Noise-aware Missing Shipment Return Comment Classification in E-Commerce

Avijit Saha[*]
Flipkart Internet Private Limited
Bangalore, India
avijit.saha@flipkart.com

Vishal Kakkar[*]
Flipkart Internet Private Limited
Bangalore, India
vishal.kakkar@flipkart.com

T. Ravindra Babu
Flipkart Internet Private Limited
Bangalore, India
ravindra.bt@flipkart.com

ABSTRACT

E-Commerce companies face a number of challenges in return requests. Claims of *missing-items* is one such challenge, where customer claims that main product is missing from shipment through *return comments*. It is observed that dominant part of such claims are inadvertent given the limited literacy of customers. Some of them have fraud intent. At Flipkart, such claims are evaluated manually to examine whether the comment relates to missing item. Classification of the claim intent automatically saves human bandwidth and provides good customer experience by reducing the turn around time to customers. However, this is challenging as comments are replete with spell variations, non-English vernacular words, and are often incomplete and short. This is compounded by noisy labeling of such comments due to human bias and manual errors.

To classify the claim intent, we apply conventional as well as deep learning methods. To handle label noise, we employed state-of-the-art noise-aware techniques, which fail to perform due to pattern specific label noise. Motivated by the wide pattern specific label noise, we encode domain heuristics as labeling functions (LFs) which label subsets of the data. However, LFs may conflict and prone to noise. We address the conflict by defining a conflict-score to rank the LFs. Proposed method of noise handling with LFs out performs all the state-of-the-art noise-aware baselines.

KEYWORDS

E-Commerce, Text, Comment, Noise, Data Programming, Deep Learning, Machine Learning

ACM Reference Format:
Avijit Saha, Vishal Kakkar, and T. Ravindra Babu. 2018. Noise-aware Missing Shipment Return Comment Classification in E-Commerce. In *Proceedings of ACM SIGIR Workshop on eCommerce (SIGIR 2018 eCom).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

E-commerce companies face a large number of return requests of various types (reason-codes). *Missing-item* is one such reason-code where customer claims that main product is missing from shipment. Unlike a usual return, product pick up from customer is avoided in a missing-item return. Example of a missing-item return is - customer ordered a handset and received a stone in place of the handset. A confirmed case of missing item results in loss to the company since there is a definite fraud with one of the stakeholders such as buyer, seller or delivery team. Hence, a careful scrutiny is necessary before the approval of any missing-item returns.

A missing-item return request is generated broadly for two reasons. The first can be due to definite fraud with one of the stakeholders. Secondly, given limited literacy levels of customers, it is observed that the claims do not always refer to missing-item but inadvertently claimed as missing-item. For example, a missing-item return with customer comment 'I did not like the item.' clearly indicates that the return belong to a return category other than missing-item. Because here customer received the main product. Hence, the return should be cancelled. We call this a comment-mismatch (customer's comment does not match with the return reason-code). On the other hand, a missing-item return with customer comment 'I ordered a phone but received an empty box.' clearly indicates that the return belong to the missing-item category. This return should be approved. This is referred by non-comment-mismatch (customer's comment matches with the return reason-code).

At Flipkart, a dedicated operation team assesses the compatibility of customers' comments on missing-item return with the missing-item reason-code. Each missing-item return – passes through this process and – is rejected when the return comment is incompatible with the missing-item reason-code, otherwise approved. This process wastes lot of human bandwidth and is prone manual mistakes. Moreover, it hampers the customer experience due to the lag between the return placement time and its status update to the customer. Hence, we want to automate this process. In terms of Machine Learning problem, given a missing-item return comment, we want to predict whether it is a comment-mismatch (positive class) or non-comment-mismatch (negative class).

Customer comments are generally very noisy mainly because of three reasons: a) spelling mistakes: empty is misspelled in comment 'Emety box', b) usage of regional languages: comment 'Galt order ho gya h', which means I ordered wrong product, uses Hindi language, and c) varied comment length: token counts in a comment ranges $[1, 323]$. To handle such difficulties, we employ word embedding and meta features. Besides conventional classification method (xgboost [6]), we use BLSTM [24] to capture the sequential information in the comments.

Also, the manual label generation process, which marks a missing-item return comment as comment-mismatch/non-comment-mismatch, is very noisy. The sources of noise are manual error, human bias, and lack of well calibrated operation team. The label noise varies

[*]Equal Contributions

depending on the patterns and is non-iid. This is clearly visible by the fact that the overall label noise is ∼ 15% and a specific pattern 'I ordered x quantity of an item but received y quantity' has 50% noise. Due to this reason, the state-of-the-art noise-aware [11] baseline, which uses BLSTM as the base model, fails severely. In fact, we observed a performance degradation of this noise-aware BLSTM over vanilla BLSTM.

