Inconsistencies in Crowdsourced Slot-Filling Annotations: A Typology and Identification Methods

Stefan Larson ♣*† **Adrian Cheung**♦ **Anish Mahendran** ♠†
Kevin Leach ♠† **Jonathan K. Kummerfeld** ♠†

♣Rosegold AI
Ann Arbor, MI, USA

♦Clinc, Inc.
Ann Arbor, MI, USA

♠University of Michigan
Ann Arbor, MI, USA

Abstract

Slot-filling models in task-driven dialog systems rely on carefully annotated training data. However, annotations by crowd workers are often inconsistent or contain errors. Simple solutions like manually checking annotations or having multiple workers label each sample are expensive and waste effort on samples that are correct. If we can identify inconsistencies, we can focus effort where it is needed. Toward this end, we define six inconsistency types in slot-filling annotations. Using three new noisy crowd-annotated datasets, we show that a wide range of inconsistencies occur and can impact system performance if not addressed. We then introduce automatic methods of identifying inconsistencies. Experiments on our new datasets show that these methods effectively reveal inconsistencies in data, though there is further scope for improvement.

1 Introduction

Slot-filling is a key component of task-driven dialog systems, providing a way for systems to extract key properties from user queries. For example, a slot-filling model can extract the tokens "New York" as a `TO_LOCATION` slot in the query "book a flight to New York". Standard slot-filling models train or finetune on large datasets of carefully-annotated data that is domain specific. This means there is an ongoing need for annotating new datasets for new domains.

Typically, data annotation is completed via small tasks with brief instructions completed by non-expert crowd workers. Collecting high quality data in this setting is challenging, involving multiple rounds of pilot annotations and analysis to develop suitable instructions (Alonso et al., 2015). Figure 1 shows examples of inconsistencies in annotations performed by crowd workers. Training on data with these issues will lead to lower quality models, which in turn decrease the effectiveness of the overall dialog system. Most research on improving data quality has focused on mechanisms such as aggregation (Parde and Nielsen, 2017), worker filtering (Li and Liu, 2015), and attention checks (Oppenheimer et al., 2009). These all raise costs and primarily address clear inconsistencies (such as in examples 1, 4, 5, and 6) but not more subtle cases like the inclusion of "dollar" in examples 2 and 3. Additionally, annotation-as-a-service (e.g., `scale.ai`) is being widely used by developers but often cannot be customized by the developer to add these kinds of mechanisms. If we could identify all of these inconsistencies then we could fix them without expending effort on examples that were correctly annotated and we could improve task instructions to reduce further inconsistencies.

In this paper, we catalog a typology of annotation inconsistencies that occur in slot-filling data and present several automatic methods for identifying inconsistencies. To demonstrate our ideas, we introduce and analyze three new crowd-annotated datasets.[1] By analyzing the data with our typology, we find that the distribution of errors varies substantially depending on the domain. We also measure the impact of these inconsistencies on trained models and examine the relative impact of each inconsistency type through a controlled experiment where we artificially inject errors.

*Please direct correspondence to: `stefan.dataset@gmail.com`.

†Work conducted while author was employed by Clinc.

[1]Datasets can be found at `https://tinyurl.com/slot-inconsistencies`.

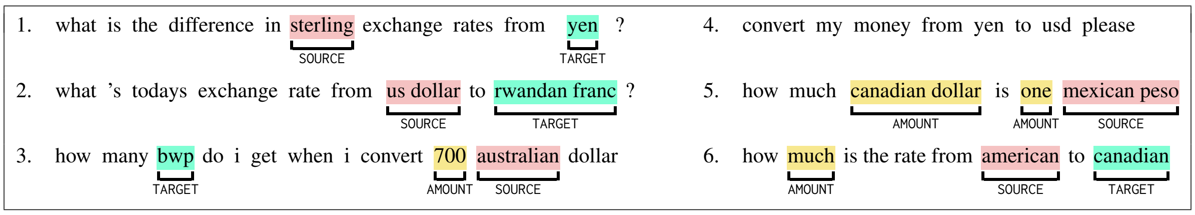
1. what is the difference in sterling exchange rates from yen ?
2. what 's todays exchange rate from us dollar to rwandan franc ?
3. how many bwp do i get when i convert 700 australian dollar
4. convert my money from yen to usd please
5. how much canadian dollar is one mexican peso
6. how much is the rate from american to canadian

Figure 1: Examples of inconsistent annotations by crowd workers. In 1, the SOURCE and TARGET labels are backwards. In 2 and 3, there is variation in whether "dollar" is included in the SOURCE span. In 4, annotations for "yen" and "usd" are missing. In 5, the label for "canadian dollar" is incorrect. In 6, "much" is annotated, but in 5 it is not.

We also propose several inconsistency identification methods that require no additional annotation, using only the data already being collected. We evaluate by comparing to manually checking every example, measuring effort required and the quality of models trained on the resulting data. The approaches have different strengths and weaknesses. One reduces effort by 16-31% and produces models within 1.6 F_1. Another reduces effort by 50-87%, but leads to weaker models. These results indicate that inconsistency identification is possible, but there is scope to further improve the tradeoff between effort and data quality. This work provides the basis for a new direction in research for addressing inconsistencies in crowdsourcing by defining a typology of inconsistency types, collecting new benchmark datasets, and exploring directions for automatic identification of inconsistencies.

