# Pairwise FastText Classifier for Entity Disambiguation

**Cheng Yu[a,b], Bing Chu[b], Rohit Ram[b], James Aichinger[b], Lizhen Qu[b,c], Hanna Suominen[b,c]**
[a] Project Cleopatra, Canberra, Australia
[b] The Australian National University
[c] DATA 61, Australia
cheng@projectcleopatra.com.au
{u5470909,u5568718,u5016706,
Hanna.Suominen}@anu.edu.au
Lizhen.Qu@data61.csiro.au

## Abstract

For the *Australasian Language Technology Association* (ALTA) 2016 Shared Task, we devised *Pairwise FastText Classifier* (PFC), an efficient embedding-based text classifier, and used it for entity disambiguation. Compared with a few baseline algorithms, PFC achieved a higher F1 score at 0.72 (under the team name BCJR). To generalise the model, we also created a method to bootstrap the training set deterministically without human labelling and at no financial cost. By releasing PFC and the dataset augmentation software to the public[1], we hope to invite more collaboration.

## 1 Introduction

The goal of the ALTA 2016 Shared Task was to disambiguate two person or organisation entities (Chisholm et al., 2016). The real-world motivation for the Task includes gathering information about potential clients, and law enforcement.

We designed a *Pairwise FastText Classifier* (PFC) to disambiguate the entities (Chisholm et al., 2016). The major source of inspiration for PFC came from *FastText* [2] algorithm which achieved quick and accurate text classification (Joulin et al., 2016). We also devised a method to augment our training examples deterministically, and released all source code to the public.

The rest of the paper will start with PFC and a mixture model based on PFC, and proceeds to present our solution to augment the labelled dataset deterministically. Then we will evaluate PFC's performance against a few baseline methods, including SVC[3] with hand-crafted text features. Finally, we will discuss ways to improve disambiguation performance using PFC.

## 2 Pairwise Fast-Text Classifier (PFC)

Our *Pairwise FastText Classifier* is inspired by the *FastText*. Thus this section starts with a brief description of *FastText*, and proceeds to demonstrate PFC.

### 2.1 FastText

*FastText* maps each vocabulary to a real-valued vector, with unknown words having a special vocabulary ID. A document can be represented as the average of all these vectors. Then *FastText* will train a maximum entropy multi-class classifier on the vectors and the output labels. Fast Text has been shown to train quickly and achieve prediction performance comparable to Recurrent Neural Network embedding model for text classification (Joulin et al., 2016).

### 2.2 PFC

PFC is similar to *FastText* except that PFC takes two inputs in the form of a list of vocabulary IDs, because disambiguation requires two URL inputs. We specify that each of them is passed into the same embedding matrix. If each entity is represented by a d dimensional vector, then we can concatenate them, and represent the two entities

[1] All source code can be downloaded from: `https://github.com/projectcleopatra/PFC`

[2] The original paper of FastText used the typography `fastText`

[3] SVC: Support vector classification

by a 2d dimensional vector. Then we train a maximum entropy classifier based on the concatenated vector. The diagram of the model is in Figure 1.

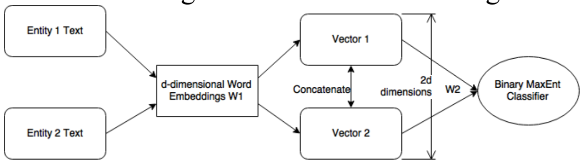

Figure 1: PFC model. W1 and W2 are trainable weights.

### 2.3 The PFC Mixture Model

The previous section introduces word-embedding-based PFC. In order to improve disambiguation performance, we built a mixture model based on various PFC sub-models: Besides word-embedding-based PFC, we also trained character-embedding-based PFC, which includes one uni-character PFC, and one bi-character PFC. In the following subsections, we will first briefly explain character-embedding-based PFC, and then show the Mixture model.

#### 2.3.1 Character-Embedding-Based PFCs

Character-embedding-based PFC models typically have fewer parameters than word-embedding-based PFC, and thus reducing the probability of overfitting.

