Sensitive Data Detection with High-Throughput Neural Network Models for Financial Institutions

Anh Truong*, **Austin Walters***, **Jeremy Goodsitt***
Capital One
Applied Research, Center for Machine Learning
Anh.Truong@capitalone.com, Austin.Walters@capitalone.com, Jeremy.Goodsitt@capitalone.com

Abstract

Named Entity Recognition has been extensively investigated in many fields. However, the application of sensitive entity detection for production systems in financial institutions has not been well explored due to the lack of publicly available, labeled datasets. In this paper, we use internal and synthetic datasets to evaluate various methods of detecting NPI (Non-public Personally Identifiable) information commonly found within financial institutions, in both unstructured and structured data formats. Character-level neural network models including CNN, LSTM, BiLSTM-CRF, and CNN-CRF are investigated on two prediction tasks: (i) entity detection on multiple data formats, and (ii) column-wise entity prediction on tabular datasets. We compare these models with other standard approaches on both real and synthetic data, with respect to F1-score, precision, recall, and throughput. The real datasets include internal structured data and public email data with manually tagged labels. Our experimental results show that the CNN model is simple yet effective with respect to accuracy and throughput and thus, is the most suitable candidate model to be deployed in the production environment(s). Finally, we provide several lessons learned on data limitations, data labelling and the intrinsic overlap of data entities.

Introduction

Named Entity Recognition (NER) is a subset of Natural Language Processing (NLP) used for identifying predefined entities in text. NER is used on both domain-specific text such as social network (Ritter et al. 2011) and biomedical text extraction (Cho and Lee 2019; Fritzler, Logacheva, and Kretov 2019), as well as more general corpora (Gui et al. 2019; Kuru, Can, and Yuret 2016; Kurniawan and Louvan 2018). With a few exceptions (Francis, Van Landeghem, and Moens 2019), the use of NER in finance has not been extensively studied as there do not exist publicly available, labeled datasets that contain the sensitive information. Within financial institutions, those sensitive information needs to be protected before data are uploaded on emails, githubs, or data repositories. In addition, there are often a large amount of unstructured and structured data with different formats that are stored across many internal storage systems. In this paper, we implement an NER system that overcomes the aforementioned obstacles. Our system generates data with sensitive information and consists of neural network models with both reasonably high accuracy and data throughput. These models have been optimized for architectures and data preprocessing over multiple computation resources. Our NER system concentrates on solving two problems: (i) predicting the presence of sensitive entities on different data formats, and (ii) predicting column-wise entities on tabular datasets in which each column contains only one type of entity. Below, we list our contributions in detail.

Data generation with sensitive information. Due to the lack of public datasets with sensitive information, a collection of common sensitive entities is generated, from which we generate multiple datasets containing these entities under different formats. The generated datasets consist of unstructured text with sensitive entities dispersed throughout, and structured single-column and multi-column data where each column contains a sensitive entity of the same type. Each structured data is represented in different file types, csv, json, and parquet. For evaluation purposes, we incorporate internal data and real email data extracted from spam email corpora.

Neural network model optimization for production environments. We explored various experiments on multiple neural network models including CNN (Convolutional Neural Network), LSTM (Long Short-Term Memory), CNN-LSTM, and CRF (Conditional Random Field) based models, CNN-CRF, BiLSTM-CRF, where CRF layer is utilized as a tag decoder. We also evaluated the popular NER library SpaCy with its default NER configurations and retrained on our training datasets. Finally, for a baseline comparison, a regex model including a set of hand curated regex filters designed specifically for the entities in our dataset, as well as a CRF model with handcrafted features are used.

We observe that our CNN model outperformed the other models with respect to both accuracy and throughput, and thus is the most suitable model for production usage. This model is also universal: it easily adapts from the entity detection task to column-wise entity prediction on tabular datasets with some slight modifications on input data preprocessing, and it is comparable to a specifically designed model for this task, Column-CNN-CRF.

*These authors have made equal contributions to this article.

Related Work

Sensitive information detection in structured text has been explored in different fields such as military and politics (Xu et al. 2019), and social network (Caliskan Islam, Walsh, and Greenstadt 2014). However, their work focused on sensitive document classification problems as opposed to our NER problem which predicts sensitive entities in text, under the financial domain. Approaches for NER include statistical modeling methods (Eddy 1996; Kapur 1989; Lafferty, McCallum, and Pereira 2001) and recently neural network models (Yang, Zhang, and Dong 2017; Peters et al. 2018; Shen et al. 2017). In this paper, we seek for effective NER models which obtain not only high accuracy but also high throughput. To that end, we first optimize two models in this framework, CNN and LSTM (Huang, Xu, and Yu 2015), evaluating different architectures to extract richer features LSTM-CNN (Chiu and Nichols 2016). We then examine the effect of the CRF layer as a tag decoder to optimize the label sequence prediction. This has been observed to be an effective addition by (Collobert et al. 2011) with CNN-CRF, and (Huang, Xu, and Yu 2015; Francis, Van Landeghem, and Moens 2019) with BiLSTM-CRF. It is worth noting that the aforementioned models are based on word embeddings, or the combination of word embeddings and character embeddings. In this work, we focus on character-level models as opposed to word-level models as the considered sensitive entities, specific to the financial domain, cannot be represented by the existing tokenizers trained on the general corpora (Boukkouri et al. 2020). Moreover, the sensitive entities we are trying to identify are unique and non-public (NPI), meaning that word-level representation of these entities potentially results in being out-of-vocabulary. That being said, later made model comparisons do include a word-level NER model from the standard NLP toolkit, spaCy which are congruent with (Kuru, Can, and Yuret 2016), showing its disadvantage with such types of data. Beside spaCy, there exists other well-known NER tools such as NLTK and recently Presidio from Microsoft, a tool for PII detection on text and image. However, spaCy was selected for comparison as it is ubiquitous and has been utilized in many studies (Dutt et al. 2018; Gelernter and Zhang 2013; Malmasi S. 2016).