2 Related Work

2.1 Annotation Consistency

Inconsistencies have been studied across a wide range of tasks. Part-of-speech tagging has received particular attention, with a range of methods based on model scores (Abney et al., 1999; Eskin, 2000; Matsumoto and Yamashita, 2000; van Halteren, 2000; Ma et al., 2001; Nakagawa and Matsumoto, 2002). One particular method based on variation in POS tags of n-grams (Dickinson and Meurers, 2003) has been extended to predicate-argument relations (Dickinson and Lee, 2008), and dependency parses (Dickinson, 2010; Dickinson and Smith, 2011). More recent work has explored automatic identification methods for word sense and multi-word-expression annotation (Dligach and Palmer, 2011; Hollenstein et al., 2016).

2.2 Crowdsourcing Quality

A range of options have been developed for improving crowdsourcing quality. The most common approach is to collect multiple annotations and then aggregate them (Hovy et al., 2013; Passonneau and Carpenter, 2014; Parde and Nielsen, 2017; Dumitrache et al., 2018). This can identify inconsistencies, but at significant cost as each example must be annotated multiple times. Less expensive options include using examples with known answers to test worker attention (Oppenheimer et al., 2009), or filtering workers based on qualification tasks or preliminary annotation (Li and Liu, 2015; Roit et al., 2020). However, these primarily address worker attentiveness, which does not cover all the inconsistency types we consider. In particular, we also cover inconsistencies related to subtle cases without an obvious correct answer.

2.3 Error Type Categorization

Another line of work has explored automatic identification of error types for tasks such as constituency parsing (Kummerfeld et al., 2012; Kummerfeld et al., 2013), coreference resolution (Kummerfeld and Klein, 2013), semantic role labeling (He et al., 2017), and slot-filling (Béchet and Raymond, 2018). However, they focus on evaluating *system* outputs in comparison to a gold standard reference in order to understand shortcomings of the systems. One notable exception (Niu and Penn, 2019) establishes a taxonomy of annotation errors, although this is for a single corpus (ATIS (Hemphill et al., 1990)).

Type	Example	Explanation
Slot Format	1. change $\underbrace{\text{twenty}}_{\text{AMOUNT}}$ $\underbrace{\text{us dollars}}_{\text{SOURCE}}$ to the $\underbrace{\text{british pound}}_{\text{TARGET}}$ 2. convert $\underbrace{\text{1200}}_{\text{AMOUNT}}$ of $\underbrace{\text{new zealand}}_{\text{SOURCE}}$ dollars to $\underbrace{\text{the cad}}_{\text{TARGET}}$ 3. what is the exchange rate between $\underbrace{\text{pesos}}_{\text{SOURCE}}$ and british $\underbrace{\text{pounds}}_{\text{TARGET}}$	These are cases where the boundaries of a slot are inconsistent. In these examples, "british" is part of TARGET in 1 but not 3, and "the" is part of TARGET in 2 but not 1, and "dollars" is part of SOURCE in 1 but not 2.
Omission	1. if i have $\underbrace{\text{pounds}}_{\text{SOURCE}}$ then how many japanese yen would that be 2. please convert mexican pesos to $\underbrace{\text{canadian dollars}}_{\text{TARGET}}$	When an unlabeled span should be labeled as a slot. Here, "japanese yen" should be TARGET and "mexican pesos" should be SOURCE.
Addition	1. how $\underbrace{\text{many}}_{\text{AMOUNT}}$ $\underbrace{\text{pesos}}_{\text{SOURCE}}$ is $\underbrace{\text{twelve}}_{\text{AMOUNT}}$ $\underbrace{\text{dollars}}_{\text{TARGET}}$	A span is labeled that should not be. Here, "many" should not be labeled.
Wrong Label	1. show exchange rate between the $\underbrace{\text{dollar}}_{\text{AMOUNT}}$ and the $\underbrace{\text{yen}}_{\text{TARGET}}$ 2. what is the rate of $\underbrace{\text{lari}}_{\text{SOURCE}}$ to $\underbrace{\text{chf}}_{\text{SOURCE}}$	When a token span is labeled as one slot type but should be labeled as another. Here "dollar" should be SOURCE and "chf" should be TARGET.
Swapped Labels	1. please convert $\underbrace{\text{twenty}}_{\text{AMOUNT}}$$\underbrace{\text{british pounds}}_{\text{TARGET}}$ into $\underbrace{\text{american dollars}}_{\text{SOURCE}}$ 2. what is the exchange rate to $\underbrace{\text{swiss franc}}_{\text{SOURCE}}$ from $\underbrace{\text{korean won}}_{\text{TARGET}}$	When a pair of annotated slots have each other's label and the labels form a natural pair. Here, SOURCE and TARGET are flipped.
Chop and Join	1. what is $\underbrace{\text{400}}_{\text{AMOUNT}}$ $\underbrace{\text{canadian}}_{\text{SOURCE}}$ $\underbrace{\text{dollars}}_{\text{SOURCE}}$ in $\underbrace{\text{euros}}_{\text{TARGET}}$ 2. what is the conversion from $\underbrace{\text{south korean won}}_{\text{SOURCE}}$ to $\underbrace{\text{canadian dollars}}_{\text{TARGET}}$ 3. exchange $\underbrace{\text{new zealand dollar}}_{\text{SOURCE}}$ for $\underbrace{\text{hong kong dollars}}_{\text{TARGET}}$	When sometimes one span is used and sometimes two spans are used. Here, "canadian dollar" is labeled as a SOURCE slot in two parts in the first case, but not the others.