Uni-character embedding maps each character in the URL and search engine snippet into a 13-dimensional vector, take the average of an input document, concatenate the two documents, and then train a maximum entropy classification on top of the concatenated vectors.

Bi-character embedding model has a moving window of two characters and mapped every such two characters into a 16-dimensional vector.

Our implementation of the character-embedding based PFC model includes only lowercase English letters and space. After converting all letters to lowercase, other characters are simply skipped and ignored.

#### 2.3.2 Mixing PFC Sub-models

The mixture model has two phases. In phase one, we train each sub-model independently. In phase 2, we train a simple binary classifier based on the probability output of each individual PFC. The diagram of the PFC mixture model is shown in Figure 2.

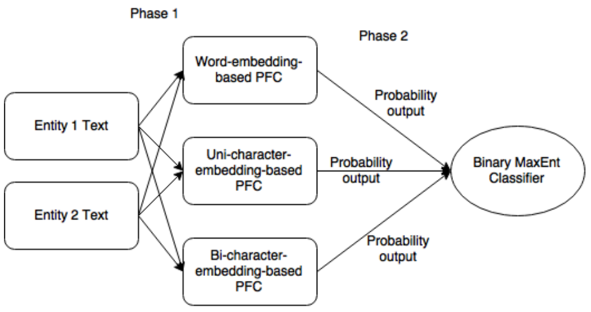

Figure 2: The PFC Mixture Model.

## 3 Augmenting More Training Examples Deterministically

Embedding-models tend to have a large number of parameters. Our word-embedding matrix has over 3700 rows, and thus it is natural to brainstorm ways to augment the training set to prevent overfitting.

We created a method to harvest additional training examples deterministically without the need for human labelling, and the data can be acquired at no additional cost.

### 3.1 Acquiring Training Examples for the Negative Class[4]

To acquire URL pairs that refer to different people, we wrote a scraping bot that visits LinkedIn, and grabs hyperlinks in a section called "People that are similar to the person", where LinkedIn recommends professionals that have similar to the current profile that we are browsing. LinkedIn restricts the number of profiles we can browse in a given month unless the user is a Premium user, so we upgraded our LinkedIn account for scraping purpose. We used the LinkedIn URLs provided to us in the training samples, and grabbed similar LinkedIn profiles, which ended up with about 850 profiles, with some of the LinkedIn URLs no longer up to date.

### 3.2 Acquiring Training Examples for the Positive Class

To acquire training examples of different social media profiles that belong to the same person, we used examples from `about.me`. About.me is a platform where people could create a personal page showing their professional portfolios and links to various social media sites. We wrote a scraping bot that visits about.me/discover, where the site showcases their users, and clicks open

[4] In the Shared Task, if a pair of URL entities refer to different persons or organisations, the pair belongs to the negative class. if a pair of URL entities refer to the same persons or organisations, the pair belongs to the positive class.

each user, acquires their social media links, and randomly selects two as a training example. For example, for someone with 5 social media profiles, including Facebook, Twitter, LinkedIn, Pinterest, and Google+, the bot can generate (5, 2) = 10 training examples.

## 4 Experimental Setup

Using the training data provided by the Organiser and data acquired using the method mentioned in Section 3, we evaluated the performance of our PFC and PFC Mixture against a few baseline models.

### 4.1 Datasets

The organiser prepared 200 labelled pairs of training samples and 200 unlabelled test samples (Hachey, 2016). All baseline methods and PFC methods are trained using the original 200 URL pairs. The only exception is "PFC with augmented dataset", which uses the method in the previous section to acquire 807 negative class URL pairs, and 891 positive class URL pairs.

### 4.2 Pre-Processing

Text content for the PFC comes from the search engine snippet file provided by the Organiser and text scraped from the URLs provided by the training examples. Unknown words in the test set are represented by a special symbol.

### 4.3 Baselines

The reason we choose a few baseline models is that there is no gold-standard baseline model for URL entity disambiguation. Baseline models are explained as followed.