The pattern specific noise variation and high noise on certain patterns motivate us to use noise correction based on domain heuristics. Like data programming[20], we express weak supervision strategies or domain heuristics as labeling functions (LFs) which label subsets of the data. However, LFs may conflict and prone to noise. To our best knowledge, no one has employed LFs to rectify label noise.

In a typical data programming setting, only data points are available and LFs are created to generate labels. The LFs may conflict on certain data points and have varying error rates. To handle it, data programming defines a generative process over the LFs to learn the correctness probability of each LF on each data instance. This information is then used to fit a noise-aware classifier.

The key difference between our setting and the data programming is the availability of noisy labels (we call it as true LF) in our case. Unlike data programming, we would like introduce less noisy LFs compared to the true LF. Due to all these reasons, instead of applying data programming directly, here resort to a simple method. and promising methods to correct label noise using LFs, and leave out the exploration of data programming as future work.

We define multiple LFs to alter the noisy labels in our dataset. However, applying them directly to flip the noisy labels is impossible because the LFs conflict with each other - same data instance is labeled as positive by a LF and negative by another LF. This requires us to generate a ranked list of LFs. To do that, we define a conflict score, which captures how less a LF conflicts with other LFs. Then, the ranked LFs are applied to alter the true labels. In case of conflict between LFs, the LF with the least conflict score is chosen to alter the noisy label. We show that proposed method of noise handling with LFs out performs all the state-of-the-art noise-aware baselines as well as vanilla baselines.

We integrate the following aspects in the paper.

- Explanation of a real world problem and its challenges
- Exploratory data analysis and feature engineering
- State-of-the-art baselines - xgboost and LSTM and their noise-aware variants
- Noise handling with labeling functions
- Proposed conflict-score to handle conflicts
- Shown superior performance of the proposed method

Section 2 contains insights into data. Related work, feature engineering, modeling, experimentation, and conclusion are described in Section 3, 4, 5, 6, and 7, respectively.

2 DATA

2.1 Description

The data in our study comes from the customer comments on missing-item return requests. We use one year of customer comments: April 2017 - Mar 2018. We consider a comment-mismatch as positive label and a not-comment-mismatch as negative label. Table 1 describes the dataset statistics. We can observe ∼ 41% are positive labels. We count the number of words in a comment by tokenizing it. Table 1 also shows that the word count of comments ranges [1, 323]. The plot clearly shows the existence of widely variable length comments in our dataset. To deep dive, we plot a histogram of number of comments w.r.t. word count in Figure 1. We clip the plot at word length 75 for better visualization. Clearly, it is a long tail distribution - approximately 51% of comments has less than ten words. However, there exists a fat tail of comments with very high word count.

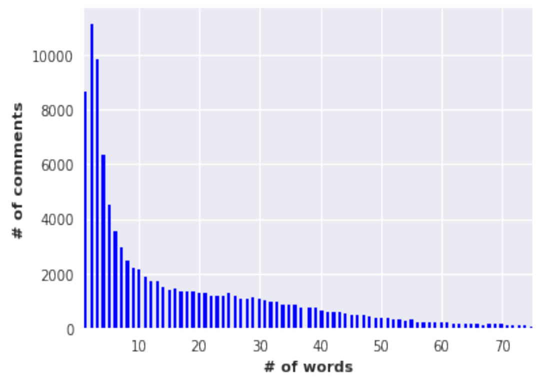

Figure 1: Frequency distribution of comments w.r.t. word count

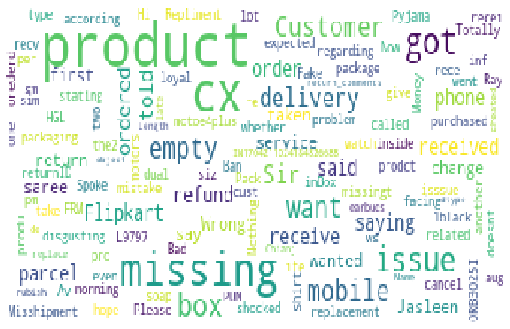

Figure 2: Word clouds

Table 2 shows example of customer comments with different word count. Interestingly, we can observe the presence of spell error - 'My mistek' and regional language - 'Sir khali box mila h' (Hindi). Often, comments are very noisy and do not adhere to grammar rules - 'Not working Good; tow time same product bye mistack accepted so remove it'. To provide more insight on the data, we show a word cloud of our dataset in Figure 2. Some of the important keywords are cx (customer), product, mobile, box, missing, and empty. The cx words occurs in the comments when customer calls customer care to place the return request.

