JHU/APL CONSENSE: The Confidence-Based System Ensembler

Michael D. Lieberman*, Christine Piatko, I-Jeng Wang, Joseph Downs, Perry Wilson
The Johns Hopkins University Applied Physics Laboratory
11000 Johns Hopkins Road
Laurel, MD 20723 USA
Christine.Piatko@jhuapl.edu

Abstract

CONSENSE, JHU/APL's submission for the TAC KBP 2015 slot filling validation (SFV) and ensembling track, is presented. CONSENSE's design revolves around a constrained optimization framework for validation and ensembling, based on confidence values produced by slot filling systems. One advantage of this design is that no prior knowledge is required of how the confidence values were estimated. CONSENSE consists of three phases: feature-based filtering of SFV queries, weighted confidence scaling using constrained optimization, and confidence-based validation and ensembling. Details of these phases are provided, along with results of a preliminary evaluation of CONSENSE indicating its suitability for a variety of use cases.

1 Introduction

In this paper we present CONSENSE, JHU/APL's submission for the TAC KBP 2015 slot filling validation (SFV) and ensembling track[1]. CONSENSE's design revolves around a constrained optimization framework for validation and ensembling, based on confidence values produced by slot filling (SF) systems. A key advantage of this design is that no prior knowledge is required of how the confidence values were estimated, or indeed whether they are meaningful at all. Another useful aspect of our design is that it does not require access to the original source documents, instead relying on characteristics of fillers, confidences, and consensus among SF systems. Aspects of our design were previously explored in JHU/APL's submission to an earlier SFV evaluation (Wang et al., 2013), although CONSENSE represents several significant improvements over this earlier work.

Figure 1 shows CONSENSE's overall design. The design consists of three phases, which we briefly describe here: *feature-based filtering*, *confidence scaling*, and *validation and ensembling*. First, in the filtering phase (described in Section 2), we derive features from various characteristics of SFV queries, and use these to train a support vector machine (SVM) classifier used as a filter. We intended this filter to eliminate highly likely errors and improve overall precision, since CONSENSE's later stages tend to emphasize recall. Next, in the confidence scaling phase (Section 3), we scale the confidences of each remaining SFV query by comparing the outputs of several slot filling systems. For each output, we assign a weight to its corresponding confidence value, using one of several weighting schemes, and optimize over weighted confidence values, resulting in probabilities for each filler. For CONSENSE, we investigated three weighting schemes, namely *uniform*, *inverse precision*, and *logistic regression* (LogReg) weighting. Finally, in the validation and ensembling phase, we retain the queries whose fillers have the largest aggregated confidence, with different strategies for single-valued and list-valued slots. We describe these phases in more detail in subsequent sections, along with results of preliminary analyses of CONSENSE's performance that demonstrate its effectiveness.

To briefly clarify terminology, SFV builds on the SF task (Surdeanu, 2013) which concerns the retrieval of specific attributes, termed *slots*, of given entities from collections of natural language text. For example, an SF query q_{SF} may be, "what is the age of the entity e mentioned in document d?", where "age" is the slot. A SF system would provide an answer a to q_{SF} consisting of multiple parts: a value for the slot, termed a *filler*, such as "23"; its confidence in the filler's correctness; the filler's provenance (i.e., the documents used to determine the filler); and other characteristics. Note that some slots are *single-valued* (i.e., will have one correct filler, such as "age") while others are *list-valued* (can have multiple correct fillers, such as "cities of residence"). Building on

* Michael Lieberman moved to Google after this work was completed. E-mail: mlieb@google.com.

[1] http://www.nist.gov/tac/2015/KBP/SFValidation

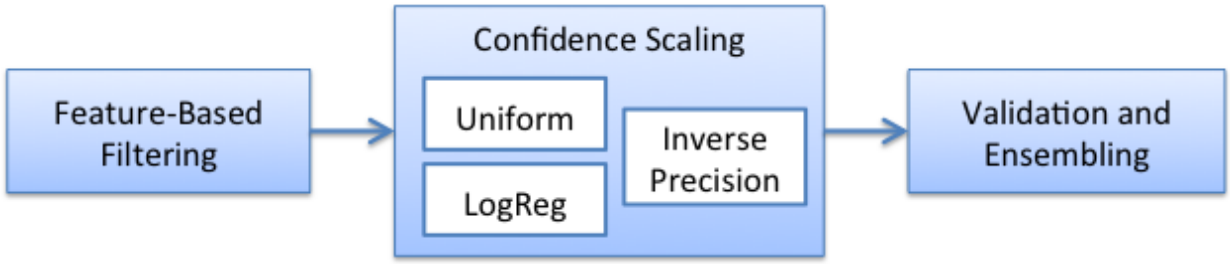

Figure 1: CONSENSE's architecture, consisting of three phases: feature-based filtering, confidence scaling, and validation and ensembling. The confidence scaling phase covers multiple weighting schemes (Section 3.3).

this, the SFV task is concerned with validating all a produced by SF systems—i.e., determining whether each a is correct or incorrect, essentially acting as a classifier of each a's correctness. Throughout the paper we will refer to an SF query and its corresponding SFV queries. As a final note, the SFV 2015 evaluation differs from previous years in that it introduces *multi-hop* queries, involving multiple levels of slots. For example, a multi-hop query q_{SF} might ask "what are the ages of the siblings of the entity e mentioned in document d?", and answering such a query would require finding e's siblings ("hop 0") and then the age of each sibling ("hop 1"). While CONSENSE is usable for multi-hop queries, it does not leverage multi-hop information.

We continue with descriptions and algorithms for CONSENSE's three processing stages, namely feature-based filtering (Section 2), confidence scaling (Section 3), and validation and ensembling (Section 4). We then present results of preliminary experiments on CONSENSE's effectiveness for SFV 2014 and SFV 2015 data in Sections 5 and 6 respectively. Finally, we provide an overview of related work in Section 7, and conclude with thoughts for future work in Section 8.