**Word-Embedding with Pre-Trained Vectors:** The training corpus Google comes from News Articles (Mikolov et al., 2013). For each URL entity, we calculated the mean vector of the search result snippet text by using pre-trained word embedding vectors from Google. Unknown words were ignored. Then we concatenated the vectors and trained a maximum entropy classifier on top of it.

**SVC with Hand-Selected Text Features:** Our Support Vector Classifier is built on top of hand-selected text features. For each pair of URLs, we manually selected the following text features. Explanation of these features is available in Appendix-A.

**LSTM Word-Embedding:** We passed each document token sequentially using word embedding into an LSTM layer with 50 LSTM units (Brownlee, 2016) (Goodfellow et al., 2016), concatenated the two output vectors, and trained a maximum entropy classifier on top of it. To reduce overfitting, we added dropout layers with the dropout parameter set to 0.2 (Zaremba, Sutskever, & Vinyals, 2014).

**Neural Tensor Network:** Inspired by Socher et al., by passing a pair of documents represented in vector form into a tensor, we built a relationship classifier based on the architecture in the paper (Socher et al., 2013). Document vectors are calculated from pre-trained Google embedding word vectors.

## 5 Results and Discussion

The experimental results from the setup is summarised in the table.

| | **Method** | **F1 Public** | **F1 Private** | **F1 Total**[5] |
|---|---|---|---|---|
| PFC-based | PFC with Word-Embedding | 0.75 | 0.64 | 0.69 |
| | PFC Mixture Model | 0.74 | 0.71 | 0.72 |
| | PFC with augmented dataset | 0.65 | 0.69 | 0.67 |
| Baseline | Neural tensor network | 0.67 | 0.6 | 0.64 |
| | SVC using hand-selected features | 0.75 | 0.69 | 0.72 |
| | LSTM word-embedding | 0.51 | 0.53 | 0.52 |

Table 1: Result comparison.

### 5.1 Issues with Augmented Dataset

Adding more training data seems to hurt the F1 score for the Shared Task. However, if we allow the newly acquired training examples to be part of the validation set, the validation set accuracy could reach 0.92. Due to time constraint, we were only able to acquire about 1700 training examples,

[5] F1 total is the simple average of F1 Public (calculated from half of half of the test data) and F1 Private (from the second half of the data)

with approximately equal number in each category. Whether adding more training data can improve disambiguation performance remains to be experimented.

### 5.2 Improve PFC

The performance of the PFC might improve if we use a similarity scoring function $s(v_1, v_2) = v_1^T \mathcal{D} v_2$, where $\mathcal{D}$ is a diagonal matrix. The binary classifier becomes $y = \sigma(s(v_1, v_2))$, while the original PFC classifier is $y = \sigma(\mathcal{W}^T[v_1, v_2])$. Both $\mathcal{D}$ and $\mathcal{W}$ are learnable weights.

### 5.3 Compare PFC with Baseline SVC

In our experiments, the PFC mixture model achieves the best performance, comparable to SVC with hand-selected features. Uni-character model by itself tends to under fit because the training data themselves cannot be separated by the model alone. PFC is robust because allows text features to be learnt automatically.

## 6 Conclusion

We introduced *Pairwise FastText Classifier* to disambiguate URL entities. It uses embedding-based vector representation for text, can be trained quickly, and performs better than most of the alternative baseline models in our experiments. PFC has the potential to generalise towards a wide range of disambiguation tasks. In order to generalise the application of the model, we created a method to deterministically harvest more training examples, which does not require manual labelling. By releasing all of them to the public, we hope for the continual advancement in the field of disambiguation, which could be applied to identity verification, anti-terrorism, and online general knowledge-base creation.

## Appendix A

Appendix A includes manually selected text features for the SVC baseline model.