Table 1: Examples of the inconsistency types we propose in Section 3.

Recent work has also explored error types for crowdsourced paraphrases (Yaghoub-Zadeh-Fard et al., 2019), another important aspect of dialogue data collection.

3 Inconsistencies

In this section, we define six inconsistency types, measure their frequency in three new datasets, and evaluate their impact on model performance. We developed the set of inconsistencies by manually inspecting crowd-annotated data and looking for patterns in the common mistakes.

3.1 Types of Inconsistencies

We identify six inconsistency types that arise in the crowdsourcing setting when multiple non-expert human annotators work independently. Table 1 briefly describes the types through examples. Some inconsistencies arise when there is ambiguity in an aspect of the conventions for labeling slots, with different workers resolving the ambiguity differently. Other inconsistencies are mistakes that directly violate the specified annotation request, arising due to either confusion or rushed effort.

3.1.1 *Slot Format* Inconsistency

In some cases, it is clear that a slot is present, but the precise boundaries of the slot may be unclear. For example, consider labeling an ACCOUNT in the text "to my checking account please". There are four possible reasonable annotations:

1. "to $\underbrace{\text{my checking account}}_{\text{ACCOUNT}}$ please"
2. "to my $\underbrace{\text{checking account}}_{\text{ACCOUNT}}$ please"
3. "to $\underbrace{\text{my checking}}_{\text{ACCOUNT}}$ account please"
4. "to my $\underbrace{\text{checking}}_{\text{ACCOUNT}}$ account please"

Depending on how the extracted slots are used, which of these is right may differ. For example, the "my" indicates that the checking account belongs to the user, which could be important for distinguishing it from another context (e.g. "Sarah's checking"). It is possible that any of these would work for some downstream uses, but training with inconsistent labels will present a confusing training signal for models.

3.1.2 *Omission* Inconsistency

This covers cases where a span is meant to be labeled with a certain slot label, but is not. Consider the following examples:

5. “transfer $\underbrace{\text{30 dollars}}_{\text{AMOUNT}}$ to $\underbrace{\text{savings}}_{\text{ACCOUNT}}$ please”
6. “please withdraw $\underbrace{\text{45 bucks}}_{\text{AMOUNT}}$ from savings”
7. “transfer over $\underbrace{\text{\$200}}_{\text{AMOUNT}}$ now from $\underbrace{\text{checking}}_{\text{ACCOUNT}}$”

The span “savings” should be labeled as ACCOUNT in example 6, but is not. This might occur if a worker is moving quickly through examples and submits after assigning the first label.

3.1.3 *Addition* Inconsistency

These occur when a span is labeled with a slot label but should be unlabeled. Consider the following:

8. “do you mind $\underbrace{\text{checking}}_{\text{ACCOUNT}}$ my $\underbrace{\text{savings}}_{\text{ACCOUNT}}$ please”

“savings” is a valid ACCOUNT, but “checking” is not. This might occur if a worker is simply pattern matching on key terms rather than carefully reading each example.

3.1.4 *Wrong Label* Inconsistency

This is when a span is correctly identified as a slot, but assigned an incorrect label. For example:

9. “transfer $\underbrace{\text{30 dollars}}_{\text{ACCOUNT}}$ please”

Here, “30 dollars” is marked as a slot, but with ACCOUNT instead of AMOUNT. These could be the result of a typo or mis-click when using an annotation tool, or due to a misunderstanding of the slot.

3.1.5 *Swapped Label* Inconsistency

This is a special case of *Wrong Label* inconsistency. A *Swapped Label* inconsistency is when (a) two spans in the same sample have the wrong label, but would be correct if they were swapped, and (b) the two labels are for the same item type (e.g., a currency, or a location). For example, consider labels for the SOURCE and TARGET currencies below:

10. “how much is $\underbrace{\text{one}}_{\text{AMOUNT}}$ $\underbrace{\text{usd}}_{\text{TARGET}}$ in $\underbrace{\text{gbp}}_{\text{SOURCE}}$ ”
11. “show the exchange rate between $\underbrace{\text{dollars}}_{\text{SOURCE}}$ and $\underbrace{\text{yen}}_{\text{TARGET}}$ ”

The first example clearly has the two labels reversed. The second example is ambiguous and so may be annotated one way by some workers and the reverse by others.