Sensitive Data Generation

The datasets used for model training and evaluation are created via synthetic generation and labeling since a public dataset for the desired entities did not exist. These datasets contain both sensitive and non-sensitive entities which would commonly be found within a financial institution's database.

Sensitive Entity Generation

Below is the list of rules associated with the 19 entities considered in this paper (the detailed examples of these entities are given in (Truong, Walters, and Goodsitt 2020)):

Sensitive entities:
Address (ADDRESS): US address which may be multi-line and can contain newline characters. "City/City, State" are not included in this set.
Bank Account Number (BAN): Numbers associated with an account at a bank. While technically these could be alphanumeric 1-18 in length, we limited to 10-18 digits as those are the far more common.
Credit Card Number (CREDIT_CARD): Amex, Visa, Mastercard, and Discover credit card numbers with the optional delimiters (",", " ", ".", "_").
Datetime (DATETIME): Datetime formats recognizable by the python datetime library.
Email Address (EMAIL_ADDRESS): An email address or email portion of the URI not contained within a url.
Hash or Encryption Key (HASH_OR_KEY): Randomized concatenation of 16 or more alphanumeric characters and special characters "=", "-", "\", "/", "+" for md5, SHA1, SHA256, SHA512, or any encryption key.
IPv4 (IPV4): Standard IPv4 formats.
IPv6 (IPV6): Standard IPv6 formats.
MAC Address (MAC_ADDRESS): Standard MAC Address formats with delimiters ("-", ":", " ", ".").
Persons Name (PERSON): Name of a person including titles, but not including possessive nouns.
Phone Number (PHONE_NUMBER): US/-Canada/UK/Ireland phone numbers with or without international code, zip code, or extensions.
Social Security Number (SSN): US Social Security Numbers with varying delimiters ("-", ".", "", " ").
URL (URL): Any url with a network scheme, www initialized, or valid url if the aforementioned were prepended.
UUID (UUID): UUID4 format.

Non-Sensitive Entities:
Background (BACKGROUND): Any text which does not fall into the other categories.
Float (FLOAT): Digits with single decimal surrounded by whitespace or punctuation.
Integer (INTEGER): Non-sensitive digits without decimal, surrounded by whitespace or punctuation.
Ordinal (ORDINAL): Representations of order/position via words or alphanumeric mixes or version identifiers. Versions must have an alpha character associated with the version unless it is 3 digits separated by periods.
Quantity (QUANTITY): Integer or float prepended/appended with text or quantity characters.

Throughout the paper, we refer *entity/entity type* as entity category, *entity format* as variants per entity type, and *entity value* as values per format of each entity. For example, entity CREDIT_CARD has a format XXXX-XXXX-XXXX-XXXX, of which one of the values is 1111-2222-3333-4444.

Synthetic Data Generation

Both structured and unstructured data formats are synthetically generated: (1) unstructured text, (2) multi-column structured data, and (3) single-column structured data.

Unstructured Text Unstructured lines of text are randomly generated without context for entity placement. To incorporate different text structures, the following text formats are included. (a) sentences, (b) JSON, and (c) delimited data. JSON and delimited data are ingested as text to investigate the case where such structured data are read as text intentionally or due to the schema error. For all formats, a list of adjectives, adverbs, nouns, verbs, stop words, and punctuation are generated from the WordNet corpus [1] and considered background words. These words occur with a probability distribution obtained from the brown corpus [2], which are then randomly uppercased and combined with the random delimiters such as "-" and ",". For a non-background word, the selection is uniform, random across all other entities. The tunable parameters are given in (Truong, Walters, and Goodsitt 2020). Combining background and NPI words, the three unstructured formats are generated as follows:

- *Sentences*: Data are generated by first randomly determining the number of background words in a sentence, and then iteratively, randomly determining if each subsequent word is background or otherwise until the number number of background words is met.
- *JSON*: Words are randomly generated for keys and values with no allowed nesting.
- *Delimited Data*: Similar to random sentences, words are iteratively generated except separated by a randomly chosen delimiter (",", " ", "\t", ";", "\x00", "\x01").