2 Feature-Based Filtering

In the first stage of CONSENSE's processing, we derive features from various characteristics of SFV queries, and use these to train an SVM classifier used as a filter. This filter eliminates highly likely errors to improve precision, since CONSENSE's later stages emphasize recall. We also use these features later in CONSENSE's LogReg weighting scheme (Section 3.3). For the SFV 2015 evaluation, we used the 2013 and 2014 SFV datasets for training, and the preliminary 2015 data for parameter tuning and system precision estimates.

2.1 Featurization

Table 1 contains the list of features used in CONSENSE's processing, organized into several groups. Filler features were based on string properties of slot fillers, such as the number of words or the filler's capitalization. Provenance features were based on the filler's provenance, such as the type of document (e.g., news article) or the length of the provenance region within the document. Data type features were based on the type of slot agreeing with the type of filler (e.g., having a city name in a city slot). These features were based on preconstructed lists of names from various sources. Finally, ensembling features were based on multiple query outputs from the same system, or agreement between multiple systems. Note that none of our features examine provenance documents in detail, instead relying solely on system outputs. Also, though they are not included in the table, for each data type feature we added an inverse feature that was also used by the classifiers. For example, is-int-in-numeric-slot would have a partner feature, is-int-in-non-numeric-slot, capturing fillers of the same form in all the non-numeric slots. Including all these variants, we used 47 features in total. Later, we assess these features' effectiveness in our experiments (Section 5.5).

2.2 Filtering

Our approach for filtering was to train a classifier to classify queries as correct (retain) or incorrect (filter). For this purpose, we trained a support vector machine (SVM) classifier with radial basis function (RBF) kernel on the subset of features marked with * from Table 1. These include the features we deemed most likely to qualify for hard filtering, mostly because they had large coefficients for logistic regression weighting. In particular, most of the negations of the data type features represent incorrect formatting, which should result in incorrect answers.

2.3 Filler Normalization

After featurizing the SFV queries, we perform an additional step of preprocessing prior to entering the confidence scaling phase (Section 3). In particular, multiple SFV queries that address the same SF query may have the same filler, but with slight differences. For example, a SF query q_{SF} may ask for the cities of residence of a given entity e. The corresponding SFV queries may in-

Table 1: Features used in CONSENSE's filtering and weighting schemes. * indicates features used to train the SVM filter when both SVM filtering and LogReg weighting were used. † indicates features based on multiple queries. Feature types include boolean (B), numeric (N), and categorical (C).

Feature	Type	Description	Example
Filler features			
is-empty-date*	B	is an empty date	XXXX-XX-XX
has-initial-capitalization	B	has first letters capitalized	Orange County Register
num-words-normalized	N	number of words in filler	Orange County Register = 3
has-multiple-names	B	is a name with multiple words	Martin Landtman
string-length-normalized	N	string length of filler	Martin Landtman = 15
has-stop-words	B	contains common English words	a heart attack
last-word-lower	B	ends in lowercase word	News executive
last-word-possessive*	B	ends in 's	Moody's
filler-is-query-name-suffix*	B	is a suffix of the query name	Martin Landtman, Landtman
Provenance features			
provenance-region-size	N	difference of provenance offsets	299–311 = 12
document-source-category	C	type of provenance document	NYT-ENG-2010. . . = NYT-ENG
Data type features			
is-int-in-numeric-slot*	B	integer in a number slot	9986000, 9,986,000
is-date-in-date-slot*	B	well-formed date in a date slot	1938-06-19, XXXX-06-19
is-city-in-city-slot*	B	city in a city slot	Cairo
is-state-in-state-slot*	B	US state in a state slot	Maine
is-country-in-country-slot*	B	country in a country slot	Pakistan
is-demonym-in-origin-slot*	B	demonym in an origin slot	Canadian
is-job-in-title-slot*	B	job title in title slot	Wood Crafter
human-name-in-human-slot*	B	human name in a person slot	Martin Landtman
is-url-in-url-slot*	B	URL in the website slot	www.cml.org.uk
has-org-suffix-in-org-slot*	B	org suffix in an org slot	Hudson Institute
Ensembling features			
rank-provenance-doc-id†	N	fraction of queries that agree on the filler's provenance document	
per-system-confidence-normed†	N	system confidence of this query, relative to all its other queries	
per-system-slot-fraction†	N	fraction of this system's queries that address this slot	
per-slot-system-rank†	N	rank based on fraction of this system's queries for this slot	
dates-inconsistent*†	B	contradicts another date filler, e.g., date-founded after date-dissolved	
per-query-string-length†	N	string length relative to other queries for the same entity and filler	

clude fillers such as "Washington DC" and "washington dc", which should all be considered equivalent when scaling confidences. Thus, we perform filler normalization to group these fillers into equivalence classes, based on case insensitivity and ignoring whitespace differences. We did experiment with other techniques such as string distance, but found them to have little effect for our approach. After filler normalization, weighter implementations assign weights and confidences to each class of fillers, rather than each individually.

3 Confidence Scaling

After filtering, we scale the confidences of each remaining SFV query by comparing the outputs of several slot filling systems. For each output, we assign a weight to its corresponding confidence value using one of several weighting schemes. We then aggregate this weighted probabilistic evidence by optimizing over weighted confidence values, resulting in probabilities for each filler. The optimization is performed for each entity/slot combination to produce a normalized confidence for each unique slot filler. We apply a constrained optimization based approach originally developed by Predd et al. (2008) for aggregation of forecasts, and first applied to slot filler validation by Wang et al. (2013). For validation of single-valued slots, we simply take the top-ranked filler, while for list-valued slots we search for a "gap" between top-ranked and lower-ranked fillers, and select those above the gap. This gap is enhanced by our use of an alternate cost function for list-valued slots. Here we provide a brief overview of the approach.