3.1.6 *Chop* and *Join* Inconsistency

These occur when either a span is labeled as one slot where several annotations are appropriate (*Join*), or when a span is labeled as several annotations where a single annotation is appropriate (*Chop*). Consider the following two samples related to scheduling a flight departure date (DATE):

12. “buy tickets for $\underbrace{\text{jan 14 in 2020}}_{\text{DATE}}$”
13. “schedule on the $\underbrace{\text{20th}}_{\text{DATE}}$ of $\underbrace{\text{december}}_{\text{DATE}}$ $\underbrace{\text{2016}}_{\text{DATE}}$”

In the first sample, the DATE slot is applied to one continuous token span, but in the second sample the DATE slot is applied to each individual attribute (i.e., day, month, and year).

3.1.7 *Other*

When annotating inconsistencies we also include an *Other* category. This covers annotations that do not fit within the cases defined above, but do seem inconsistent with the rest of the annotations.

Dataset	# Slot Types	Example
Forex	3	if i have $\underbrace{\text{23}}_{\text{AMOUNT}}$ $\underbrace{\text{chinese renminbi}}_{\text{SOURCE}}$ then how many $\underbrace{\text{korean won}}_{\text{TARGET}}$ could i get
Companies	7	what is the top $\underbrace{\text{auto}}_{\text{SECTOR\_NAME}}$ company with a $\underbrace{\text{p/e ratio}}_{\text{METRIC}}$ above $\underbrace{\text{224 billion}}_{\text{NUMERIC}}$ dollars ?
Flights	6	need $\underbrace{\text{economy}}_{\text{CLASS}}$ class flight to $\underbrace{\text{st. louis}}_{\text{TO\_LOCATION}}$ from $\underbrace{\text{la guardia}}_{\text{FROM\_LOCATION}}$

Table 2: Examples from each dataset used in our experiments.

Dataset	Omission	Addition	Slot Format	Wrong Label	Swapped Label	Chop-Join	Other	Any
Forex	7.8	2.9	9.4	4.8	6.3	0.0	1.3	27.5
Companies	33.4	5.0	35.2	12.4	0.0	0.0	12.0	67.6
Flights	15.8	1.8	25.4	4.2	0.6	0.6	0.6	44.6

Table 3: The percentage of samples in each dataset with each inconsistency type.

3.2 Datasets

We crowdsourced slot annotations for datasets in three domains to analyze the frequency of each inconsistency type and the effect inconsistencies have on slot-filling model performance. Table 2 shows examples from each dataset and the number of different slot types. The sentences used for two (Forex and Companies) were collected using the crowdsourcing approach described by Larson et al. (2019) and sentences for the third (Flights) were sampled from prior work (Jaech et al., 2016).

Forex This dataset consists of short queries about currency exchange rates. Crowd workers were asked to annotate spans with three slots: SOURCE, TARGET, and AMOUNT.

Companies This contains queries about metrics measuring properties of companies. There are seven slot types: SECTOR_NAME, METRIC, NUMERIC, LOCATION_OUTSIDE_NAME, LOCATION_INSIDE_NAME, RANK, and DATE_METRIC. Annotating this dataset is challenging because there are more slot types and the setting is probably unfamiliar to most crowd workers due to the domain's technical nature.

Flights These queries are about booking flights between US cities. The slots are: CLASS, TO_LOCATION, FROM_LOCATION, DEPART_DATE, RETURN_DATE, and NUM_PASSENGERS.

The Flights and Companies datasets contain 500 samples each, and Forex contains 520. All samples are in English. We crowdsourced slot label annotations using Amazon Mechanical Turk, paying workers $0.10 per sample. Each sample was annotated by one worker, who was asked to annotate all slot types present. Instructions contained (1) descriptions of each slot label type, and (2) examples of annotations. The crowdsourcing task prompts contained basic annotated examples, and the instructions and examples did not emphasize any particular policy regarding possible inconsistencies.

3.3 Manually Labeling Inconsistencies

Once annotated by the crowd, we manually checked each annotated sample for inconsistencies. Inconsistencies in the data were labelled independently by two of the authors of this paper, who then discussed and reconciled any disagreements. We did not predefine an annotation policy for every slot, particularly the *Slot Format* inconsistencies. Instead, we looked at the data and followed the dominant behavior of workers over all similar annotations in a dataset, labeling annotations in the minority as inconsistencies. We used a search tool (Larson et al., 2020) to help find all relevant samples to determine the majority annotation scheme. For the Flights dataset we also used the original annotations to help identify inconsistencies (but again, followed the crowd majority conventions even when they deviated from the original data).

	Dataset		
Train	**Forex**	**Companies**	**Flights**
Raw Crowd Annotations (Crowd)	0.83	0.51	0.77
Slot Format Inconsistencies Resolved (Formatted)	0.86	0.57	0.87
Consistent Annotations (Consistent)	0.95	0.90	0.94

Table 4: Slot-filling F_1 scores when training on data with different levels of inconsistency. The evaluation sets are composed of consistently annotated data.