Table 1: Data Statistics

# of instances	% of negatives	% of positives	min word count in any comments	max word count in any comments
O(100k)	59	41	1	323

Table 2: Example of comments with different word count

Word Count	Comment
1	Missing
	Sorry
2	Product missing.
	My mistek
4	I got one jacket
	empty box no watch
8	Charged more than 172/- from the MRP on box
	Sir mera phone nahi tha box ke andar

Table 3: Example of noisy labels

Word count	Comment	Actual Label	True Label
1	Ghh	-1	1
	Missing	1	-1
2	Wrong delivery	-1	1
	Missing product	1	-1
4	Customer wants the refund	-1	1
	Product is not there	1	-1
8	Please replacement this product otherwise return my money	-1	1
	Item missing your department , Very poor bad service	1	-1

2.2 Label Noise

Recall, non-comment-mismatch implies a genuine return request of type missing-item and comment-mismatch implies a return request anything other than missing-item type. Given a missing item return comment, our operation team at Flipkart mark it as either comment-mismatch (positive) or non-comment-mismatch (negative). This label generation process is very noisy due to manual error, human bias, and lack of well calibrated operation team. Table 3 shows example of noisy labels for comments with different word count. Actual label and true labels represents the label in the dataset and the expected label, respectively. We show noisy labels from one, two, four, and eight word count comments. For each word count, we show one comment whose label should be positive but marked as negative and one comment whose label should be negative but marked as positive.

2.3 Noise Statistics

To estimate the overall noise in our dataset, we manually relabeled $3k$ randomly chosen examples and consider it as test dataset. We calculate the noise by considering mismatch between the actual label and true label on the test dataset. The overall noise is estimated to be 15.65%. Also, the positive and negative classes have 10.06% and 19.45% noise, respectively.

3 RELATED WORK

Preprocessing is an important step for text classification. Two important blocks [1] of preprocessing are - 1) Tokenization and 2) Filtering. Tokenization [23], which is the initial step of preprocessing, divides a text document into words known as tokens. In Filtering, stop words are removed.

Then, the preprocessed text is converted into feature vectors. One of the widely used model for feature generation is the bag-of-words (BOW) model [18]. It represents a document to a k-dimensional feature vector, where the individual co-ordinate represents the count of a specific word in the document. Often, term frequency-inverse document frequency (tfidf) [18] is used to penalize a frequently occurring word. Recently, word2vec (w2v) [19] model gained much attentions. It embeds a word to a k-dimensional vector space by preserving the property that words occurring in the same context will have higher similarity score. Word2vec [16] features are used for text classification in many ways - summation of the w2v embedding vectors, mean of the w2v embedding vectors, and tf-idf based weighted sum of the w2v embedding vectors corresponding to all the words in a document.

Support vector machines (SVMs) [14] are widely employed for text classification. Athanasiou [2] has applied gradient boosting machine for sentiment analysis task, and shown superior performance over SVM, Naive Bayes, and neural network. Gradient boosting machine is a boosting algorithm where each iteration fits a new model to get better class estimation. Each newly added model is correlated with the negative gradient of the loss function, and the loss is minimized using gradient descent. Extreme gradient boosting (Xgboost) [6] is another boosting algorithm with better regularization and performs well in practice.

Recently deep learning algorithms[15, 22] have shown promising performance in text classification. Specifically, recurrent neural networks (RNNs) [15] are the widely used architectures to capture the sequential information. Long term short memory networks (LSTMs) [12] is a variant of RNN which helps to overcome some of the problem of RNN like, vanishing gradient problem and helps to remember the context over long text. Many flavours of LSTMs are proposed [22] to for text classification, such as Multilayer-LSTM, Bidirectional-LSTM (BLSTM), and Tree-Structured LSTM. In Multilayer LSTM, LSTMs are stacked over each other to capture the non-linearity. In BLSTM, both past and future information are preserved using two hidden states, and it helps to learn the context better.

Noisy label can be handled broadly by three approaches [9]: a) label-noise robust models [4, 6], b) data cleansing methods [5, 13], and c) label-noise tolerant learning algorithms [3, 11]. In label-noise robust methods, label-noise is handled by reducing the overfitting.