3.1 Single-Valued Slots

Given an entity/slot pair, let F denote the set of distinct slot fillers from the collection of systems under consideration. For each filler $f \in F$, we assume that $\{(c_f(j), w_f(j))\}_j$ represents the set of confidences produced by slot filling systems with associated weights (discussed later). For single-valued slots, we scale the confidence values from slot filling systems by solving the following quadratic program:

$$\min_{\mathbf{x}} \sum_{f \in F} W_f \left(x_f - \overline{c_f}\right)^2, \tag{1}$$

subject to the constraints

$$\sum_{f \in F} x_f \leq 1,\ x_f \geq 0, \tag{2}$$

where

$$W_f = \sum_j w_f(j) \tag{3}$$

denotes the total weight assigned to f, and

$$\overline{c_f} = \frac{1}{W_f} \sum_j w_f(j) c_f(j) \tag{4}$$

is the weighted average of confidences produced by the systems. The optimization produces a scaled confidence for each unique filler that will be used for validation of the slot. When the confidences generated by the systems do not violate the constraints, the quadratic program defined above reduces to weighted averaging. As the constraints (2) become more active, i.e., $\sum_{f \in F} \overline{c_f} \gg 1$, the filler f with the largest resulting x_f will approach the result of a weighted majority vote.

3.2 List-Valued Slots

In contrast to the total probability constraint (2) used for single value slots, we instead added an L_1 regularization term to our original quadratic cost function (1) for list-valued slots:

$$\min_{\mathbf{x}} ||\mathbf{x}||_1 + \lambda_s \sum_{f \in F} W_f \left(x_f - \overline{c_f}\right)^2 \tag{5}$$

The regularization term $|\mathbf{x}||_1$ results in a sparse confidence vector from which a subset of answers can be identified for validation. We also introduced a single parameter λ_s to trade-off the original quadratic cost and the L_1 term; the parameter is selected based on training data for each list-valued slot.

This sparse optimization approach enhances the "gap" in confidences—i.e., the difference between higher-ranked and lower-ranked fillers—which we leverage in validation for list-valued slots. We assume a bimodal distribution of scaled confidences. After sorting fillers by decreasing confidence, we find the position i in the list of confidences where, if we were to partition the fillers at i, would minimize an error function of the two partitions—in particular, for each partition, the sum of absolute differences of each confidence value with the partition mean. That is, we select the gap location i that minimizes the error function

$$\min_i \sum_{j=1}^{i} |c_j - \mu_H| + \sum_{k=i+1}^{n} |c_k - \mu_L| \tag{6}$$

where n is the number of fillers, and μ_H and μ_L are the means of higher and lower ranked confidence partitions, respectively. For example, for a list of scaled confidences [0.7, 0.6, 0.6, 0.3, 0.2, 0.2], the optimal gap would partition the list into two lists [0.7, 0.6, 0.6] and [0.3, 0.2, 0.2], with an error of 0.267.

3.3 Weighting Schemes

The choice of weights determines how much influence a particular SFV query will have on the final scaled confidences. Each weighting scheme assigns a weight w_f

to each unique filler (f, c_f) produced by a system (here we dropped the indexing for instances of the same filler f from different systems for the ease of presentation). The overall weight W_f and associated weighted average confidence $\overline{c_f}$ are then computed based on (3) and (4), respectively, as inputs to the optimization-based scaling presented in previous sections.

For CONSENSE, we tested three weighting schemes, in combination with per-team normalization, described below.

Uniform weighting. This scheme gives equal weight to all slot fillers from any system and is used in the absence of additional information. The approach was previously explored by Wang et al. (2013) and is used as our baseline. Under the uniform weighting scheme, we assign unit weight to each slot filler. W_f is the number of queries with the filler from any system, and $\overline{c_f}$ is simply the mean of the associated query confidences.

Inverse precision rank (InvRank) weighting. Under this scheme, we give more weight to systems with higher precision as measured from training data. In other words, the weight assigned to each instance of a slot filler is determined solely by the system and its associated relative precision for the training data. Specifically, we consider a ranking-based weighting that gives successively decreasing weight to systems in decreasing order of precision. In cases where where ground truth is not available for the training data, the same scheme can be implemented using bootstrapping. For example, surrogate ground truth may be obtained to determine the precision-based ranking using top-ranked answers with uniform weighting, as explored in previous work (Wang et al., 2013). However, for our submissions, we used the preliminary assessed 2015 data for training to set the weights.

Logistic regression (LogReg) weighting. This scheme assigns weights based on the classifier's probability of correctness for the SFV query. Under this scheme, W_f is the sum of probabilities for filler f, and $\overline{c_f}$ is the mean of the associated query confidences. The LogReg weighter was trained using the same features as the SVM filter (Section 2.1), although when both the filter and weighter were used, we split the features between the two.

Per-team normalization. Finally, in concert with our choice of weighting scheme above, we apply additional per-team weighting. Under the slot filling evaluation, teams may make multiple system submissions. We observed that systems submitted by the same team often produce the same fillers. As a result, fillers submitted by a team with multiple systems would get higher weight simply due to the number of submissions rather than by consensus with other teams' systems. To address this, we normalize the filler weights so that each team gets equal weight. For example, assume team 1 submitted three systems and team 2 submitted one, and all of team 1's systems produced filler f_1, while team 2's system produced f_2. Under uniform weighting, f_1 would have weight 3 and f_2 would have weight 1, but after per-team normalization, both f_1 and f_2 would have weight 1.