The above formats can be tuned to have different lengths and probability of NPI occurrence or co-occurrence.

Multi-Column Structured Data Tabular datasets with 1 to 20 columns are generated with random schemas in CSV, PARQUET, JSON, and AVRO formats. Each column has a 50% probability of being background with uniform random probability for all other entities. All entities within a column contain the same format.

Single-Column Structured Data Tabular single-column datasets are generated in CSV, PARQUET, JSON, and AVRO formats. Each dataset has uniform random probability for all entities. All entities within the dataset contain the same format.

Internal Structured Data

Twenty-five schemas found within a financial institution were replicated via synthetic data, matching the format and statistics of the underlying real data. Each dataset contains 1000 samples (where a row is a sample). When the dataset has a tabular schema, CSV, PARQUET, AVRO, and JSON files are created. Otherwise, only JSON and AVRO datasets are created due to the nested structure of the dataset.

Public Email

A random subsample of emails from the Enron, Trec07, and ADCG SS14 Challenge corpora [3] are selected for manual labeling. For each corpus, 200 unique emails are randomly selected. Each email can have full header format or reduced header format containing only Date, From, To, and Subject. Not all entities existed in this dataset.

Datasets for Entity Detection Task

Datasets and the corresponding number of entities for train and test data are given in Table 1. For synthetic structured data, the train (test) data contain 50 (30) variations of multi-column schemas each with 100 (50) samples, and 250 (25) variations of single-column schemas each with 200 (200) samples. The tunable parameters for unstructured text are detailed in (Truong, Walters, and Goodsitt 2020).

Data	Train	Test
Total (K)	413	841
Unstructured Text (K)	193	22
Multi-column Structured Data (K)	121	19
Single-column Structured Data (K)	99	32
Internal Structured Data (K)		25
Public Email (K)		768

Table 1: Number of entities in the train and test datasets

Datasets for Column-wise Entity Prediction Task

Separate training and testing datasets are generated for evaluation of columnar-level models, which predict entity type for a given column.

Training Each training dataset contains 75k entity values. To investigate the value-grouping effect on the prediction accuracy, values are randomly subsampled for a given entity type and aggregated in different sizes. Ten datasets are generated with aggregate sizes ranging from 1 to 10.

Testing Testing datasets are obtained from synthetic structured testing datasets and internal structured testing datasets. As with the training dataset, values are randomly subsampled from a column in a given dataset and aggregated with sizes ranging from 1 to 10. This aggregation approach is resampled 10 times for the columns in each dataset.

Modeling Approaches

Character-level Neural Network Models

Our general character-level model architecture is given in Figure 1. Input strings are character encoded using ASCII indices. As string lengths vary across samples, the zero padding is applied at the end of each sample. To represent these encoded indices in latent character features, a pretrained Glove Character embedding (Pennington, Socher, and Manning 2014) is used. The embeddings are fed into the next layers such as CNN and LSTM to extract more detailed features. Finally, a prediction network, fully connected layers or a tag decoder such as CRF, is applied to optimize the sequence prediction. From this general model architecture, the following models are considered in our paper: CNN, LSTM, CNN-LSTM, CNN-CRF, and BiLSTM-CRF. These models are selected to investigate (i) the effect of convolutional layer, recurrent network layer and their combination on feature learning, and (ii) the

[1] https://wordnet.princeton.edu/

[2] http://korpus.uib.no/icame/brown/bcm.html

[3] https://www.cs.cmu.edu/ enron/, https://plg.uwaterloo.ca/ gvcormac/treccorpus07/, https://www.kaggle.com/c/adcg-ss14-challenge-02-spam-mails-detection/data

effect of additional tag decoder layer (CRF) to the overall prediction accuracy of the models.

Data processing optimizations
In order to maximize the throughput of our models in a production environment, the input character encoding component is integrated into the tensorflow computation graph. In addition, a flattening mechanism is applied to the input data as illustrated in Figure 2. Input characters are all concatenated and chunked into an array of *max_length* characters. This step increases the model throughput on both CPU instances (6x) and GPU instances (3x-4x).

CNN Model

Detailed in Figure 3(a), the core components of the CNN model are the four convolutional blocks followed by two connected blocks. Each convolutional block consists of a 1-d convolutional layer, dropout layer, and batch normalization layer. Each fully-connected block consists of a dense layer and dropout layer. Dropout layers help regularize the network, and increases the accuracy since our data contains random context. Batch normalization is applied to reduce the effect of the internal covariate shift (Ioffe and Szegedy 2015). Through manual optimization, the CNN model obtains the best accuracy with the following parameters: *epochs=10, num-conv-layer=4, num-dense-layer=2, batch-size=24, embedding-dimension=64, max-length=3400, filter-size=13, dense-layer-size=96, and dropout=7.3%*.