Even though, theoretically learning algorithms are robust to label-noise, in practice performance varies from algorithm to algorithm, such as bagging perform better than the boosting [8]. Data cleansing methods filter out data points which appears to be mislabeled. Filtering can be done with various approaches, such as outlier detection [13], removal of all the misclassified data points by a classifier [5], and removal of any points which disproportionately increases the model complexity [10]. In label-noise tolerant algorithm, noise is handled explicitly in the modeling step. Label-noise robust logistic regression [3] modifies the loss function to handle noise. Recently, a probabilistic neural-network based framework [11] is developed, which views the true label as a latent variable and a softmax layer is used to predict it. The noise is explicitly modeled by an additional softmax layer that predict the noisy label based on both the true label and the input features.

As creating labeled training data is difficult and time consuming, many approaches are developed to generate training data automatically, such as distant supervision [7, 17] and data programming [20]. Distant supervision heuristically maps a knowledge base of known relations to an unknown domain to generate training data. Data programming is a generic framework to create dataset pragmatically using distant supervision. It expresses weak supervision strategies or domain heuristics as labeling functions (LFs) which label subsets of the data. The LFs may conflict on certain data points and have varying error rates. To handle it, data programming defines a generative process over the LFs to learn the correctness probability of each LF on each data instance. This information is then used to fit a noise-aware classifier.

4 FEATURE ENGINEERING

In this Section, we will discuss all the hand crafted features which are used for conventional Machine Learning models.

4.1 Meta Features

We construct nine meta features as shown in Figure 3. Word count, char count, alpha count, digit count, and non-alphanumeric count compute the number of words, characters, alphabets, digits, and non-alphanumeric characters in a comment. To show the discriminating power of each meta feature, in Figure 3, we show histogram of each feature w.r.t. both positive and negative class. In each subplot, the blue and green histogram represents the distribution for negative and positive class, respectively.

In each subplot, the blue distribution is right shifted. It implies that in general comments from the negative class are longer, have more alphabets, digits and non-alphanumeric characters compare to the comments from the positive class. This is explained by the fact that short comments lack descriptive ability, and thus have higher chance of being comment-mismatch. Interestingly, unique character count and unique alphabet count are the most discriminating features - there is a clear separation between the distribution of positive and negative class for these two features.

4.2 Word2vec Features

We observed that our dataset is very noisy. For example, a keyword like 'missing' has numerous variations - missing, misssing, misisng, missig, etc. Moreover, there are semantically similar words, such as 'empty' and 'khali' both means empty. Also, customers use regional language, such as hindi ('mujhe product nahi mila'). To handle these complexities, we train a word2vec model with 200 dimension on one year customer's return comments data from all the return reason-codes. The number of comments are $O(10M)$. We train the word2vec by considering words which occurs at-least 25 times in our corpus. For tokenization, we use Gensim [21] simple_preprocess method.

Table 4: Example of Spell Variation and Synonym

missing	**mobile**	**product**	**ordered**
mising	mobil	prodcut	orderd
misssing	fone	produict	odered
misisng	device	produc	order
khali	handset	prioduct	booked
empty	phone	item	purchased

Table 4 shows similar words from the word2vec model for four keywords - missing, mobile, product, and ordered. We can observe that the word2vec model is able to capture spelling variations. It is also able to capture semantically similar word, such as missing and empty, mobile and handset, device and phone, and ordered and purchased. Moreover, the regional language variation is also captured, such as missing and khali (hindi), mobile and fone (hindi), and missing and illa (tamil).

4.2.1 *Sum of Word2vec Features:*

We sum the individual 200-dimensional embedding vector for each word in a comment and use it as the final feature in model. To handle out-of-vocabulary word, we fall back to the 200-dimensional zero vector.

4.2.2 *Weighted Word2vec Features:*

We fit a tfidf model on the training data. Then, we calculate a weighted sum of the individual 200-dimensional embedding vector for each word in a comment. Where weight of individual word embedding is assigned from the tfidf score of the word.

4.3 Bag-of-words (BOW) Features

Each comment in the training dataset is preprocessed with Gensim preprocessing. Then, a bag-of-words (BOW) model is trained with 5k vocabulary size and with English stop words removal.