3.4 Team Mappings

As a final note, this year's SFV evaluation included additional information about each SF answer's team identity—i.e., the system that produced each SF query answer. Since many of this year's SF teams participated in previous TAC KBP evaluations, these identities could be mapped to the same teams in previous years. These mappings are useful in a variety of ways, in particular for predicting known teams' relative performances in the 2015 evaluation. We could leverage this for some of our weighting techniques, in particular inverse precision rank weighting, where we could garner more accurate weight estimates based on each known team's previous performance. However, JHU/APL did not have access to these mappings prior to the evaluation, and thus we used a fully boostrapped approach. Of course, such information is only useful for teams that participated previously, and there were many new teams in 2015, so it is unclear how much using known team mappings would affect the performance of our approach.

4 Validation and Ensembling

In this section, we present our algorithms for validation and ensembling, as well as for training the λ parameters used in list-valued confidence scaling.

4.1 Validation

Recall that our general strategy for validation is to minimize one of our two objective functions, Equations (1) and (5), to first perform confidence scaling, and then to retain those fillers with high scaled confidence. For single-valued slots, we select the single highest confidence filler, while for list-valued slots we search for a gap using the strategy described in Section 3.2 and retain fillers above the gap.

First, we present VALIDATE, our algorithm for validation, as Algorithm 1. As input, it requires the sets of SF and SFV queries Q_{SF} and Q_{SFV}, the set of training SFV queries T, the weighter implementation W, and the λ values for each slot. Later, we introduce the algorithm for training λ (Algorithm 2). VALIDATE produces R, the subset of Q_{SFV} to be retained, while the remainder are filtered. After initialization (line 2), we iterate over each SF query q_{SF} (3) and gather the SFV queries addressing q_{SF}, as well as their distinct fillers (4–6). Then, for each distinct filler f, we populate weights $\mathbf{w}$ and confidences $\overline{\mathbf{c}}$

Algorithm 1 Validate SFV queries.

1: **procedure** VALIDATE($Q_{SF}, Q_{SFV}, T, \mathrm{W}, \lambda$)
input: Q_{SF}, Slot-filling queries
input: Q_{SFV}, Slot-filling validation queries
input: T, Training SFV queries
input: W, Weighter implementation
input: λ, Per-slot lambda terms
output: R, Subset of Q_{SFV} to be retained
2: $R \leftarrow \{\}$
3: **for** $q_{SF} \in Q_{SF}$ **do**
4: $subs \leftarrow$ subset of Q_{SFV} responding to q_{SF}
5: $slot \leftarrow \textsc{Slot}(q_{SF})$
6: $F_{subs} \leftarrow \{\textsc{Filler}(s) : s \in subs\}$
7: **for** $f \in F_{subs}$ **do**
8: $subs_f \leftarrow \textsc{FillerSet}(subs, \{f\})$
9: $w_f, \overline{c_f} \leftarrow \mathrm{W}(subs_f)$
10: **if** $slot$ is single-valued **then**
11: $\mathbf{x} \leftarrow \textsc{OptSingle}(\mathbf{w}, \overline{\mathbf{c}})$
12: $f_1 \leftarrow f \in F_{subs}$ with largest x_f
13: $R \leftarrow R \cup \textsc{FillerSet}(subs, \{f_1\})$
14: **else**
15: $prec \leftarrow \textsc{Precision}(slot, T)$
16: $nf \leftarrow |F_{subs}| \cdot prec$
17: $\mathbf{x} \leftarrow \textsc{OptList}(\mathbf{w}, \overline{\mathbf{c}}, nf, \lambda_s)$
18: $gap \leftarrow \textsc{FindGap}(\mathbf{x})$
19: $F_G \leftarrow \{f \in F_{subs} : x_f \geq gap\}$
20: $R \leftarrow R \cup \textsc{FillerSet}(subs, F_G)$
21: **return** R

function FILLERSET(Q, F)
return $\{q \in Q : \textsc{Filler}(q) \in F\}$

for use in confidence scaling, using the Weighter implementation W (7–9). Note that the FILLERSET function returns the set of queries q such that q's filler is one of the given fillers F.

We then proceed with confidence scaling, which will minimize our scaling objective function to produce scaled confidences $\mathbf{x}$. For single-valued slots, we execute OPTSINGLE, which returns the $\mathbf{x}$ that minimizes Equation (1), our single-valued objective function (line 11). We take the top-ranked filler f_1 as measured by the largest corresponding x_f, and add to R all the SFV queries with f_1 as their fillers (12–13). For list-valued slots, we first compute nf, the expected number of correct fillers for q_{SF}, as the product of $|F_{subs}|$, the number of distinct fillers for q_{SF}, and the precision of $slot$ as measured from training data T (15–16). We then execute OPTLIST to minimize Equation (5), our list-valued objective function (17). Recall that OPTLIST requires nf to set optimization constraints, as well as the λ term for $slot$ for use in the objective function. After retrieving $\mathbf{x}$, we search for a gap in the scaled confidences to establish a minimum confidence threshold. We then select the set of fillers F_G whose corresponding x_f is above the gap, and add to R all the SFV queries whose filler is in F_G (18–20). At the end of VALIDATE, R contains our retained SFV queries, and we discard the remainder.

Note that VALIDATE always retains at least one SFV query for each SF query—those with the top-ranked filler for single-valued slots, and one or more above the gap for list-valued slots. As a result, in general, CONSENSE as a whole tended to emphasize recall over precision, as our findings in Sections 5 and 6 confirm.