3.4 Inconsistency Rates

Table 3 (previous page) shows the observed rates of each inconsistency type in each dataset. Rates are calculated by dividing the number of samples containing the inconsistency type by the number of samples in the dataset. (Note that *Swapped Label* cases are not included in the count of *Wrong Label* cases.) The frequency of samples having *any* type of inconsistency is quite high, ranging from 27.5% to 65%. Datasets with more slot types have more inconsistencies. The Companies dataset has the highest rate of inconsistencies, confirming our hypothesis that this is a challenging dataset for non-experts to annotate. The *Slot Format* inconsistency is the most common type for all of the datasets and *Omission* is the second most common type. This observation guided the focus of the next section of the paper, where we develop automatic methods for finding inconsistencies. As expected, datasets with slot values that can be one of two slot types (e.g., `TO_LOCATION` and `FROM_LOCATION`) have a higher occurrence of swapped labels.

3.5 Measuring the Impact on Model Performance

Given that annotated slot-filling data is used to train slot-filling models, we study the impact that the noisy datasets have on slot-filling performance. We trained slot-filling models on three versions of each dataset: the raw crowd-annotated dataset (Crowd), a version in which we fixed *Slot Format* inconsistencies (Formatted), and a version in which we fixed all inconsistencies (Consistent). We include the Formatted version in our analysis since *Slot Format* inconsistencies are the most common.

Each model was tested on a test set with consistent annotations. We used a Bi-LSTM architecture similar to Finegan-Dollak et al. (2018)'s template-based approach with randomly-initialized word embeddings for the slot-filling model. We computed the slot F_1 score using 10-fold cross-validation.

Results Table 4 shows slot-filling F_1 scores for each training set. The difference in model performance when trained on noisy versus consistently annotated training samples is substantial, with the Companies dataset seeing an absolute difference of roughly 40 percent. The differences for the Forex and Flights datasets are also quite profound. We also note that fixing *Slot Format* inconsistencies (Formatted) yields model improvements, but certainly not enough to reach Consistent performance, indicating that the other, less frequent inconsistencies cause substantial model degradation.

Inconsistency Type Impact To investigate the relative impact of each inconsistency type, we performed a second experiment in which we introduced artificial inconsistencies into the Consistent version of the Flights dataset. Separately for each inconsistency type, we automatically modified a fraction of the training samples to introduce that inconsistency. We trained models and measured performance. Figure 2 (next page) shows that an increase of any type of inconsistency negatively affects model performance. *Swap* and *Chop* inconsistencies have the largest effect on model performance. Fortunately, they are rare in our crowdsourced data (see Table 3). *Addition* inconsistencies had relatively little impact, but are also relatively rare. The remaining types, which are the more common ones, all have similar impacts.

The results from these experiments demonstrate that model performance is degraded when training on inconsistently-annotated data. This observation is the motivation of our next goal: automatically detecting inconsistent annotations.

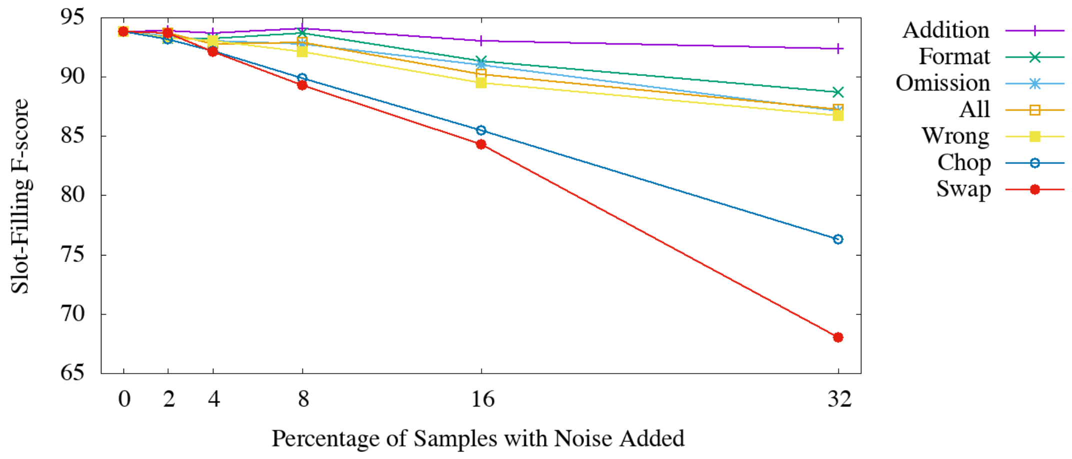

Figure 2: Slot-filling F_1 scores with various levels of artificial inconsistency injected into the Flights dataset. The *All* case is an even distribution of the six inconsistency types. The legend is ordered to match the order of lines.

