $c[d](a^{\gamma *})$ and its entries with index w to $c[w](a^{*\delta})$. By the assumed monotonicity of $c[d]$ and $c[w]$, it follows that $c[d](a^{-*}) \leq c[d](a^{+*})$ and $c[w](a^{*-}) \leq c[w](a^{*+})$. Thus if L satisfies the Edgeworth shape constraint, we have

$$L(c_\alpha(x)^{++}) - L(c_\alpha(x)^{-+}) \geq L(c_\alpha(x)^{+-}) - L(c_\alpha(x)^{--}),$$

Observing that $c_\alpha(x^{\gamma\delta}) = c_\alpha(x)^{\gamma\delta}$ for all $\gamma, \delta \in \{+,-\}$, the above can be re-written:

$$L(c_\alpha(x^{++})) - L(c_\alpha(x^{-+})) \geq L(c_\alpha(x^{+-})) - L(c_\alpha(x^{--})).$$

Thus if L satisfies the Edgeworth shape constraint, then so does g.

It remains to be shown that the conditions in the lemma imply that L satisfies the Edgeworth shape constraint. We do this separately for the multilinear and simplex interpolation kernels.

Proof for Multilinear Interpolation. In this case, L is given by:

$$L(t) = \sum_{\mathbf{p} \in \{0,1\}^D} \theta_\mathbf{p} \prod_i (t[i])^{p[i]} (1 - t[i])^{1-p[i]}$$

Differentiating both sides with respect to $t[d]$, and noting that the expression inside the product is $1 - t[i]$ if $p[i]$ is 0 and $t[i]$ if $p[i]$ is 1, we obtain

$$\begin{aligned} \frac{\partial L}{\partial t[d]} &= \sum_\mathbf{p} (-1)^{\mathbf{p}[d]+1} \theta_\mathbf{p} \prod_{i \neq d} (t[i])^{p[i]} (1 - t[i])^{1-p[i]} \\ &= \sum_{\mathbf{p}: p[d]=0} (\theta_{\mathbf{p}+\mathbf{e}_d} - \theta_\mathbf{p}) \prod_{i \neq d} (t[i])^{p[i]} (1 - t[i])^{1-p[i]}, \end{aligned}$$

Thus $\partial L / \partial t[d]$ is the multilinear interpolated lattice on the $D-1$ dimensional unit hypercube with vertex values given by $\{\theta_{\mathbf{p}+\mathbf{e}_d} - \theta_\mathbf{p} : \mathbf{p} \in \{0,1\}^D, \mathbf{p}[d] = 0\}$. By Lemma 1 of Gupta et al. (2016), the inequality (7) implies that this lattice or equivalently $\partial L / \partial t[d]$ is monotonically increasing in $t[w]$. The result now follows by integrating $\partial L / \partial t[d]$ with respect to $t[d]$ as integration is a monotonic operator.

Proof for Simplex Interpolation. For an interior point s of a simplex of $[0,1]^D$ (that is, the entries of s are distinct and each one lies in $(0,1)$), $\partial L / \partial t[d](s) = \theta_{[s \geq s[d]]}$ - $\theta_{[s > s[d]]}$, where $[s \geq s[d]]$ (resp. $[s > s[d]]$) denotes the vector in $\{0,1\}^D$, with ith entry 1 if $s[i] \geq s[d]$ (resp. $s[k] > s[d]$) and 0 otherwise. See (Gupta et al., 2016) and the references therein for proofs.