4.2 Ensembling

To generate ensembled output, we select a subset of the SFV queries validated by VALIDATE. Our ensembled output consists of a subset of the validated output. Recall that VALIDATE produces for each SF query q_{SF} a set of SFV queries Q_{SFV} deemed to be valid. For ensembling, we group the Q_{SFV} into equivalence classes based on their fillers, and select only one q_{SFV} from each class for our ensembled output—in particular, the q_{SFV} with largest system-determined confidence. In other words, rather than retaining all Q_{SFV}, we select one q_{SFV} for each unique filler in Q_{SFV}. This avoids redundant fillers. For example, a SF query q_{SF} may ask for the `per:cities_of_residence` for an entity e, and the corresponding validated Q_{SFV} may contain q_1 = "Washington DC" filler, and q_2, q_3, q_4 = "Baltimore MD" filler. In this case, we would select q_1 and the one of q_2, q_3, q_4 with highest confidence for our ensembled output.

4.3 Training λ

Recall from Section 3.2 that λ controls the sparsity of our optimization solution for list-valued slots. A smaller λ will create a larger gap between scaled confidences, and hence will reduce the number of validated SFV queries. The challenge is to select a λ_s for each slot s that provides an appropriate level of sparsity for s. A λ_s that is too small will filter out too many SFV queries, and vice versa.

Hence, to choose a good λ_s for a slot s, we must determine an appropriate number of fillers for s. To address this challenge, we first posit that different SF systems will have different accuracies for different slot types, which we also observed in our experiments (Section 5.1). We seek to estimate the precision of SF systems for fillers of s, based on SF systems' overall precision for s as measured from training data. The precision will tell us how many correct fillers we expect to see for s, and we select a λ_s that produces a similar number of validated fillers. In other words, for slots with higher precision, we will accept more fillers, and vice versa. We performed a linear search over λ values based on empirical evidence; see Section 5.4 for more details.

Algorithm 2 presents TRAINLAMBDA, our algorithm

Algorithm 2 Train lambdas for list-valued slots.

1: **procedure** TRAINLAMBDA(Q_{SF}, Q_{SFV}, T, W, L)
   input: Q_{SF}, Slot-filling queries
   input: Q_{SFV}, Slot-filling validation queries
   input: T, Training SFV queries
   input: W, Weighter implementation
   input: L, List of lambda values to check
   output: λ, Lambda values for list-valued slots
2: $S_{list} \leftarrow$ set of list-valued slots
3: **for** $s \in S_{list}$ **do**
4: $prec \leftarrow$ PRECISION(s, T)
5: $exp_s \leftarrow prec \cdot |$SLOTSET$(T, s)|$
6: $nf_s \leftarrow 0$, $\lambda_s \leftarrow 0$
7: **for** $l \in L$ **do**
8: $\lambda'_s \leftarrow l$ for all $s \in S_{list}$
9: $ret \leftarrow$ VALIDATE$(Q_{SF}, Q_{SFV}, T$, W, $\lambda')$
10: **for** $s \in S_{list}$ **do**
11: $d \leftarrow exp_s - |$SLOTSET$(ret, s)|$
12: **if** $|d| < nf_s$ **then**
13: $nf_s \leftarrow |d|$, $\lambda_s \leftarrow l$
14: **return** λ

function SLOTSET(Q, s)
return $\{q \in Q :$ SLOT$(q) = s\}$

for training the per-slot λ parameters used in Equation (5). Inputs to TRAINLAMBDA are the same as for VALIDATE (Algorithm 1), except that in place of λ is L, a list of λ-values over which to search, and it outputs λ. We begin by initializing several values for each list-valued slot s (lines 3–6): exp_s, which holds the expected number of correct fillers for s, computed as the total number of fillers for s multiplied by the precision of slot s; and nf_s and λ_s, the current closest number of fillers to exp_s and the corresponding λ value that generated the fillers. Note that SLOTSET returns the set of queries in Q whose slot matches s. Then, we begin the search over λ values in L. For each $l \in L$, we call VALIDATE using l as λ for all slots (8–9). Then, for each slot s, we compute the difference between the number of fillers produced for s in total, and the expected number of fillers for the slot exp_s (11). If the difference is smaller than our previous best for s, we save the new best difference and its corresponding lambda value l (12–13). After iterating through all lambdas value L, we have the final λ values for each list-valued slot.

5 Preliminary Experiments

We performed experiments to assess CONSENSE's overall accuracy in terms of our five submissions to the 2015 SFV evaluation. These submissions consisted of different combinations of CONSENSE's filtering and weighting schemes (Section 3.3). Our five submissions are:

1. "Uniform": No filter, uniform weighting
2. "InvRank": No filter, InvRank weighting
3. "LogReg": No filter, LogReg weighting
4. "SVM+LogReg": SVM filter, LogReg weighting
5. "SVM+Uniform": SVM filter, uniform weighting

Due to a lack of final SFV 2015 assessment data at the time of publication, we performed two sets of preliminary experiments to measure CONSENSE's performance. For our first set of experiments, described in this section, we measured CONSENSE's performance as if had it been entered into the 2014 SFV evaluation. Thus, for training in this evaluation, we used only the SFV 2013 and SFV 2014 preliminary assessments, the latter of which constituted 10% of the assessments. We used the remaining 90% of 2014 assessments as performance test data. Later, in Section 6, we present CONSENSE's performance characteristics on a small subset of assessed 2015 data provided by the TAC KBP 2015 organizers.

For our experiments, we used the standard precision, recall, and f-score measures, as well as accuracy (i.e., the Rand index (Rand, 1971)), which provides a view of performance on incorrect SFV queries. For assessing correctness, we considered both correct and redundant SFV queries (as judged by human annotators) as positive examples, and the remainder as negative.

Table 2: Statistics by slot and filler type for the combined 2013 and 2014 SFV evaluation datasets.