4 Automatic Inconsistency Detection

Addressing the challenge of inconsistent annotations usually involves a combination of two approaches: improving annotation instructions and manual post-processing. Developing better instructions is a challenge, as it requires either foreseeing all scenarios where there may be inconsistency in annotation conventions, or looking at enough annotated data to identify inconsistencies across examples. Even if comprehensive instructions are developed, there is no guarantee that annotators will follow the conventions perfectly. Manual post-processing can address these issues, but without any means of guiding effort to potentially problematic examples it involves a substantial amount of effort. Collecting multiple annotations of each sample can reduce these issues, but increases costs and will not resolve all cases as the majority annotation for one example may be inconsistent with the majority for another example.

In this section, we explore methods of automatic inconsistency detection. These could be used either to assist in refinement of instructions or to guide post-processing to focus effort. We propose a range of automatic inconsistency detection algorithms that are a first step towards addressing this challenge.

The setting we consider is when a new dataset has been annotated by crowd workers with one annotation per example. There is no in-domain gold standard annotated data available and no other data with the same annotation scheme available. This matches the application described in Section 1: developing a dialog system for a new domain with its own set of slots.

4.1 Format Checker

The Format Checker is specifically tailored to detect *Slot Format* inconsistencies. It has three steps: (1) form sets of characteristic left and right n-grams for slots, (2) check when text adjacent to a span is in an n-gram set, (3) use majority voting to suggest an n-gram should be part of a slot or not. We say a sample has been "flagged" if it contains a token span that has a suggested change. Suggestions to flagged samples can be applied automatically, or on a case-by-case basis. We explain these three steps in more detail below.

N-gram sets The first step is to construct left- and right-n-gram sets. A left n-gram set is the set of all token sequences in a span that do not include the last token. For example, for the text span "my premier savings account", the left-n-gram set is {"my", "premier", "savings", "my premier", "premier savings", "my premier savings"}. A right n-gram set is the same, except it excludes the first token of a span rather than the last one. We build these sets over all text spans in the data for each slot.

Checking adjacent tokens Using the n-gram sets, we identify cases where a boundary may be inconsistent. For each slot in each example, we consider the text to the left of the slot and see if it is in the left-n-gram set. Similarly, we check if the text to the right of the slot is in the right-n-gram set. If there is a match, then there may be an inconsistency, where the adjacent text should be part of the span. For example, consider these three samples:

1. use $\underbrace{\text{primary savings account}}_{\text{ACCOUNT}}$ please
2. from primary $\underbrace{\text{checking account}}_{\text{ACCOUNT}}$ last week
3. please show $\underbrace{\text{primary checking account}}_{\text{ACCOUNT}}$

All three samples contain an ACCOUNT slot. The left n-gram set for these samples is {“primary”, “primary savings”, “savings”, “checking”, “primary checking”}. When checking the second example, the algorithm would identify “primary” as a word that is in the left n-gram set but not in the span, which indicates that there may be a span inconsistency here.

Voting The previous step provides a set of possible inconsistencies. For each one, we then use a voting process to give further information about what the dominant convention appears to be. This works by identifying how frequently that n-gram is included in the span and how often it is adjacent to the span. If it is included more often then the suggestion is to add it in the other cases, otherwise the suggestion is to remove it from the cases it is used in. In the example above, including “primary” is more common than excluding it, and so the suggestion would be to add it in the second sample.

4.2 Label Variation Detector

This approach is an adaptation of the variation n-gram method proposed by (Dickinson and Meurers, 2003) for detecting errors in part-of-speech annotations. The approach involves two steps: (1) extract n-gram patterns, (2) use majority vote on annotations in the n-gram patterns to identify the most common pattern. These suggestions can be applied automatically, or on a case-by-case basis.

N-gram patterns In this step, for each slot in each example we extract an n-gram consisting of the slot and *k* tokens either side. Consider these four examples:

1. “... from my $\underbrace{\text{checking}}_{\text{SOURCE}}$ to her ...”
2. “... use my $\underbrace{\text{checking}}_{\text{TARGET}}$ to send ...”
3. “... with my $\underbrace{\text{checking}}_{\text{SOURCE}}$ to provide ...”
4. “... out of my checking to my ...”

If *k* is one, the n-gram extracted would be “my checking to”. In these examples, the n-gram receives a SOURCE label in the first and third cases, TARGET label in the second, and no annotation in the fourth.

Identifying the most common annotation For each n-gram in the previous step, we count how many times each annotation occurs. The most common annotation is suggested as the way to standardize and all examples with the n-gram are presented. In the example above, this means that the suggestion would be to add a label in the fourth case and change the label in the second case.

4.3 FCLVD: Combining Methods

The two methods discussed so far are designed to be complementary, with the Format Checker designed to be used first to detect and fix *Slot Format* inconsistencies, and then the Label Variation Detector designed to detect everything else. We can combine these methods together to detect a larger set of potential inconsistencies.

4.4 CRF Agreement

In this method, we use cross-validation to train models on the data and also predict labels for the data. Any disagreement between the prediction and the data is considered an inconsistency. We use a conditional random field (CRF) (Lafferty et al., 2001) as it is very fast to train and is less likely to overfit the data than a large neural model. The reasoning behind this approach is that if there is an inconsistency the CRF will learn one of the conventions, leading to disagreements when the other convention is present.