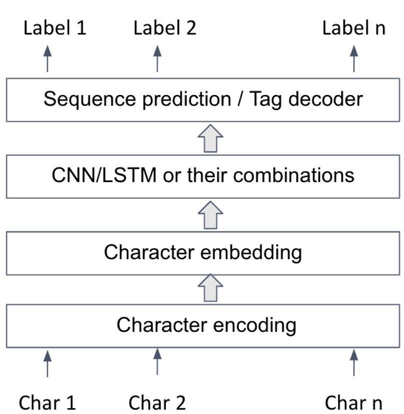

Figure 1: Character-level neural network model architecture

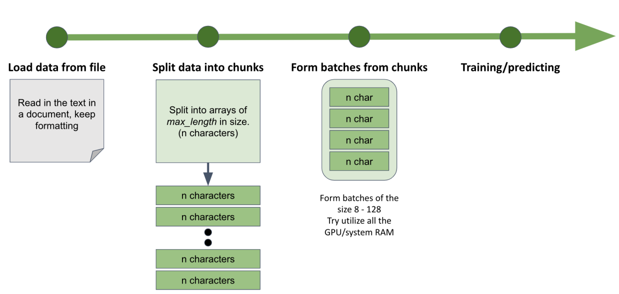

Figure 2: Flattening preprocessing for input text

LSTM Model

Throughout manual optimization, only one LSTM layer is utilized and illustrated in Figure 3(b). The best parameters for the LSTM model are given as: *epochs=10, num-lstm-layer=1, batch-size=24, embedding-dimension=64, max-length=3400, lstm-size=64, activation=tanh, recurrent-dropout=10%, dense-layer-size 32, and dropout 10%*. CuDNNLSTM layers are used instead of the LSTM layer to optimize throughput as we will see later that CuDNNLSTM shows the significant run-time improvement without sacrificing the accuracy.

CNN-LSTM Model

We consider the combined model in which the CNN layers as in Figure 3(a) is put before the LSTM layer followed by the 2-layer dense network. The detailed model is given in Figure 3(c). Similar to the LSTM model, CuDNNLSTM layer is used in place of LSTM layer to optimize throughput. The best obtained parameters are given as: *epochs=10, num-conv-layer=4, num-lstm-layer=1, num-dense-layer=2, batch-size=24, embedding-dimension=64, max-length=3400, filter-size=13, lstm-size=64, activation=tanh, recurrent-dropout=10%, dense-layer-size=96, and dropout=10%.*

CRF-based Models

CRF has shown provable advantages over the fully connected layer for tag decoding step as it is able to learn the label of each character based on its neighbors (Huang, Xu, and Yu 2015; Kuru, Can, and Yuret 2016). To investigate the effectiveness of this CRF layer as tag decoder, two following models are investigated.

CNN-CRF: same architecture as the CNN model except the dense layers are replaced by the CRF layer illustrated in Figure 3(d). Similar to LSTM related models, CRF based models run slowly compared to other models. Through the optimization process, the best parameters for this model is given as: *epochs=15, num-crf-layer=1, num-conv-layer=4, batch-size=128, embedding-dimension=64, max-length=3400, filter-size=13, dropout=7.3%, optimizer=rmsprop.*

BiLSTM-CRF: suggested as the best model on various NER tasks (Huang, Xu, and Yu 2015), the BiLSTM layer is swapped with the CNN layer from the CNN-CRF model as depicted in Figure 3(e). The best obtained parameters are as follows: *epochs=8, num-crf-layer=1, num-bilstm-layer=1, batch-size=128, embedding-dimension=64, max-length=2500, bilstm-layer=1, lstm-size=64, activation=tanh, recurrent-dropout=0%, merge-mode=concat, dropout=20%, optimizer=rmsprop*.

Existing NER Models

Regex A list of regular expressions for all entities except PERSON (too large of a search space) and BACKGROUND is manually generated with respect to the training dataset.

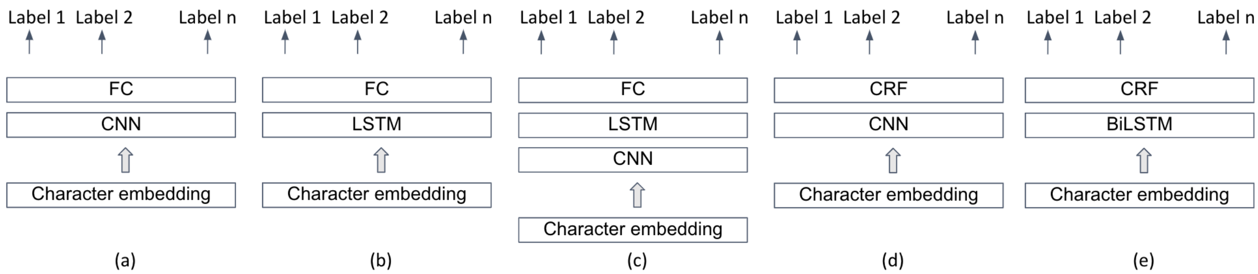

Figure 3: Character-level models. (a) Character-level CNN, (b) Character-level LSTM, (c) Character-level CNN-LSTM, (d) Character-level CNN-CRF, (e) Character-level BiLSTM-CRF

All regex rules are applied to the input text, sets of characters not matching a rule are considered BACKGROUND and labels were evenly split for ties. Since regex can become quite complex, only simple regex expressions or those which are quickly discoverable online are used. Additionally, each regex pattern is surrounded by encapsulators which ensure that any matching string in unstructured text is delimited by the specified characters. The detailed regex pattern is given in (Truong, Walters, and Goodsitt 2020).