Now assume that (7) holds for all suitable $\mathbf{p}$, and let $s, r \in [0,1]^D$ be 2 interior points of simplices of $[0,1]^D$, such that $r = s + \delta \mathbf{e}_w$ for some positive real δ. We will show that $(\partial L / \partial t[d])(r) \geq \partial L / \partial t[d])(s)$, or equivalently that

$$\theta_{[r \geq r[d]]} - \theta_{[r > r[d]]} \geq \theta_{[s \geq s[d]]} - \theta_{[s > s[d]]}. \tag{13}$$

Since r and s differ only in their wth coordinate, if $r[w], s[w] < s[d]$ or $r[w], s[w] > s[d]$, then (13) holds with equality. Otherwise, it must be that $s[w] < s[d] < r[w]$. Set $\mathbf{p} = [s > s[d]]$. It's easy to verify that $[s \geq s[d]] = \mathbf{p} + \mathbf{e}_d$, $[r > r[d]] = \mathbf{p} + \mathbf{e}_w$, and $[r \geq r[d]] = \mathbf{p} + \mathbf{e}_d + \mathbf{e}_w$. Inequality (13) follows by substituting the LHS of these equalities into (7).

Thus
$$\frac{\partial L}{\partial t[d]}(s) \leq \frac{\partial L}{\partial t[d]}(s + \delta \mathbf{e}_w)$$

for every positive real δ and interior point s of a simplex of $[0,1]^D$. Since L is continuous, the result follows by integrating $\partial L / \partial t[d]$ w.r.t $t[d]$ as integration is a monotonic operator. □

G. Satisfying the Trapezoid Constraint with a Calibrated Lattice

Proof of Lemma 3. By Lemma 1 of Gupta et al. (2016) for multilinear interpolation and by Lemma 3 of Gupta et al. (2016) for simplex interpolation, given the assumption $c[w]$ is monotonically increasing, and that the calibrators are surjections, the inequalities $\theta_{\mathbf{p}+\mathbf{e}_d+\mathbf{e}_w} \geq \theta_{\mathbf{p}+\mathbf{e}_d}$ are equivalent to g in (6) being monotonically increasing w.r.t $x[w]$ on the sub-domain defined by $x[d] = d_{\max}$, and because the composition of a decreasing function with an increasing function is decreasing (by the chain rule), the inequalities $\theta_{\mathbf{p}+\mathbf{e}_w} \leq \theta_{\mathbf{p}}$ are equivalent to g in (6) being monotonically decreasing w.r.t. $x[w]$ on the sub-domain defined by $x[d] = d_{\min}$, that is,

$$\left.\frac{\partial g(x)}{\partial x[w]}\right|_{x[d]=d_{\max}} \geq 0 \qquad \text{and} \qquad \left.\frac{\partial g(x)}{\partial x[w]}\right|_{x[d]=d_{\min}} \leq 0. \tag{14}$$

The first derivative in (14) is equivalent to (5), and the second derivative in (14) is equivalent to (4). Note (14) means calibrated lattice functions that satisfy the trapezoid constraint *twist* over every 2-d slice of the input domain over d and w, as illustrated in Fig. 2. □

H. Complete Example of SFE Runtime Evaluation

Fig. 4 illustrates how the SFE runtime logic works where the input z is a set of categories. The general idea is to find large subsets of the input to cover all the categories, and if a large subset is not found in a pre-built table of token estimates, then fall back to its smaller subsets, and do so recursively until estimates are found that cover all of the original input z if possible. We recommend not storing an estimate for a token (here, a subset of categories) if it does not appear frequently enough in the training data to be useful (how frequent is frequent enough which may depend on the application, the noise in the training labels, and the desired precision-recall trade-off). For the example in Fig. 4 we only included estimates if there were at least five relevant examples to use in computing a token's estimate.

I. Complete Example of SFE Token Estimate Computation

Here we show how to compute a table of token estimates following a predefined partial order of tokens. Suppose the SFE training set consists of the following 20 examples:

We tokenize each of the 20 examples into its ngrams (unigrams, bigram, and trigrams). We only include ngrams appearing at least 5 times.

At evaluation time, we will filter out any ngrams if their parent ngram has an estimate, which we find improves debuggability and often accuracy. To be consistent with that runtime filtering, at training time we should not double count an example for both a ngram and its sub-ngrams. We partition the set of unique ngrams into a sequence of disjoint subsets $\{\mathcal{T}_q\}, q = 1, \ldots, 3$, in which $\{\mathcal{T}_q\}$ is the set of ngrams with order q. Then we iterate through a decreasing q, compute estimates of ngrams in $\{\mathcal{T}_q\}$ and add those appearing enough times to the Token Table. While computing estimate of ngram t, we only use examples that are not already used to estimate its parent ngrams in the Token Table.