Dataset	Method	Precision	Recall	F_1 Score	% Flagged
Forex	CRF	0.38	**0.96**	**0.54**	69
(27.5%)	FCLVD	**0.74**	0.36	0.49	13
Companies	CRF	0.78	**0.97**	**0.86**	84
(67.6%)	FCLVD	**0.65**	0.48	0.62	50
Flights	CRF	0.56	**0.89**	0.69	71
(44.6%)	FCLVD	**0.86**	0.70	**0.78**	36

Table 5: Performance comparison of inconsistency detection methods. Higher is better for Precision, Recall, and F_1, while lower is better for % Flagged. The percentage in parenthesis indicates what proportion of the examples in the dataset contain an inconsistency (from Table 3).

	Dataset		
Method	**Forex**	**Companies**	**Flights**
Format Checker	0.69	0.72	0.96
Label Variation Detector	0.12	0.31	0.13
Format Checker + Label Variation Detector (FCLVD)	**0.71**	**0.77**	**0.98**

Table 6: Recall scores for various inconsistency detection methods on *Slot Format* inconsistencies. The Format Checker is tailored toward finding *Slot Format* inconsistencies, and hence has a much higher recall than the Label Variation Detector. The FCLVD approach yields an even higher recall.

4.5 Model Discussion

One strength of the Format Checker and Label Variation Detector is that they produce very interpretable results because they use pattern matching. Not only do the algorithms specify the token range of a flagged inconsistency, they also inform the user of precisely which other samples are annotated in a manner that is inconsistent with the flagged sample. In contrast, the CRF Agreement method provides no cross-example feedback, only specifying that a given annotation does not match its predictions.

On the other hand, the CRF Agreement approach is more flexible, capable of identifying any error type, including those in the *Other* category. It is possible that future work could develop specific detection methods for more error types, but that remains an open question.

5 Evaluation

5.1 Experiments

We evaluated inconsistency detection performance on our three crowd-annotated datasets (§ 3.2). We measured inconsistency detection recall, precision, F_1 score, and % Flagged:

$$\text{Recall} = \frac{\text{\# inconsistent samples flagged}}{\text{\# inconsistent samples}} \qquad \text{Precision} = \frac{\text{\# inconsistent samples flagged}}{\text{\# samples flagged}} \qquad \text{\% Flagged} = \frac{\text{\# samples flagged}}{\text{\# samples}}$$

Precision, recall, and F_1 are standard measures of performance. The % Flagged indicates the proportion of the samples that were flagged as containing inconsistencies. In practice this means the proportion of the data that might need to be verified by an expert. This metric is related to precision, but more directly measures the effort required when using the method.

5.2 Results

Table 5 shows results for all three datasets with the CRF method and the FCLVD method that combines the Format Checker and Label Variation Detector. The CRF has consistently higher recall, catching almost all of the inconsistencies. However, that comes at the cost of creating more work, as the % flagged is consistently higher, by a factor of five in the case of the Forex data. Table 5 presents a tradeoff between the two approaches: the CRF has higher recall but flags more samples than the actual observed inconsistency rates, while the FCLVD method flags fewer and has higher precision.

	Dataset		
Method	**Forex**	**Companies**	**Flights**
No corrections	0.83	0.51	0.77
FCLVD	0.87	0.63	0.91
CRF	0.95	0.89	0.92
Consistent	**0.95**	**0.90**	**0.94**

Table 7: Slot-filling F_1 scores when training on data fixed based on the output of each method. The evaluation sets are composed of consistently annotated data.

Table 6 (previous page) highlights the difference in performance between the Format Checker and Label Variation Detector methods on *Slot Format* inconsistencies. We analyze this inconsistency type in particular since it is the most common type (see Table 3). The Format Checker has much higher recall on this inconsistency type than Label Variation Detector, which is expected since the Format Checker is specifically designed to detect *Slot Format* inconsistencies. When combined together, the two methods have higher recall on *Slot Format* inconsistencies than both methods individually.

Finally, Table 7 shows slot-filling F_1 scores when training on datasets with corrections based on each automatic inconsistency detection approach. For FCLVD, the improvement can be as large as 14 points, coming within 3 points of consistent data with checking only applied to 36% of examples (from Table 5). CRF does even better, reaching within 1.6 points of consistent data on all three datasets, but as shown in Table 5, it requires checking for 69-84% of the examples.

6 Conclusion

Understanding the nature and frequency of different types of annotation inconsistencies in noisy slot-filling corpora is important for the development of robust task-driven dialog systems. This paper presents a typology of inconsistency types. Applying the typology to three new crowdsourced datasets, we find the overall inconsistency rate is high, and the *Slot Format* and *Omission* types are the most common.

We show that correcting inconsistencies improves the quality of models and propose several methods of automatically detecting inconsistencies. By evaluating our approaches we find no clearly dominant choice, and that there is scope for further work to balance detection recall with the overall number of samples flagged as inconsistent. By detecting inconsistencies we can fix issues in annotations, improve task instructions, and better understand the challenges of slot-filling.

Acknowledgements

We thank Andrew Lee for the Companies dataset, and the anonymous reviewers for their constructive feedback.