CRF Model with Handcrafted Features (Ngram-CRF) A standalone CRF model with the following handcrafted feature extraction is considered:

-*char.lower()*: get the lowercase character
-*char.isupper()*: check if the character is uppercased
-*char.isdigit()*: check if the character is digit
-*char.isalnum()*: check if the character is alphanumeric

For each character, its extracted features are combined with features of its neighbors within a sliding window, instead of being fed into encoding and embedding components. Through manual optimization, the model obtain the best results with the parameters: *window-len=4, batch-size=1000, max-length=2500, l1-coefficient=0.1, l2-coefficient=0.1, max-iterations=100, all-possible-transitions=True, all-possible-states=True*

spaCy Model The spaCy model is fine-tuned on our training datasets with the default parameters. As spaCy processes data at the token level, the input strings are fed to this model without splitting to the character level. However, at the evaluation stage, the predictions from the spaCy model are split into character labels from token results to compare with other models.

Columnar-Level Models

Four model variations are evaluated: (1) the best character-level model trained on the unstructured training dataset given in Table 1 (Char-Best-Pretrained), (2) the best character-level model trained on the columnar-level dataset (Char-Best-Retrained), (3) a columnar-based CNN-CRF model (Column-CNN-CRF), and (4) a columnar-based CNN-BiLSTM model (Column-CNN-BiLSTM). Each model utilizes subsampled columns to make generalized predictions for an entire column.

Best Character-Level Model For the best character-level models, data are preprocessed by taking a number of sampled rows per column and concatenating them into a single sample delimited by five \x01 characters. Postprocessing on the model output is applied to convert the character entity values into a single subsample entity by taking the mode of character entity values, excluding PAD and separator characters. In cases of a tie during prediction, a non-background entity is randomly selected.

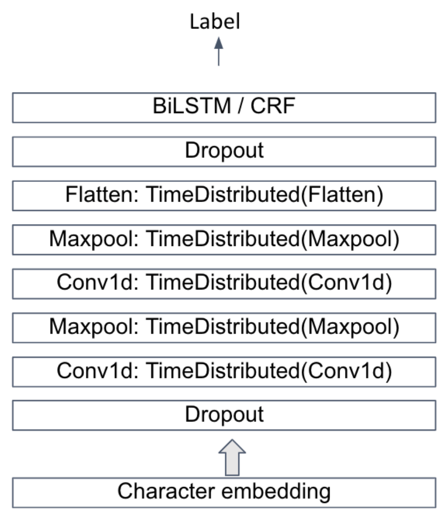

Figure 4: Columnar-level neural network model

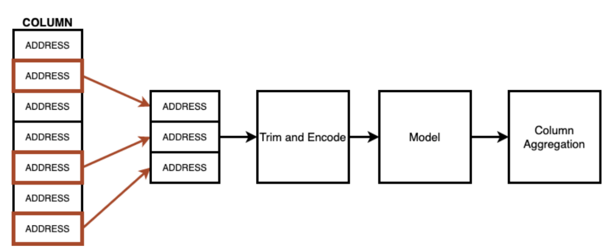

Figure 5: Columnar-level model workflow

Columnar-Based CNN Models Subsampled rows in a column were processed similarly to words in a sentence (Chiu and Nichols 2016; Fritzler, Logacheva, and Kretov 2019) within the columnar-based CNN approaches and fed into the model as a single sample. Before being fed into the model, each word is limited to 52 characters and subsequently encoded. The model architecture, derived from (Chiu and Nichols 2016), is described in Figure 4. The model output is an entity per subsampled row within the column which was aggregated via the mode into a single value, identical to the best character-level models. The following were the parameter values for these models (where applicable): *dropout=0.5, num-conv-filters=30, conv-size=9, dropout-recurrent=0.25, lstm-state-size=100, learning-rate=0.0105, optimizer='nadam', num-conv-layers=2, embedding-dim=30*. Postprocessing on the model output is applied to convert the word entity values into a single subsample entity by taking the mode of character entity values, as seen in Figure 5. In cases of a tie during prediction, a non-background entity is randomly selected. This subsample entity is the assumed generalized entity selection for the column.

Evaluation Results

Evaluation Metrics

In this paper, precision, recall, and F1-score are used as the model accuracy comparison metrics. Performance on the test dataset is evaluated using micro and macro averages across all entities excluding PAD and BACKGROUND. In addition to accuracy, model throughput, measured as GB of data processed per hour (GB/hr), is evaluated on both CPU and GPU AWS EC2 instances.

