Restoring Punctuation and Casing in English Text

Timothy Baldwin[1,2] and Manuel Paul Anil Kumar Joseph[1]

[1] Department of Computer Science and Software Engineering
University of Melbourne, VIC 3010, Australia
[2] NICTA Victoria Laboratories
University of Melbourne, VIC 3010, Australia
tb@ldwin.net, mjoseph@students.csse.unimelb.edu.au

Abstract. This paper explores the use of machine learning techniques to restore punctuation and case in English text, as part of which it investigates the co-dependence of case information and punctuation. We achieve an overall F-score of .619 for the task using a variety of lexical and contextual features, and iterative retagging.

1 Introduction

While digitised text data is growing exponentially in volume, the majority is of low quality. Such text is often produced automatically (e.g. via speech recognition or optical character recognition [OCR]) or in a hurry (e.g. instant messaging or web user forum data), and hence contains noise. Normalisation of case and punctuation in such text can greatly improve its consistency and accessibility to natural language processing methods [10].

This research is focused on the restoration of case and punctuation. To illustrate the task, given the following input text:

(1) ... club course near mount fuji marnie mcguire of new zealand winner of the mitsukoshi cup ladies in april had a ...

we would hope to restore it to:

(1') ... club course near Mount Fuji. Marnie McGuire of New Zealand, winner of the Mitsukoshi Cup Ladies in April, had a ...

There are some notable effects taking place, such as *mcguire*, where the first and third letters are capitalised, and *april*, which is both capitalised and has a comma attached to it. While *cup* and *ladies* would not standardly be capitalised, they require capitalisation in this context because of their occurrence in the proper name *Mitsukoshi Cup Ladies*. Similarly, words such as *prime* and *minister*, and *new* and *york*, need to be capitalised primarily when they co-occur. The above example illustrates the complexities involved in the task of case and punctuation restoration.

This research has applications over the output of speech dictation systems and automatic speech recognition (ASR) systems, which tend to have difficulties predicting where to insert punctuation and sentence boundaries, and also over noisy web data (e.g. as found in web user forums), where case and punctuation are often haphazard [2].

In the process of exploring the complexity of this task and proposing a restoration method, we have developed a benchmark dataset for research on case restoration and punctuation restoration.

2 Related Work

Case and punctuation restoration is a relatively unexplored field. CYBERPUNC [2] is a lightweight method for automatic insertion of intra-sentence punctuation into text. It uses a simple hidden Markov model with trigram probabilities to model the comma restoration problem, restoring the punctuation of 54% of the sentences correctly. [15] tackle the problem of comma restoration using syntactic information, and improve on this to achieve an accuracy of 58%. In both of these cases, sentence boundaries are assumed to be given. In our case, we assume no punctuation whatsoever, including sentence boundaries, and hence direct comparison with our work is not possible.

The above-mentioned methods deal with punctuation restoration at the sentence level, i.e., the input to both systems is a single sentence. For instance, the following input instances:

(2) the golf tournament was at the country club course near mount fuji
(3) marnie mcguire of new zealand winner of the mitsukoshi cup ladies in april had a 72 for 212

would be converted to:

(2') The golf tournament was at the country club course near Mount Fuji.
(3') Marnie Mcguire of New Zealand, winner of the Mitsukoshi Cup Ladies in April, had a 72 for 212.

This simplifies the task significantly, as the sentence boundaries are explicitly specified. This is not the case in our system, where the input is a stream of words, thus requiring the system to detect sentence boundaries (explicitly or implicitly). Hence it is not possible to apply these systems over our data or compare the results directly.

In ASR systems, researchers have made use of prosodic information, disfluencies and overlapping speech to predict punctuation, which they have then supplemented with language models [15].

[10] look into the task of truecasing, or case restoration of text. They propose a language model-based truecaser, which achieves a *word* accuracy of around 98% on news articles. The high accuracy reported here can be used as an indication that the case restoration task is simpler in comparison to punctuation restoration. Note that a direct comparison of the accuracy of punctuation methods

Table 1. Size of the training, development and test datasets

Dataset	Number of tokens
Training	66371
Development	65904
Test	64072

mentioned above and the truecasing task is misleading: the accuracy reported for the punctuation tasks is at the *sentence* level, whereas, in case of the truecasing task it is at the *word* level.

3 Task Description

To generate our dataset, we randomly selected 100 articles each (roughly 65K words) from the AP Newswire (APW) and New York Times (NYT) sections of the English Gigaword Corpus, as training, development and test data. We then tokenised the data, before stripping off all punctuation and converting all the text to lower case. The only punctuation that was left in the text was hyphenation, apostrophes and in-word full stops (e.g. *U.S* and *trade-off*).[1] Each of these Table 1 shows the number of words in each of the datasets.

Each token is treated as a single instance and annotated with a class indicating the punctuation and case restoration that needs to take place, in the form of a capitalisation class, indicating the character indices requiring capitalisation, and a list of zero or more punctuation labels, each representing a punctuation mark to be appended to the end of the word. For example, CAP1+FULLSTOP+COMMA applied to *corp* would restore it to *Corp.,*. The class ALLCAPS is used to represent that all letters in the word need to be converted to uppercase, irrespective of the character length of the word.