	Count	Precision
Query type		
All	94340	0.2625
Single	18381	0.3541
List	75959	0.2404
Person	65660	0.2628
Org	28680	0.2619
Filler type		
Person name	25346	0.2446
Org name	18480	0.2045
Location	18026	0.2293
Date	4253	0.3158
Number	2737	0.5963

5.1 Datasets

First, we collected statistics related to our evaluation dataset, i.e., the combined 2013 and 2014 SFV data. Table 2 contains the results. We counted the numbers of SFV queries in various groups of slots broken down by category, as well as the assessed precisions for each category. Query type categories included single-valued, list-valued, person-specific (i.e., slots for person queries such

Table 3: Validation performance of CONSENSE's different filtering and weighting schemes. Best results are in bold.

Method	Precision	Recall	F-score	Accuracy
Uniform	0.4789	**0.8468**	0.6118	0.7378
InvRank	**0.5902**	0.6626	0.6243	**0.8054**
LogReg	0.5310	0.7568	0.6241	0.7775
SVM+Uniform	0.5159	0.8005	**0.6274**	0.7680
SVM+LogReg	0.5416	0.7411	0.6258	0.7837

Table 4: Ensembling task performance of CONSENSE's different filtering and weighting schemes. Best results are in bold.

Method	Precision	Recall	F-score
Baseline systems			
Mean	0.3144	0.1631	0.1979
Median	0.3232	0.1698	0.2146
Max	0.7711	0.3177	0.3960
CONSENSE			
Uniform	0.2627	**0.5385**	0.3531
InvRank	**0.3799**	0.3936	0.3867
LogReg	0.3121	0.5095	**0.3871**
SVM+Uniform	0.2891	0.4945	0.3649
SVM+LogReg	0.3256	0.4655	0.3832

as `per:age`) and organization-specific (slots for organization queries such as `org:parents`). We also provide statistics for the types of fillers of each slot, including person names, organization names, locations, dates, and numbers. Precision over all queries was around 0.25, indicating the difficulty of the SF and SFV tasks. Also, note that single-valued slot queries had much higher precision than list-valued queries, perhaps to be expected due to a larger number of fillers. Finally, number and date slots had higher than average precision. This may be due to easier formatting.

5.2 Validation

Next, we measured CONSENSE's performance for the validation task, listed in Table 3. Note that despite all submissions having f-scores between 0.61–0.63, the precision and recall vary greatly, between 0.48–0.59 and between 0.66–0.85 respectively. These ranges indicate suitability for different use cases. Comparing Uniform and InvRank to their SVM variants, we see small increases in f-score, indicating some utility in filtering. Interestingly, InvRank weighting produced highest precision, even over the SVM filtering variants.

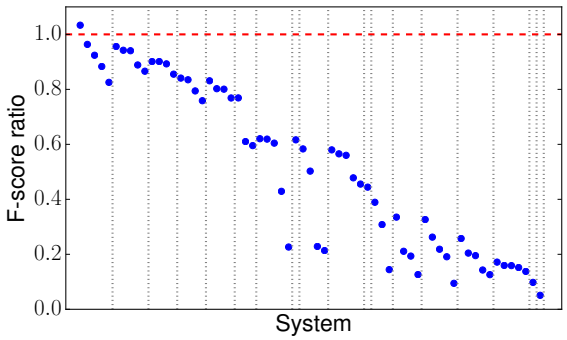

Figure 2: Ratio of slot filling system F-scores to CONSENSE submission 4, SVM + LogReg approach. Only one team outperforms the ensemble.

5.3 Ensembling

Next, we measured CONSENSE's performance on the ensembling task and compared it to the original slot filling system submissions. We evaluated CONSENSE's ensembled output using the same SF scoring program used for the original systems. Table 4 presents performance results for CONSENSE's variants, with summary statistics for baseline system performance. CONSENSE approaches or outperforms all the baseline systems for all evaluation metrics by a wide margin in terms of mean and median. In particular, the CONSENSE variants greatly improve on recall over all baseline systems, while maintaining a comparable precision. While not unexpected given the ensembling task's purpose, it does corroborate CONSENSE's focus on recall. Comparing ensembling with CONSENSE's validation performance (Table 3), we see similar balances between precision and recall for the same filtering and weighting strategies. Again, this indicates CONSENSE's applicability to a range of use cases.

Continuing the analysis, Figure 2 compares CONSENSE to all the slot filling systems in terms of f-score performance. For this analysis we focused on submission 4, with SVM filtering and LogReg weighting. Each point represents a slot filling system and its value is the ratio of CONSENSE's f-score and the system's f-score. Systems are divided into groups based on the team that submitted them. CONSENSE outperforms all baseline systems in terms of f-score, except for one variant. This variant had higher precision, while CONSENSE had higher recall.

5.4 Lambdas

We investigated the λ values produced by CONSENSE's pipeline, trained using a combination of 2013, 2014 and preliminary 2015 data, and using submission 4 (SVM filtering and LogReg weighting). For each slot, we searched a range of λ values between 0.1 and 3.0, de-

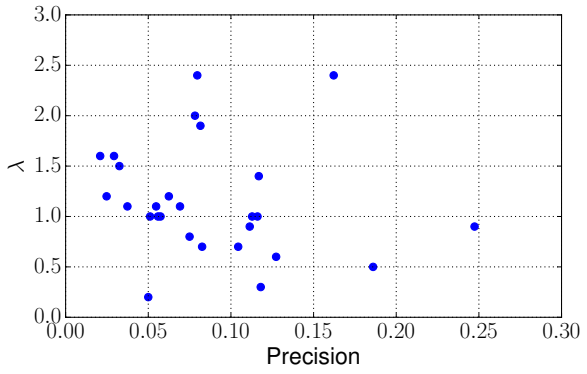

Figure 3: Precision and lambda values for list-valued slots, measured from training data.

termined empirically by examining the precision values and number of fillers. Figure 3 shows precision plotted versus λ, where each data point represents a different list-valued slot. We only plot slots with at least 100 assessed answers—in total, 27 out of 61 slots. In the figure, the precision ranges from 0.024 (`org:parents`) to 0.247 (`gpe:residents_of_country`), and λ values range from 0.2 (`gpe:births_in_city`) to 2.4 (`org:subsidiaries`). The precisions for each slot are relatively low, indicating the difficulty of slot filling in general. The precision values and their corresponding λ are slightly negatively correlated, although not significantly. Also, the mean and median λ were approximately 1.0, indicating a balance between the sparsity term and the scaling term.