References

Steven Abney, Robert E. Schapire, and Yoram Singer. 1999. Boosting applied to tagging and pp attachment. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Omar Alonso, Catherine C. Marshall, and Marc Najork. 2015. Debugging a crowdsourced task with low inter-rater agreement. In *Proceedings of the 15th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL)*.

Frédéric Béchet and Christian Raymond. 2018. Is ATIS too shallow to go deeper for benchmarking Spoken Language Understanding models? In *Proceedings of Interspeech 2018*.

Markus Dickinson and Chong Min Lee. 2008. Detecting errors in semantic annotation. In *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC)*.

Markus Dickinson and W. Detmar Meurers. 2003. Detecting errors in part-of-speech annotation. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*.

Markus Dickinson and Amber Smith. 2011. Detecting dependency parse errors with minimal resources. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT)*.

Markus Dickinson. 2010. Detecting errors in automatically-parsed dependency relations. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Dmitriy Dligach and Martha Palmer. 2011. Reducing the need for double annotation. In *Proceedings of the 5th Linguistic Annotation Workshop (LAW)*.

Anca Dumitrache, Lora Aroyo, and Chris Welty. 2018. Crowdsourcing ground truth for medical relation extraction. *ACM Trans. Interact. Intell. Syst.*, 8(2).

Eleazar Eskin. 2000. Detecting errors within a corpus using anomaly detection. In *1st Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2017. Deep semantic role labeling: What works and what's next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Charles T. Hemphill, John J. Godfrey, and George R. Doddington. 1990. The ATIS spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop*.

Nora Hollenstein, Nathan Schneider, and Bonnie Webber. 2016. Inconsistency detection in semantic annotation. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC)*.

Dirk Hovy, Taylor Berg-Kirkpatrick, Ashish Vaswani, and Eduard Hovy. 2013. Learning whom to trust with MACE. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*.

Aaron Jaech, Larry Heck, and Mari Ostendorf. 2016. Domain adaptation of recurrent neural networks for natural language understanding. In *Proceedings of InterSpeech 2016*.

Jonathan K. Kummerfeld and Dan Klein. 2013. Error-driven analysis of challenges in coreference resolution. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Jonathan K. Kummerfeld, David Hall, James R. Curran, and Dan Klein. 2012. Parser showdown at the wall street corral: An empirical investigation of error types in parser output. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP)*.

Jonathan K. Kummerfeld, Daniel Tse, James R. Curran, and Dan Klein. 2013. An empirical examination of challenges in Chinese parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL)*.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*.

Stefan Larson, Anish Mahendran, Joseph J. Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K. Kummerfeld, Kevin Leach, Michael A. Laurenzano, Lingjia Tang, and Jason Mars. 2019. An evaluation dataset for intent classification and out-of-scope prediction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Stefan Larson, Eric Guldan, and Kevin Leach. 2020. Data query language and corpus tools for slot-filling and intent classification data. In *Proceedings of the 12th Language Resources and Evaluation Conference (LREC)*.

Hongwei Li and Qiang Liu. 2015. Cheaper and better: Selecting good workers for crowdsourcing. In *Third AAAI Conference on Human Computation and Crowdsourcing (HCOMP)*.

Qing Ma, Bao-Liang Lu, Masaki Murata, Michinori Ichikawa, and Hitoshi Isahara. 2001. On-line error detection of annotated corpus using modular neural networks. In *Proceedings of International Conference on Artificial Neural Networks (ICANN)*.

Yuji Matsumoto and Tatsuo Yamashita. 2000. Using machine learning methods to improve quality of tagged corpora and learning models. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC)*.

Tetsuji Nakagawa and Yuji Matsumoto. 2002. Detecting errors in corpora using support vector machines. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING)*.

Jingcheng Niu and Gerald Penn. 2019. Rationally reappraising ATIS-based dialogue systems. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Daniel M. Oppenheimer, Tom Meyvis, and Nicolas Davidenko. 2009. Instructional manipulation checks: Detecting satisficing to increase statistical power. *Journal of Experimental Social Psychology*, 45(4):867 – 872.

Natalie Parde and Rodney Nielsen. 2017. Finding patterns in noisy crowds: Regression-based annotation aggregation for crowdsourced data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Rebecca J. Passonneau and Bob Carpenter. 2014. The benefits of a model of annotation. *Transactions of the Association for Computational Linguistics*, 2:311–326.

Paul Roit, Ayal Klein, Daniela Stepanov, Jonathan Mamou, Julian Michael, Gabriel Stanovsky, Luke Zettlemoyer, and Ido Dagan. 2020. Controlled crowdsourcing for high-quality QA-SRL annotation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Hans van Halteren. 2000. The detection of inconsistency in manually tagged text. In *Proceedings of the COLING Workshop on Linguistically Interpreted Corpora*.

Mohammad-Ali Yaghoub-Zadeh-Fard, Boualem Benatallah, Moshe Chai Barukh, and Shayan Zamanirad. 2019. A study of incorrect paraphrases in crowdsourced user utterances. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*.