Ngram	**Label**
green tea	1
green tea	1
green tea	1
green tea	1
mint green tea	0
birthday cake	0
birthday cake	1
stale cake	0
cheese cake	1
chocolate cake	1
almond cake	1
black forest cake	1
green eggs and ham	0
d green	0
black tea	0
pu-erh tea	0
chamomile tea	0
roobis tea	0
mint tea	0
oolong tea	0

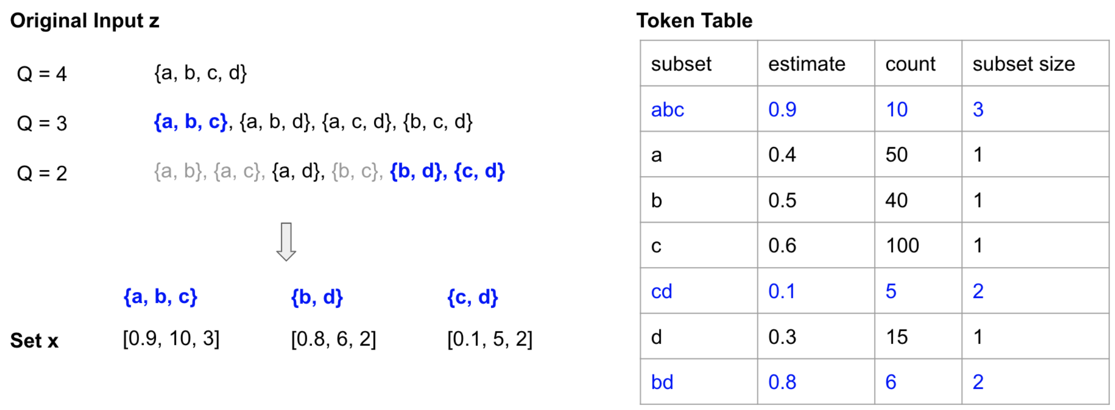
We first look at all the trigrams *mint green tea*, *black forest cake*, *green eggs and*, *eggs and ham*. None of these trigrams meets the specified count threshold of appearing 5 times, so we do not compute estimates for any of them.

Next, we find the set of bigrams that appear at least 5 times: *green tea*. The bigram *green tea* appears 5 times, including the 4 *green tea* examples and the *mint green tea* example. Using these examples we compute the maximum likelihood estimate of $E[Y|green\ tea] = 4/5$ and add it to the token table (shown below).

Finally for each unigram, we count the number of examples containing the unigram, excluding those that have already been used to estimate higher-order ngrams covering this unigram. This means that the 4 "*green tea*" examples and the "*mint green tea*" example are not used when estimating the values for the unigrams *green* and *tea* because they were already used to estimate their parent bigram *green tea*. But "*mint green tea*" is still used for unigram *mint*, which is not in the token table. This way we get the set of frequent unigrams *cake*, *tea* that have at least 5 examples to use in the estimation. We compute their estimates $E[Y|cake] = 5/7$, $E[Y|tea] = 0/6$ and add them to the token table.

We also add other information about each of the tokens that we think will be useful for the SFE set function, such as the ngram order and the number of counts the estimate was based on (ngram frequency).