5 MODEL

5.1 BASELINE

We tried multiple conventional Machine Learning algorithms widely used for text classification, such as SVM, naive Bayes, logistic regression, random forest, and xgboost. In our data, around 50% of the comments are long and we found that often long comments have sequential information, such as 'customer ordered 2 items but received only 1'. To capture the sequential information, we experimented with different RNN based models, such as RNN, LSTM, Multi-layer LSTM, and Bi-directional LSTM (BLSTM). In below, we only describe the best performing models from each of this approach.

5.1.1 *Xgboost:*

A xgboost model is trained with all the features described in Section 4. The model parameter is tuned by grid search.

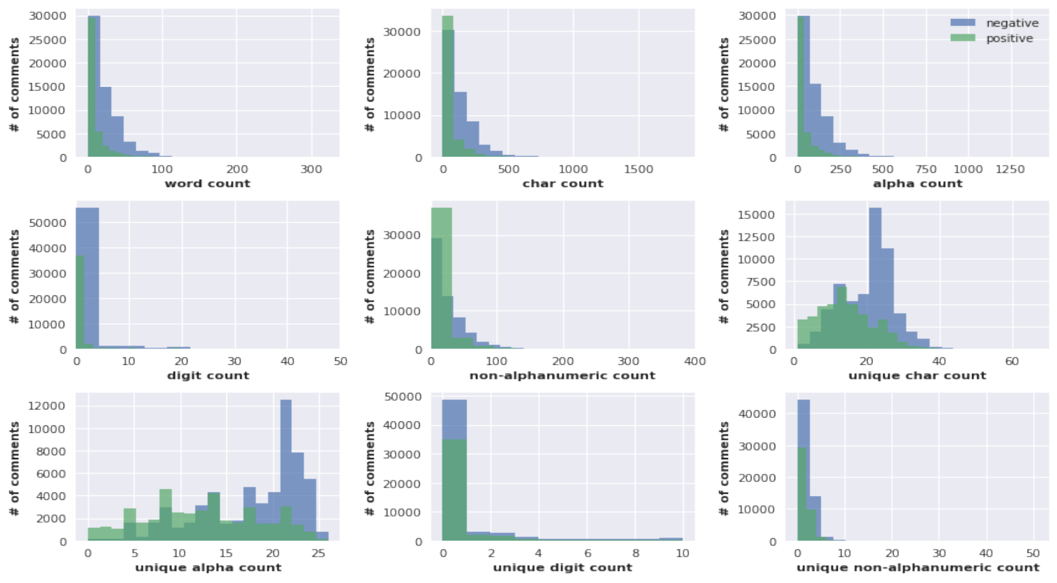

Figure 3: Meta features

While performing the grid search, we restrict the max depth of individual tree to 8 and max number of estimators to 500. The best parameters are chosen using 5-fold cross-validation.

5.1.2 BLSTM:. We used BLSTM which avoids feature engineering. In BLSTM, both past and future information are preserved using two hidden states, and it helps to learn the context better. We experimented with word2vec pretrained embedding from the word2vec model as well as learning the embedding from scratch in the network itself. We tune the number of neurons in the BLSTM. We also experimented by adding fully connected relu layes in the network before the output layer. The best parameters are tuned based on a validation set.

5.2 NOISE-AWARE BASELINE

We tried two approaches to handle noisy labels: 1) data cleansing method and 2) label noise-tolerant algorithm.

5.2.1 Data Cleansing Method: The goal of such methods is to filter out data points which appears to be mislabeled. Here, we apply a model prediction based filtering method [5].

In this filtering approach, we divide the training dataset into k-folds (five-folds). We train a xgboost model (with the best parameters found in 5.1.1) on k-1 folds and apply it to predict labels for k-th fold. This process is repeated k times to get labels for the entire training dataset. Then we filter out instances with disagreement between the actual label and the predicted label. On the filtered training data, a xgboost model (with same parameter) is trained which forms the final model. We refer this method as xgboost+filtering.

5.2.2 Label Noise-Tolerant Algorithm: In label-noise tolerant algorithm, noise is handled explicitly in the modeling step. Here, a probabilistic neural-network based framework [11] is considered to handle label-noise. This framework views the true label as a latent variable. A softmax layer is used to predict the true label. The noise is explicitly modeled by an additional softmax layer that predict the noisy label based on both the true label and the input features.