### A.1 URL Features

| ID | Feature Name | Description |
|---|---|---|
| 1 | Country code difference | If one URL has "au" and another one has "uk", then the value is 1, otherwise 0. |
| 2 | Edit distance between the two URLs | Simply the Levenshtein distance between the string tokens of the two URLs (Jurafsky & Martin, 2007). |
| | | Below are a list of URL features specific to one URL. |
| 3 | isEducation(url_a) | If the first URL contains domain names such as ".ac.uk" or ".edu", then the value is 1. Otherwise 0. |
| 4 | isEntertainment(url_a) | If the url includes imdb, allmusic, artnet, mtv.com, or band, it returns 1. Otherwise 0. |
| 5 | isProfessional(url_a) | If the url contains linkedin.com or researchgate.com, it returns 1. Otherwise 0. |
| 6 | isNonProfitOrGov(url_a) | If the url contains ".org" or ".gov", then it returns 1. |
| 7 | isSportsStar(url_a) | If the url contains "espn", "ufc.com", or "sports", then the feature is 1. Otherwise 0. |
| 8 - 12 | Features for url_b | Analogous to Feature 3 - 7 |

### A.2 Title Features

| ID | Feature Name | Description |
|---|---|---|
| 13 | Edit distance of the first part of the title for the two URLs | Due to the differences of length between different titles, only the first part of the titles is preserved for calculating the Levenshtein distance. This feature is chosen because the first part of the title usually contains the first and last name of the person or the name of the company. |
| 14 | Cosine distance of the embedded matrices | The vector representation of the text is same as FastText except that the embedding matrix is pre-trained from Google. Any token not trained by Google will be ignored (Weston, Chopra, & Bordes, 2015). |

### A.3 Snippet Features

This refers to features made from fields "ASnippet" and "BSnippet" of the search result file provided by the Organiser.

| ID | Feature Name | Description |
|---|---|---|
| 15 | Word Mover Distance between the nouns and named entities between "ASnippet" and "BSnippet" (Pele & Werman, A linear time histogram metric for improved sift | Using pre-trained Google word-embedding vectors |

| | | |
|---|---|---|
| | matching, 2008) (Pele & Werman, Fast and robust earth mover's distances, 2009). | |
| 16 | Word Mover Distance between the nouns and named entities between "ASnippet" and "BSnippet" | Using the pre-trained Stanford GloVe vectors (Pennington et al., 2014). |

## Reference

Brownlee, J. (2016, July 26). *Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras.* Retrieved from Machine Learning Mastery: http://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

Chisholm, A., Hachey, B., & Molla, D. (2016). Overview of the 2016 ALTA Shared Task: Cross-KB Coreference. *Proceedings of the Australasian Language Technology Association Workshop 2016*.

Chisholm, A., Radford, W., & Hachey, B. (2016). Discovering Entity Knowledge Bases on the Web. *Proceedings of the 5th Workshop on Automated Knowledge Base Construction*

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*, pages 373 - 420.

Hachey, B. (2016). *How was the data obtained? - ALTA 2016.* Retrieved from Kaggle: https://inclass.kaggle.com/c/alta-2016-challenge/forums/t/23480/how-was-the-data-obtained

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of Tricks for Efficient Text Classification. *arXiv preprint arXiv:1607.01759*.

Jurafsky, D., & Martin, J. H. (2007). *Speech and Language Processing, 3rd edition [Draft].* Chapter 2.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781.*

Pele, O., & Werman, M. (2008). A linear time histogram metric for improved sift matching. *Computer Vision--ECCV 2008*.

Pele, O., & Werman, M. (2009). Fast and robust earth mover's distances. *2009 IEEE 12th International Conference on Computer Vision*.

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. *Empirical Methods in Natural Language Processing*, pages 1532 – 1543.

Socher, R., Chen, D., Manning, C., & Ng, A. (2013). Completion, Reasoning With Neural Tensor Networks for Knowledge Base. In Advances in Neural Information Processing Systems, 2013a.

Weston, J., Chopra, S., & Bordes, A. (2015). Memory Networks. *arXiv:1410.3916*.

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent Neural Network Regularization. *arXiv preprint arXiv:1409.2329.*