The final SFE token table used at runtime is:



subset	estimate	count	subset size
abc	0.9	10	3
a	0.4	50	1
b	0.5	40	1
c	0.6	100	1
cd	0.1	5	2
d	0.3	15	1
bd	0.8	6	2

Figure 4. Example SFE handling at run-time of an input z that is a set. SFE training has already produced a stored table with per-token estimates and other features, labeled *Token Table* in the figure. The left side of the figure shows how the SFE handles an input z that is a set of 4 attributes: $\{a, b, c, d\}$. SFE starts by enumerating the highest-order subsets of z, here $Q = 4$, and then SFE checks if any of those enumerated tokens have estimates in the Token Table. For $Q = 4$, there is only the one token $\{a, b, c, d\}$ and it is not found in the Token Table, so next SFE enumerates all four $Q = 3$-order tokens. Of the four $Q = 3$ subsets, one is found in the Token Table: $\{a, b, c\}$, which is marked in blue to denote it was found. If the original input z had just been $\{a, b, c\}$, we would now be finished because the entire z has been accounted for. But since $z = \{a, b, c, d\}$, there is still the attribute d that is not covered by our token estimates. So the SFE continues on, enumerating all the $Q = 2$ subsets. Out of the 6 subsets of size 2, SFE filters out the three $Q = 2$ subsets $\{a, b\}$, $\{a, c\}$, $\{b, c\}$ that are already fully covered by the found higher-order $Q = 3$ token $\{a, b, c\}$; the filtered subsets are marked in gray. Out of the remaining three subsets of size $Q = 2$, two of them are found in the Token Table, $\{b, d\}$ and $\{c, d\}$. Now, all attributes in z have been covered by a token estimate, so SFE does not need to enumerate the $Q = 1$ subsets. The three tokens found form the set $x = \{x_1, x_2, x_3\}$, each of which is a $D = 3$ feature vector (the estimate, the count, the subset size). The set x is passed to the set function, which produces a final estimate for z.

Ngram	E[Y‖Ngram]	Order	Freq.
green tea	0.8	2	5
cake	0.71	1	5
tea	0.0	1	6

J. Experiment Details: Hyperparameter Choices

Hyperparameters were chosen by joint validation over all hyperparameter options on the validation set (see Table 1 for dataset sizes) using parallelized training on the cloud. Hyperparameter ranges were expanded when optimal models were on the edge of an initial search range, explaining the differences across models.

Deep Sets Hyperparameters: The learning rate for the ADAM optimizer from 1e-6 to 1e-3, the number of units $\{10, 20, 30, 40, 50, 80, 160, 640\}$ in each hidden layer (fixed to be the same for all 6 layers), and the number of training epochs $\{50, 100, 200, 400, 800, 1600\}$. Preliminary experiments using only 2-layers for ϕ and ρ did not improve the metrics.

Deep Lattice Network Set Function Hyperparameters: The learning rate for the Adagrad optimizer from 0.01 to 1.6, the number of keypoints $\{5, 10, 15, 20, 25, 30\}$ in the 1-d calibrators c (fixed to be the same for all calibrators), the number of aggregations K $\{1, 2, 3, 4, 5, 8, 16\}$, and the number of training epochs $\{100, 200, 500, 1000\}$. All the DLN models were run with monotonicity constraints on the primary feature.

Deep Neural Networks Hyperparameters: The learning rate for the ADAM optimizer from 1e-6 to 1e-3, the number of hidden layers $\{1, 2, 3, 4, 5\}$, and the number of training epochs $\{50, 100, 150\}$ (note the DNN's had $9\times$ as much training data as the set functions - as detailed in the experiments section).

K. Recipe Experiment: More Details

We detail the inner workings of the aggregation function on an example from the recipes dataset. The example, whose true cuisine is French but for which we are evaluating the candidate cuisine of Mexican, consists of the six ingredients {sugar, salt, fennel bulb, water, lemon olive oil, grapefruit juice}. As mentioned above, we consider subsets of up to size 3 for this problem. As the combination of ingredients is rather rare, we only find 2 subsets of size 3 (out of $\binom{6}{3}$ possibilities) that appeared in the training data at least 5 times and are therefore in the token table: {salt, water, fennelbulb} and {salt, water, sugar}. Since neither lemon olive oil nor grapefruit juice appear in any frequent subsets of size 3, the model searches for any subsets of size 2 containing one of those ingredients and doesn't find any; finally it finds grapefruit juice as a singleton in the token table, while lemon olive oil never appears in any form in the token table.