Entity detection on Multiple Data Formats

Accuracy Evaluation Table 2 shows results of the models on the test set. Model performance is consistently higher on the synthetic datasets which have more similar schema and context to the training dataset as opposed to the internal and email datasets which do not. Additionally, regex, spaCy, Ngram-CRF, and the LSTM model are less accurate than the other models. Models with CRF layer as the tag decoder such as CNN-CRF, BiLSTM-CRF, and the combined CNN-CuDNNLSTM model provide marginal to no improvement over the CNN model. Detailed results regarding individual entities are described in (Truong, Walters, and Goodsitt 2020).

Throughput Evaluation Model throughput is evaluated on a CPU instance, c5.2xlarge (8 vCPU, 16 GiB Memory), and varying GPU instances, g4dn.xlarge (4 vCPU, 16 GiB Memory, 1 Tesla T4 GPU, 16 GiB GPU Memory), g4dn.8xlarge (32 vCPU, 128 GiB Memory, 1 Tesla T4 GPU, 16 GiB GPU Memory), p3.2xlarge (8 vCPU, 61 GiB Memory, 1 Tesla V100 GPU, 16 GiB GPU Memory), on the entire test dataset as shown in Table 3. The CNN model is the highest performing model on both CPU & GPU instances. The next highest throughput models are the CuDNNLSTM and CNN-CuDNNLSTM models which are 2x-5x slower as they contain the LSTM layers. Note that unlike CuDNNLSTM layers, the regular LSTM layers can be utilized on the CPU, but are substantially slower on GPU.

CRF-based models (CNN-CRF, BiLSTM-CRF) suffer low throughput. Additionally, since the CRF layer relies on an RNN implementation, the models CRF-CNN, LSTM, BiLSTM-CRF, and CNN-LSTM have similar throughput. The CRF model with the lowest throughput is Ngram-CRF, likely due to its implementation using the sklearn-crfsuite package which supports CPU only.

For spaCy model, we configured the data pipeline to achieve maximum throughput. This data pipeline is different from our accuracy evaluation, and could result in slightly reduced accuracy. The spaCy model obtained higher throughput on the GPU rather than CPU, but is one of the slowest lowest performing models. Note that the SpaCy library uses the CuPy to execute the graph on GPU and does not take advantage of optimizations provided by Tensorflow, such as CuDNN.

Column-wise Entity Prediction on Tabular Datasets

In this section, we evaluate only structured datasets whose columns contain consistent entity formats. Our goal is to predict the entity type of each column. The columnar-level models along with the CNN model, the most optimal model in terms of accuracy and throughput, are evaluated.

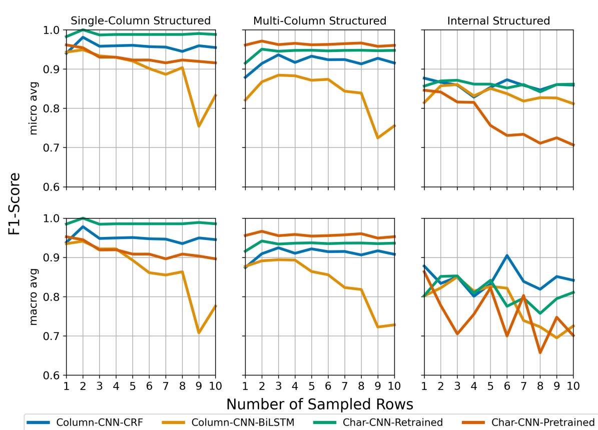

Figure 6: Accuracy (F1-score) for columnar-level models

Accuracy Evaluation The accuracy results for column-wise entity prediction are given in Figure 6. Char-CNN-Retrained, Char-CNN-Pretrained and Column-CNN-CRF models obtained the best macro-average f1-score which depicts how the Char-CNN-Retrained and Char-CNN-Pretrained are context and task invariant after slight modifications to the data processing pipeline. Additionally, increasing the number of sampled rows does not appear to improve model accuracy. However, improvement via increased sampled rows may be dependent on the complexity of each entity as seen in the breakdown of accuracy results for individual sensitive entities given in (Truong, Walters, and Goodsitt 2020).