5.5 Feature Importance

We analyzed which features presented in Table 1 were most useful in CONSENSE's filtering and scaling. To do so, we used each feature's logistic regression coefficient as a measure of importance. We ranked features by decreasing coefficient absolute value, and computed the median ranks of each feature group (filler, provenance, data type, ensembling). Table 5 presents our results. We see a clear progression of importance, with ensembling features being most useful by a wide margin. This confirms that ensembling across queries and systems serves as CONSENSE's most useful tool for SFV. Interestingly, filler features were ranked least important, essentially negating the most visible aspect of SFV queries.

6 SFV 2015 Experiments

In the previous section, we presented results of experiments performed as if CONSENSE had been submitted to the TAC SFV 2014 evaluation. Here, we show analysis results based on CONSENSE's performance on assessed SFV 2015 data. Note that at the time of writing, only a small subset of the data from TAC SFV 2015 has been assessed by evaluators, so these results should be considered preliminary. Also, we excluded multi-hop queries (see the terminology overview in Section 1) from the evaluation since they were not our primary focus in designing CONSENSE. Finally, recall from Section 3.4 that additional per-team mapping information was available. However, since JHU/APL did not have access to this information prior to the evaluation, we have not rerun our system to exploit this information. Were we to make use of these mappings, our results may significantly improve.

Table 5: Importance of CONSENSE's features, measured by ranks of classifier coefficients.

Feature type	Median rank
Ensembling	10
Data type	19
Provenance	28
Filler	32

Table 6: 2015 validation and ensembling performance for CONSENSE's different filtering and weighting schemes. Best results are in bold. Ensembling precision and recall information were not available.

	Precision	Recall	F-score
Validation			
Uniform	0.2504	**0.4042**	**0.3092**
InvRank	0.2443	0.2452	0.2448
LogReg	**0.2691**	0.3372	0.2993
SVM+LogReg	0.2648	0.3036	0.2828
SVM+Uniform	0.2542	0.3811	0.3050
Ensembling			
Uniform	–	–	**0.4468**
InvRank	–	–	0.3493
LogReg	–	–	0.4320
SVM+LogReg	–	–	0.3924
SVM+Uniform	–	–	0.4138

6.1 Validation and Ensembling

We measured 2015 validation and ensembling performance; results are in Table 6, with best performance in bold. Note that at the time of writing, ensembling precision and recall information were not available in the preliminary 2015 evaluation results. First, we see that the uniform weighting scheme had the best overall performance for both validation and ensembling, in particular for validation recall, over the other schemes which were based on training data from previous years. Comparing with the previous evaluation, we see greatly reduced val-

Table 7: Validation performance for single- and list-valued slots. Best results are in bold.

	Precision	Recall	F-score
Single-valued			
Uniform	0.4611	0.2869	0.3538
InvRank	0.4819	0.2999	0.3697
LogReg	**0.4986**	**0.3103**	**0.3825**
SVM+LogReg	0.4583	0.2757	0.3443
SVM+Uniform	0.4325	0.2602	0.3249
List-valued			
Uniform	0.2251	**0.4493**	0.2999
InvRank	0.1949	0.2241	0.2085
LogReg	**0.2324**	0.3475	0.2785
SVM+LogReg	0.2317	0.3143	0.2668
SVM+Uniform	0.2318	0.4277	**0.3006**

idation performance, and approximately equal or slightly better ensembling performance. Since the non-uniform methods were trained on previous years' data, and this year's data involved many new systems with different characteristics, we attribute these results to the possible overfitting of weighting parameters in the scaling process, as well as aggressive filtering which reduced recall in some cases. However, more analysis is needed to determine the specific causes.

6.2 Single and List-Valued Slots

Next, we separated queries and results by slot, into single-valued and list-valued groups, and measured performance for each group. Table 7 presents our results. Generally, we observe higher precision for single-valued slots, and higher recall for list-valued slots. Also, all weighting methods produced similar performance for single-valued slots, with LogReg having best performance across the board. In contrast, the performance varied widely for list-valued slots, in particular for recall, with the Uniform and SVM+Uniform schemes having much higher recall than the other methods. As before, we attribute this to overfitting based on previous years' data.

6.3 Slot Categories

We then performed a fine-grained slot category analysis using different subsets of slots with shared characteristics.

For this analysis, we used our best-performing 2015 submission (i.e., the Uniform method) and measured validation performance in each category. Table 8 contains our results. We analyzed categories based on slot type (i.e., person (PER), organization (ORG), and geopolitical entity (GPE)) and based on filler type (i.e., location name, human name, organization name, numeric, and date fillers). For slot type categories, we also measured single-valued slots separately. Categories are listed in decreasing order of F-score. For slot types, CONSENSE performed best on PER slots, followed by ORG and GPE slots. For filler types, location, numeric and date filler slots had best performance. These numbers reflect the relative difficulties of slot filling for CONSENSE on these categories. Note that many of the individual categories have fairly high F-scores, which is promising. Future work could include focused research on the more difficult slots, namely ORG and GPE types.

Table 8: Per-category validation performance using uniform weighting. Categories are listed by decreasing F-score.