Assuming the non-linear function applied on an input x be $h = h(x)$, the true label y is modeled by:

$$p(y = i|x; w) = \frac{exp(u_i^T h + b_i)}{\sum_{l=1}^k exp(u_l^T h + b_l)}, \quad i = 1, ..., k \tag{1}$$

Where k is the number classes and w is the network parameter-set (including the softmax layer). Next a softmax output layer is added to predict the noisy label z based on both the true label and the input features:

$$p(z = j|y = i, x; w_{\text{noise}}) = \frac{exp(u_{il}^T h + b_{il})}{\sum_{l=1}^k exp(u_{il}^T h + b_{il})} \tag{2}$$

$$p(z = j|x) = p(z = j|y = i, x)p(y = i|x) \tag{3}$$

Where, w_{noise} represents parameters in the second softmax layer. Given n training data points with feature vectors $x_1, x_2, ..., x_n$ with corresponding labels $z_1, z_2, ..., z_n$ and true labels $y_1, y_2, ..., y_n$, the log likelihood term of the model parameters is written as:

$$\sum_t \log p(z_t|x_t), \; t = 1, ..., n \tag{4}$$

This modeling approach is named as c-model and learned using a neural-network training. For our experiment, the h function is considered to be a BLSTM, with the same parameter as in Section 5.1.2. This model is referred as BLSTM+noise-aware.

Table 5: Example of Labeling Functions (LFs)

def $\text{lambda}_{\text{partial}}(x)$:
return 1 **if** # of positive integer tokens in x >= 2 **else** 0

def $\text{lambda}_{\text{soap}}(x)$:
return -1 **if** IsTokenPresent(soap, x) **else** 0

5.3 NOISE HANDLING WITH WEAK SUPERVISION

We encode domain heuristics as labeling functions (LFs) [20], which label subsets of the data. However, LFs may conflict and prone to noise. Assuming data point and class pair (x, y) are drawn from the distribution $X \times \{-1, 1\}$, a LF $\lambda_i : X \longrightarrow \{1, 0, -1\}$ is a user-defined function that encodes some domain heuristic, and provides non-zero label for some subset of the data points. Where 1 and -1 refer to the positive and negative class respectively. And 0 refers to the case where LF can not label the instance. LFs collectively generate a large but potentially overlapping set of training labels. LFs can be created in many ways, such as leveraging domain specific patterns to label data points or use existing knowledge bases to generate labels.

We consider that the actual labels came from a LF named λ_{true}, which has 100% coverage. As introduction of a more noisy LF than λ_{true} will increase overall noise in the dataset, unlike data programming, we introduce less noisy LFs than λ_{true}. Lets consider λ denotes the m newly created LFs - $\{\lambda_i\}_{i=1}^{m}$, each of which looks at the domain specific patterns to label data points.

A specific pattern is observed in the comments - 'I ordered x quantity of an item but received y quantity'. This is partial delivery, as the customer has received part of the order, and should be marked as comment-mismatch. With this pattern, a LF $\text{lambda}_{partial}$ is defined in Table 5. We also observed that customer often writes they have received soap instead of a mobile phone. This is a genuine missing-item request, and should be marked as non-comment-mismatch. With this pattern, a LF lambda_{soap} is also defined in Table 5. The IsTokenPresent function returns true when soap is one of the token of comment x. Table 6 describes the statistics of all the LFs. Where coverage represents the percentage of instances with at least one label, conflict represents the percentage of instances with conflicting labels, and overlap depicts the percentage of instances with more than one labels.

To deep dive, we show conflict and overlap count in Table 7 and Table 8, respectively. Where a cell (i, j) in Table 7 defines the number data instances where ($\lambda_i == 1$ and $\lambda_j == -1$) or ($\lambda_i == -1$ and $\lambda_j == 1$). Similarly, a cell (i, j) in Table 8 defines the number data instances where ($\lambda_i == 1$ and $\lambda_j == 1$) or ($\lambda_i == -1$ and $\lambda_j == -1$).

As LFs conflicts with each other applying them directly to flip the true labels is impossible. This requires us to generate a ranked list of LFs. To do that, for each $\lambda_i \in \lambda$, we define a conflict-score as below. Note, conflict count of λ_i is calculated by summing the number of conflict between λ_i and each rule from $\lambda - \lambda_i$.

Table 6: Statistics of Labeling Functions

# of LFs	Coverage (%)	Overlap (%)	Conflict (%)
17	100	40	9

Table 7: Conflict Count

	true	partial	phone missing	soap
true	0	5,752	368	44
partial	5,752	0	247	54
phone missing	368	247	0	0
soap	44	54	0	0

Table 8: Overlap Count

	true	partial	phone missing	soap
true	O(100k)	3,619	1,819	621
partial	3,619	9,371	0	0
phone missing	1,819	0	2,187	42
soap	621	0	42	665

$$\textbf{cf\_score}_{\lambda_i} = \frac{\text{\# unique conflicts of } \lambda_i \text{ with other LFs}}{\text{Coverage of } \lambda_i} \tag{5}$$

Intuitively, the cf_score captures how much a LF conflicts with other LFs. With this score, Algorithm 1 denoise the training data. λ^{sorted} contains the sorted list of m newly created rules in ascending order w.r.t the $\text{cf\_score}_{\lambda_i}$.