Model / Datasets	Multi-column, Structured Data	Single-column, Structured Data	Unstructured Text	Public Emails	Internal, Structured Data
Char CNN	**(0.99, 0.99, 0.99)** **(0.98, 0.97, 0.97)**	**(0.99, 0.99, 0.99)** **(0.97, 0.97, 0.97)**	**(0.98, 0.96, 0.97)** **(0.96, 0.93, 0.94)**	(0.67, 0.79, 0.73) (0.54, 0.78, 0.60)	(0.82, 0.87, 0.84) (0.74, 0.83, 0.72)
Char CuDNNLSTM	(0.94, 0.93, 0.93) (0.89, 0.89, 0.88)	(0.95, 0.93, 0.94) (0.87, 0.82, 0.84)	(0.84, 0.83, 0.83) (0.82, 0.76, 0.76)	(0.55, 0.61, 0.58) (0.46, 0.64, 0.51)	(0.54, 0.80, 0.65) (0.54, 0.70, 0.59)
Char CNN + CuDNNLSTM	**(0.99, 0.98, 0.99)** **(0.97, 0.96, 0.97)**	**(0.98, 0.98, 0.98)** (0.93, 0.93, 0.93)	(0.96, 0.94, 0.95) (0.92, 0.89, 0.89)	**(0.73, 0.75, 0.74)** **(0.61, 0.75, 0.64)**	**(0.84, 0.90, 0.87)** **(0.76, 0.85, 0.75)**
Word spaCy	(0.86, 0.84, 0.85) (0.82, 0.74, 0.77)	(0.96, 0.96, 0.96) (0.93, 0.90, 0.90)	(0.75, 0.77, 0.76) (0.74, 0.70, 0.71)	(0.62, 0.62, 0.62) (0.52, 0.60, 0.50)	(0.48, 0.66, 0.56) (0.53, 0.67, 0.48)
Char Ngram + CRF	(0.97, 0.94, 0.95) (0.90, 0.87, 0.88)	(0.97, 0.96, 0.96) (0.91, 0.85, 0.86)	(0.91, 0.88, 0.90) (0.82, 0.80, 0.80)	(0.64, 0.71, 0.67) (0.49, 0.67, 0.54)	(0.74, 0.84, 0.79) (0.67, 0.74, 0.68)
Char CNN + CRF	**(0.99, 0.99, 0.99)** **(0.98, 0.98, 0.98)**	**(0.99, 0.99, 0.99)** **(0.96, 0.96, 0.96)**	**(0.98, 0.97, 0.97)** **(0.96, 0.94, 0.95)**	**(0.70, 0.81, 0.75)** **(0.55, 0.79, 0.62)**	**(0.83, 0.92, 0.87)** **(0.71, 0.85, 0.74)**
Char BiLSTM + CRF	**(0.99, 0.98, 0.99)** **(0.98, 0.96, 0.97)**	**(0.99, 0.98, 0.98)** (0.97, 0.93, 0.95)	**(0.98, 0.95, 0.96)** (0.95, 0.92, 0.93)	**(0.71, 0.78, 0.74)** (0.56, 0.73, 0.60)	(0.79, 0.86, 0.82) (0.73, 0.81, 0.71)
Char Regex	(0.71, 0.73, 0.72) (0.76, 0.60, 0.63)	(0.91, 0.81, 0.85) (0.79, 0.55, 0.61)	(0.67, 0.67, 0.67) (0.73, 0.58, 0.60)	(0.64, 0.62, 0.63) (0.54, 0.62, 0.53)	(0.31, 0.50, 0.38) (0.61, 0.54, 0.49)

Table 2: Evaluation results on the test set for sensitive entities detection. In each cell, the first line and second line shows the micro and macro average results, respectively. Each line represents precision, recall and F1-score respectively. Synthetic and real data are given in the first three columns and the last two columns, respectively.

Model / EC2 Instance Type	c5.2xlarge	g4dn.xlarge	g4dn.8xlarge	p3.2xlarge
Char CNN	**3.01**	**18.08**	**18.18**	**28.53**
Char LSTM	0.9574	0.4668	0.4328	0.3833
Char CuDNNLSTM	N/A	7.893	7.102	6.346
Char CNN+LSTM	0.7675	0.4407	0.4025	0.3701
Char CNN+CuDNNLSTM	N/A	4.704	4.258	5.031
Word spaCy (Unflattened)	0.054	0.045	0.045	0.032
Word spaCy (Flattened)	0.071	0.134	0.13	0.11
Char Ngram + CRF	0.087	0.071	0.087	0.087
Char CNN + CRF	0.8673	0.312	0.4014	0.2455
Char BiLSTM + CRF	0.543	0.0839	0.0979	0.0685
Char Regex	0.8388	0.7344	0.734	0.6055

Table 3: Throughput in GB/hr evaluation on four different AWS EC2 instances

Throughput Evaluation Figure 7 illustrates model throughput for the column-wise prediction task. Despite similar accuracy performance with Column-CNN-CRF, the Char-CNN models had 3x-10x lower throughput. Throughput differences may be attributed to the larger parameter count of the Char-CNN models due to both CNN hyperparameter differences such as filter-size, num-filters, numconv, and the output layer being significantly larger because of the character level prediction. Additionally, Char-CNN models did not use flattening mechanism which could also contribute to the throughput decreasing.

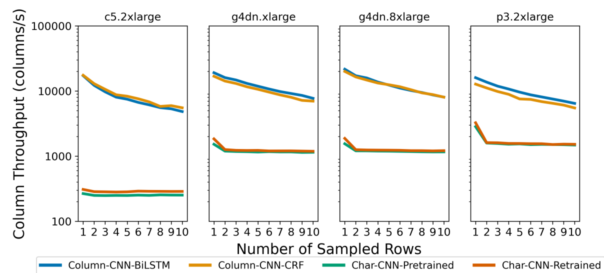

Figure 7: Throughput evaluation for columnar-level models.