	Precision	Recall	F-score
Slot type			
PER (single value)	0.5279	0.6910	0.5985
PER	0.4135	0.5794	0.4826
ORG	0.3208	0.2862	0.3026
ORG (single value)	0.4090	0.2273	0.2922
GPE	0.1798	0.4017	0.2484
GPE (single value)	0.4444	0.0661	0.1151
Filler type			
CITY NAME	0.6476	0.6602	0.6538
NUMERIC	0.5556	0.7018	0.6202
LOC NAME	0.5073	0.5939	0.5472
COUNTRY NAME	0.4522	0.6842	0.5445
DATE	0.4507	0.6038	0.5161
STATE NAME	0.4390	0.4737	0.4557
ORG NAME	0.2737	0.4273	0.3337
HUMAN NAME	0.1986	0.3778	0.2604

7 Related Work

Previous work in slot filling validation and ensembling has explored a variety of approaches. Teams have used rule-based validation for filtering, or as a classifier feature for slot fillers ((Tamang et al., 2012; Cheng et al., 2013)). Such rule-based techniques have also used to try to reduce SFV assessment errors, such as Angeli et al. (2013) who employed more sophisticated rule-based checking. Variants of majority voting have also been explored (Sammons et al., 2014), including for baseline comparisons (Viswanathan et al., 2015). Teams have used rich textual entailment to confirm whether slot fillers can be inferred from their provenance documents (e.g., (Cheng et al., 2013; Sammons et al., 2014)). Yu et al. (2014) have used a multi-dimensional truth-finding model (MTM), validating based on multiple credibility scores.

Results from Tamang et al. (2012) indicated that con-

fidence values are generally meaningful. The most similar work to our approach is that of Viswanathan et al. (2015), who used confidence-based ensembling based on stacking features from system output, such as confidences and provenance. However, they did not explore separate strategies for single and list-valued slots. Our previous work did not use filtering or classifier-based weighting for confidence scaling, and used a simpler approach for list slots (Wang et al., 2013).

Finally, two other teams participating in 2015 TAC KBP SFV also used fusion approaches that relied primarily on slot filling confidence outputs, rather than deep source document analysis. Both teams used stacked ensembling, and one of these teams also used JHU/APL's 2013 approach (Wang et al., 2013) to help normalize any unknown team system confidences. However, both systems had access to current and previous participating team names, and thus the corresponding previous year's teams' performances. JHU/APL did not have access to these mappings before the evaluation.

8 Conclusion

CONSENSE demonstrates the effectiveness of confidence scaling for validation and ensembling. Our optimization formulation is scalable, and can be generalized to streaming settings. We have some directions for future work. CONSENSE's current design has no facility for rejecting all fillers for a slot, regardless of how many systems produced the fillers, or how low their confidences are. This greatly benefits CONSENSE's recall at the cost of precision. We could improve this by thresholding or other methods. Also, though we did use features based on the provenance document, more detail could be gleaned from the document's content. Future work could also include focused research on the more difficult slots per Section 6.3, namely ORG and GPE types. Finally, as mentioned in Section 3.4, leveraging per-team mappings could greatly benefit the performance of our weighting schemes, in particular the inverse precision rank scheme.

References

Angeli, Gabor, Arun Chaganty, Angel Chang, Kevin Reschke, Julie Tibshirani, Jean Y. Wu, Osbert Bastani, Keith Siilats, and Christopher D. Manning. 2013. Stanford's 2013 KBP system. In *TAC'13: Proceedings of the 6th Text Analysis Conference*. Gaithersburg, MD.

Cheng, Xiao, Bingling Chen, Rajhans Samdani, Kai-Wei Chang, Zhiye Fei, Mark Sammons, John Wieting, Subhro Roy, Chizheng Wang, and Dan Roth. 2013. Illinois Cognitive Computation Group UI-CCG TAC 2013 entity linking and slot filler validation systems. In *TAC'13: Proceedings of the 6th Text Analysis Conference*. Gaithersburg, MD.

Predd, Joel B., Daniel N. Osherson, Sanjeev R. Kulkarni, and H. Vincent Poor. 2008. Aggregating probabilistic forecasts from incoherent and abstaining experts. *Decision Analysis* 5(4):177–189.

Rand, William M. 1971. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association* 66(336):846–850.

Sammons, Mark, Yangqiu Song, Ruichen Wang, Gourab Kundu, Chen-Tse Tsai, Shyam Upadhyay, Siddarth Ancha, Stephen Mayhew, and Dan Roth. 2014. Overview of UI-CCG systems for event argument extraction, entity discovery and linking, and slot filler validation. In *TAC'14: Proceedings of the 7th Text Analysis Conference*. Gaithersburg, MD.

Surdeanu, Mihai. 2013. Overview of the TAC2013 knowledge base population evaluation: English slot filling and temporal slot filling. In *TAC'13: Proceedings of the 6th Text Analysis Conference*. Gaithersburg, MD.

Tamang, Suzanne, Zheng Chen, and Heng Ji. 2012. CUNY BLENDER TAC-KBP2012 entity linking system and slot filling validation system. In *TAC'12: Proceedings of the Text Analysis Conference*. Gaithersburg, MD.

Viswanathan, Vidhoon, Nazneen Fatema Rajani, Yinon Bentor, and Raymond Mooney. 2015. Stacked ensembles of information extractors for knowledge-base population. In *ACL'15: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China, pages 177–187.

Wang, I-Jeng, Edwina Liu, Cash Costello, and Christine Piatko. 2013. JHUAPL TAC-KBP2013 slot filler validation system. In *TAC'13: Proceedings of the 6th Text Analysis Conference*. Gaithersburg, MD.

Yu, Dian, Hongzhao Huang, Taylor Cassidy, Heng Ji, Chi Wang, Shi Zhi, Jiawei Han, Clare Voss, and Malik Magdon-Ismail. 2014. The wisdom of minority: Unsupervised slot filling validation based on multi-dimensional truth-finding. In *COLING'14: Proceedings of the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin, Ireland, pages 1567–1578.