Algorithm 1 Label Denoising with Labeling Functions (LFs)

```
Input: $X$, $\lambda_{true}$, $\lambda^{sorted}$
Output: $Y$: final label vector
append $\lambda_{true}$ at the end of $\lambda^{sorted}$
for $x_i \in X$ do
    flag=0
    for $\lambda_j \in \lambda^{sorted}$ do
        $y_i = \lambda_j(x_i)$
        if $y_i \neq 0$ then
            flag=1
            break
        end if
    end for
    if flag==0 then
        $y_i = \lambda_{true}(x_i)$
    end if
end for
```

After denoising the training data with Algorithm 1, we apply the xgboost and BLSTM model on the denoised training data. This two approaches are named as xgboost+best-sequence and BLSTM+best-sequence.

Table 9: Train and Test Dataset Statistics

# Type	# of instances	% of neg	% of pos	Label type
Train	O(100k)	59	41	Noisy
Test	3,000	51.5	48.5	Clean

6 EXPERIMENTS

6.1 Dataset

The complete data consists of $O(100k)$ instances out of which randomly chosen $3k$ instances forms the test data and rest forms the train data. The test dataset is manually relabeled to generate clean labels. Table 9 shows train and test data statistics. Note, the test dataset size is small because manual relabeling is time consuming.

6.2 Experimental Setup

We compare performance of xgboost+best-sequence and BLSTM+best-sequence against the baselines - xgboost, BLSTM, xgboost+filtering, and BLSTM+noise-aware. Moreover, to showcase the benefits of the conflict handling with the cf_score, we compare the performance of xgboost+best-sequence and BLSTM+best-sequence with xgboost+random-sequence and BLSTM+random-sequence. For random-sequence, λ^{sorted} consists of a of random permutation of m newly created rules. Model performance varies for different random sequences. Hence, for both xgboost+random-sequence and BLSTM+best-random, we repeat experiments 10 times with different random permutation of λ and report the mean and standard deviation. For all the models, we fix random-state to 2018. As our dataset is well balanced, we use accuracy as the evaluation metric.

6.3 Results

Table 10 shows the performance comparison between xgboost, BLSTM, and their noise-aware variants. BLSTM is performing best with an accuracy of 87.43. This proves that using sequence information indeed benefits on our dataset. BLSTM and xgboost are performing better than BLSTM+noise-aware and xgboost+filtering, respectively. We can observe that the state-of-the-art noise-aware algorithms are hurting the performance. We think that the reason for such a performance degradation is due to the wide pattern specific noise variation.

Table 10: Performance of Baselines

Model	Accuracy
Xgboost	**86.90**
Xgboost+filtering	86.47
BLSTM	**87.43**
BLSTM+noise-aware	87.33

Table 11 shows the performance comparison among our methods. Again, BLSTM with best-sequence is performing best. BLSTM+best-sequence and xgboost+best-sequence are performing better than BLSTM+random-sequence and xgboost+random-sequence, respectively. It proves the benefits of conflict handling among LFs by cf_score. Note, the standard deviation of BLSTM+random-sequence is quite low, and still BLSTM+best-sequence beats the BLSTM+random-sequence, again proving conflict handling with cf_score helps.

Overall, BLSTM+best-sequence performs best, proving the efficacy of our proposed approach. Best-sequence provide benefits over random-sequence proving the benefits of conflict handling among LFs by cf_score. Sequence model BLSTM is able to provide benefits over xgboost. We were able to improve the accuracy from 86.90% to 90.04% with the help of BLSTM, LFs, and cf_score.

Table 11: Performance of Proposed Methods

Model	Mean Accuracy	Std
Xgboost+random-sequence	87.64	1.37
Xgboost+best-sequence	**88.39**	NA
BLSTM+random-sequence	88.37	0.02
BLSTM+best-sequence	**90.04**	NA

7 CONCLUSION

We discussed an important problem of classifying missing-item return comments into comment-mismatch/non-comment-mismatch. We highlighted the data and noise related challenges in both comments and labels. We have experimented with the state-of-the-art Machine Learning and Deep Learning methods as well as their noise-aware variants. We have proposed a simple method with labeling functions (LFs) to denoise the training dataset. A conflict-score is defined to handle the conflicts between LFs. Empirically, we have shown efficacy of our approach over the baselines.