Conclusions

In this paper, we tackle the problems of identifying the sensitive information in different data formats for financial institutions. We optimize a set of neural network models to be deployed in the production environment(s). Our evaluation results on both synthetic data, real email and internal data shows that the CNN model is simple yet very effective with respect to accuracy and throughput, and thus the most suitable model for the production. We believe this work will shed some light on this challenging problem from which several lessons learned along with future directions are discussed as follows:

- *Limitations of experimental data* The synthetic data is generated without context which may reduce the model performance on the real data with context, e.g., email datasets. Further directions may include the data generation with more realistic background which can be generated from some generative models such as GAN.

- *Data labeling* There exists some discrepancy in data labeling for the real email dataset, which might affect the overall results. However, it is worth noting that this is an intrinsic and common problem for this framework.
- *Overlaps of entities* There exist overlap values among several entities such as BAN, Phone-Number and SSN. Unless extra data (e.g., column names and statistical attributes of the value ranges) is provided, this problem appears inevitable in this framework.

References

Boukkouri, H. E.; Ferret, O.; Lavergne, T.; Noji, H.; Zweigenbaum, P.; and Tsujii, J. 2020. CharacterBERT: Reconciling ELMo and BERT for Word-Level Open-Vocabulary Representations From Characters .

Caliskan Islam, A.; Walsh, J.; and Greenstadt, R. 2014. Privacy Detective: Detecting Private Information and Collective Privacy Behavior in a Large Social Network. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, 3546. New York, NY, USA.

Chiu, J. P.; and Nichols, E. 2016. Named Entity Recognition with Bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics* 4: 357–370.

Cho, H.; and Lee, H. 2019. Biomedical named entity recognition using deep neural networks with contextual information. In *BMC Bioinformatics*.

Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; and Kuksa, P. 2011. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research* 12: 24932537.

Dutt, R.; Hiware, K.; Ghosh, A.; and Bhaskaran, R. 2018. SAVITR: A System for Real-Time Location Extraction from Microblogs during Emergencies. In *Companion Proceedings of the The Web Conference 2018*, 1643–1649.

Eddy, S. R. 1996. Hidden Markov models. In *Current Opinion in Structural Biology*, 361–365.

Francis, S.; Van Landeghem, J.; and Moens, M.-F. 2019. Transfer Learning for Named Entity Recognition in Financial and Biomedical Documents. *Information* 10(8).

Fritzler, A.; Logacheva, V.; and Kretov, M. 2019. Few-Shot Classification in Named Entity Recognition Task. In *SAC19: Proceedings of the 34th ACM, SIGAPP Symposium on Applied Computing*, 993–1000. New York, NY, USA.

Gelernter, J.; and Zhang, W. 2013. Cross-Lingual Geo-Parsing for Non-Structured Data. In *Proceedings of the 7th Workshop on Geographic Information Retrieval*, 6471. New York, NY, USA: Association for Computing Machinery.

Gui, T.; Ma, R.; Zhang, Q.; Zhao, L.; Jiang, Y.-G.; and Huang, X. 2019. CNN-Based Chinese NER with Lexicon Rethinking. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 4982–4988.

Huang, Z.; Xu, W.; and Yu, K. 2015. Bidirectional LSTM-CRF Models for Sequence Tagging. *CoRR* .

Ioffe, S.; and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR* abs/1502.03167.

Kapur, J. N. 1989. *Maximum-entropy models in science and engineering*. John Wiley & Sons.

Kurniawan, K.; and Louvan, S. 2018. Empirical Evaluation of Character-Based Model on Neural Named-Entity Recognition in Indonesian Conversational Texts. In *Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text*, 85–92. Brussels, Belgium.

Kuru, O.; Can, O. A.; and Yuret, D. 2016. CharNER: Character-Level Named Entity Recognition. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics*, 911–921. Japan.

Lafferty, J. D.; McCallum, A.; and Pereira, F. C. N. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 282289. San Francisco, CA, USA.

Malmasi S., D. M. 2016. Location Mention Detection in Tweets and Microblogs. In *Computational Linguistics. PACLING 2015. Communications in Computer and Information Science*. Singapore.

Pennington, J.; Socher, R.; and Manning, C. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. Doha, Qatar.

Peters, M.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2227–2237. Louisiana.

Ritter, A.; Clark, S.; Mausam; and Etzioni, O. 2011. Named Entity Recognition in Tweets: An Experimental Study. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, 1524–1534. Edinburgh, Scotland, UK.: Association for Computational Linguistics.

Shen, Y.; Yun, H.; Lipton, Z.; Kronrod, Y.; and Anandkumar, A. 2017. Deep Active Learning for Named Entity Recognition. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, 252–256. Vancouver, Canada.

Truong, A.; Walters, A.; and Goodsitt, J. 2020. Sensitive Data Detection with High-Throughput Neural Network Models for Financial Institutions (Long version). *CoRR* URL https://arxiv.org/abs/2012.09597.

Xu, G.; Qi, C.; Yu, H.; Xu, S.; Zhao, C.; and Yuan, J. 2019. Detecting Sensitive Information of Unstructured Text Using Convolutional Neural Network. In *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, 474–479.

Yang, J.; Zhang, Y.; and Dong, F. 2017. Neural Reranking for Named Entity Recognition. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017*, 784–792. Varna, Bulgaria.