We therefore have the following 3 total tokens:

1. $\{$salt, water, fennel bulb$\}$: $P\{\text{Mexican}\} = 0.059$; count = 17; subset size = 3; number of ingredients: 6; number of tokens: 3

2. $\{$salt, water, sugar$\}$: $P\{\text{Mexican}\} = 0.043$; count = 510; subset size = 3; number of ingredients: 6; number of tokens: 3

3. $\{$grapefruit juice$\}$: $P\{\text{Mexican}\} = 0.2$; count = 10; subset size = 1; number of ingredients: 6; number of tokens: 3

Recall that there are 20 possible cuisines, so 0.05 is the break-even uninformative prior value. Here, both subsets of size 3 have fairly weak/neutral evidence while the token estimate for grapefruit juice is actually quite high.

The intermediate outputs $\phi(x)$ for these tokens are:

1. $\{$salt, water, fennel bulb$\}$: -0.005

2. $\{$salt, water, sugar$\}$: -0.156

3. $\{$grapefruit juice$\}$: 0.075

Our labels are -1/1, so positive outputs represent the model leaning towards yes and negative outputs are the opposite. Interestingly, the $\{$salt, water, sugar$\}$ subset is much more negative than $\{$grapefruit juice$\}$ is positive, even though its estimate is much closer to neutral. We can explain this by looking at the supporting information: the former has a subset size of 3, teaching the model to trust it more than a subset of size 1, and it appears 510 times vs. 10, leading to the same conclusion.

Finally, the outputs are averaged together and fed through ρ, which in the $K = 1$ case is a simple piece-wise linear transform, yielding the final output of -0.28. The classifier has correctly identified the recipe as not being Mexican.

L. Kickstarter Results Details

Here is a portion of the SFE table of some of the most common title ngrams (excluding stop words, pronouns, prepositions, and other common words). The average success rate over all projects is 40.4%.

Title Ngram	# Counts	P(success)
a short film	871	0.726
debut album	751	0.634
album	1072	0.535
book	1039	0.416
documentary	853	0.407
game	1015	0.351
music	956	0.336
clothing	616	0.149
app	773	0.103

M. Enhanced Debuggability

Debuggability of a machine-learned model arises from one's ability to form hypotheses about what could be wrong, and from one's ability to test those hypotheses. In the next sections, we discuss the debuggability of both constrained set functions and of the semantic feature engine.

The SFE approach is much easier to debug than embeddings followed by deep models because one can explicitly see what tokens the tokenization produced and which tokens had estimates retrieved. Then we know that the set function is well-behaved in terms of its monotonicity constraints so that high per-token estimates are guaranteed to only increase the final score, and are possibly also guaranteed behavior in terms of its trapezoid constraints, which helps zero-in on which tokens the model is more sensitive to. Each of these steps makes it easier to form hypotheses and test hypotheses.

One can also create partial dependence plots (Friedman, 2001) to understand the effect of each of the D features on the set function. Features that are constrained to be monotonic have clearer and more generalizable partial dependence plots.

The most common problems one sees when debugging is that estimates for tokens did not exist (because the token was too rare in the SFE training set) and instead the model had to rely on less-precise tokens (for example, if "bowling alley" does not have a stored estimate, and the model is forced to fall-back to just making a decision from just the unigram "alley"), or the per-token estimates differ from the value one expected due to imprecision of the token.

Please see the extended worked examples for SFE in Appendix H, I, and J for concrete examples that highlight the information obtainable for debugging and analysis.

N. Reduced Churn

Churn measures how different the decisions of two models are (Cormier et al., 2016). Churn is low for SFE because the parameters are more limited in what they represent, for example, an SFE per-token estimate must represent the expectation of the label for that token, and thus cannot change drastically with a different random draw of training data.

Similarly, using shape constraints for set functions reduces the ways the model can fit a given random draw of training examples, and thus are more likely to fit two random draws of training data in a more similar way.