As future work, we intend to explore the complete data programming framework to handle noisy labels.

REFERENCES

[1] Mehdi Allahyari, Seyed Amin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, and Krys Kochut. 2017. A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques. *CoRR* abs/1707.02919 (2017). arXiv:1707.02919 http://arxiv.org/abs/1707.02919

[2] Vasileios Athanasiou and Manolis Maragoudakis. 2017. A Novel, Gradient Boosting Framework for Sentiment Analysis in Languages where NLP Resources Are Not Plentiful: A Case Study for Modern Greek. *Algorithms* 10 (2017), 34.

[3] Jakramate Bootkrajang and Ata Kabán. 2012. Label-Noise Robust Logistic Regression and Its Applications. In *Proceedings of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I (ECML PKDD'12)*. Springer-Verlag, Berlin, Heidelberg, 143–158. https://doi.org/10.1007/978-3-642-33460-3_15

[4] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (Oct. 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[5] Carla E. Brodley and Mark A. Friedl. 1999. Identifying Mislabeled Training Data. *J. Artif. Int. Res.* 11, 1 (July 1999), 131–167. http://dl.acm.org/citation.cfm?id=3013545.3013548

[6] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[7] M. Craven and J. Kumlien. 1999. Constructing biological knowledge bases by extracting information from text sources. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology.*

[8] Thomas G. Dietterich. 2000. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning* 40, 2 (01 Aug 2000), 139–157. https://doi.org/10.1023/A:1007607513941

[9] Benoît Frénay and Ata Kaban. 2014. *A Comprehensive Introduction to Label Noise.* i6doc.com.publ.

[10] Dragan Gamberger, Rudjer Boskovic, Nada Lavrac, and Ciril Groselj. 1999. Experiments With Noise Filtering in a Medical Domain. In *Proc. of 16 th ICML.* Morgan Kaufmann, 143–151.

[11] Jacob Goldberger and Ehud Ben-Reuven. 2017. Training Deep Neural-networks Using a Noise Adaptation Layer.

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[13] Victoria J. Hodge and Jim Austin. 2004. A survey of outlier detection methodologies. *Artificial Intelligence Review* 22 (2004), 2004.

[14] Thorsten Joachims. 1998. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *Proceedings of the 10th European Conference on Machine Learning (ECML'98)*. Springer-Verlag, Berlin, Heidelberg, 137–142. https://doi.org/10.1007/BFb0026683

[15] Ji Young Lee and Franck Dernoncourt. 2016. Sequential Short-Text Classification with Recurrent and Convolutional Neural Networks. *CoRR* abs/1603.03827 (2016).

[16] Joseph Lilleberg, Yun Zhu, and Yanqing Zhang. 2015. Support vector machines and Word2vec for text classification with semantic features.. In *ICCI*CC*, Ning Ge, Jianhua Lu, Yingxu Wang, Newton Howard, Philip Chen, Xiaoming Tao, Bo Zhang, and Lotfi A. Zadeh (Eds.). IEEE Computer Society, 136–140. http://dblp.uni-trier.de/db/conf/IEEEicci/IEEEicci2015.html#LillebergZZ15

[17] Emily K. Mallory, Ce Zhang, Christopher Rï£¡, and Russ B. Altman. 2016. Large-scale extraction of gene interactions from full-text literature using DeepDive. *Bioinformatics* 32, 1 (2016), 106–113. https://doi.org/10.1093/bioinformatics/btv476

[18] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013).

[20] Alexander J. Ratner, Christopher De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data Programming: Creating Large Training Sets, Quickly. In *NIPS*. 3567–3575.

[21] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, 45–50. http://is.muni.cz/publication/884893/en.

[22] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. *CoRR* abs/1503.00075 (2015).

[23] Jonathan J. Webster and Chunyu Kit. 1992. Tokenization As the Initial Phase in NLP. In *Proceedings of the 14th Conference on Computational Linguistics - Volume 4 (COLING '92)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 1106–1110. https://doi.org/10.3115/992424.992434

[24] Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. 2016. Text Classification Improved by Integrating Bidirectional LSTM with Two-dimensional Max Pooling. *CoRR* abs/1611.06639 (2016).