Unsurprisingly, the distribution of classes in the data is heavily skewed as detailed in Table 2, with the vast majority of instances belonging to the NOCHANGE class.

In addition to the token instances, we fashioned a set of base features for each word as part of the data release. The base features consist of:

1. the lemma of the word, based on MORPH [13];
2. Penn part-of-speech (POS) tags [12] based on FNTBL 1.0 [14];
3. CLAWS7 POS tags [16] based on the RASP tagger [4]; and
4. CoNLL-style chunk tags based on FNTBL 1.0.

[1] The decision to leave in-word full stops and hyphens in the data is potentially controversial. In future work, we intend to explore their impact on classification performance by experimenting with data which contains literally no punctuation information.

Table 2. The top-8 classes in the training data, with a description of the corresponding change to the token

Class	**Description**	**%**	**Example**
NOCHANGE	no change	75.8%	*really* → *really*
CAP1	capitalise first letter	12.4%	*thursday* → *Thursday*
NOCHANGE+COMMA	append comma	4.1%	*years* → *years,*
NOCHANGE+FULLSTOP	append full stop	3.8%	*settlement* → *settlement.*
CAP1+COMMA	capitalise first letter and append comma	1.7%	*thursday* → *Thursday,*
CAP1+FULLSTOP	capitalise first letter and append full stop	0.9%	*thursday* → *Thursday.*
ALLCAPS	capitalise all letters	0.7%	*tv* → *TV*
ALLCAPS+FULLSTOP	capitalise all letters and append a full stop	0.2%	*u.s* → *U.S.*

We generate all of these features over the case-less, punctuation-less text, meaning that we don't have access to sentence boundaries in our data. For both POS taggers and the full text chunker [1], therefore, we process 5-token sequences, generated by running a sliding window over the text. For a given token, therefore, 5 separate tags are generated for each preprocessor, at 5 discrete positions in the sliding window; all 5 tags are included as features.[2] The total number of base features is thus 16 per instance.

This dataset is available for download at `http://www.csse.unimelb.edu.au/research/lt/resources/casepunct/`.

4 Feature Engineering

While the data release includes a rich array of features, we chose to optimise classifier performance via feature engineering, modifying the feature description in various ways. In all cases, feature engineering was performed over the development data, holding out the test data for final evaluation.

4.1 Lemma and POS/chunk tag normalisation

First, we converted all 4-digit numbers (most commonly years, e.g. *2008*) into a single lemma, and all other sequences of all digits into a second number lemma. Similarly, we converted all month and day of the week references into separate lemmas. The primary reason for this was the high frequency of date strings such as *14 Jan, 2007* which require comma restoration; in this case, the string would be lemmatised into the three tokens *non-4digit-num-ersatz month-ersatz 4digit-num-ersatz*, respectively. In the case of these filters successfully matching with

[2] For tokens at the very start or end of a dataset which do not feature in all 5 positions, we populate any missing features with the value _.

the wordform, the resultant lemma substituted for that in the original dataset. For example, the lemma for *2007* is *2007* in the original dataset, but this would be replaced with *4digit-num-ersatz*. As such, this processing doesn't generate any new features, but simply modifies the existing lemma feature column.

Rather than include all 5 POS and chunk tags for a given token, we select the POS and chunk tags which are generated when the token is at the left extremity in the 5-word sliding window. That is, we remove all but the leftmost POS and chunk tags from the features provided in the dataset. Surprisingly, this simple strategy of taking the first out of the 5 POS and chunk tags provided in the dataset was superior to a range of more motivated disambiguation strategies trialled, and also superior to preserving all 5 tags.

4.2 Lexical features

We capture hyphens, apostrophes and in-word full stops by way of a vector of three Boolean lexical features per instance.

In an attempt to capture the large number of acronyms (e.g. *dvd*) and proper nouns in the data, we fashioned a list of acronyms from GCIDE and WordNet 2.1 [7]. We used the British National Corpus [5] to determine which capitalisation form had the highest frequency for a given lemma. For lemmas where the word form with the highest prior involves capitalisation, we encode the capitalisation schema (e.g. ALLCAPS+FULLSTOP for *u.s.a*) via a fixed set of Boolean features, one each for the different schemas. We additionally encode the conditional probabilities for a given lemma being lower case, all caps, having its first letter capitalised, or having the first and third letters capitalised (e.g. *mcarthur*); these were discretised into three values using the unsupervised equal frequency algorithm algorithm as implemented in NLTK [3].

4.3 Context information

Punctuation and capitalisation are very dependent on context. For example, *prime* is most readily capitalised when to the immediate left of *minister*. To capture context, we include the lemma, Penn POS tag, CLAWS7 POS tag and chunk tag (disambiguated as described in Section 4.1) for the immediately preceding and proceeding words, for each target word. That is, we copy across a sub-vector from the preceding and proceeding words. We also include: (1) bigrams of the target word and proceeding word, in the form of each of word, Penn POS tag and CLAWS7 POS tag bigrams; and (2) trigrams of the Penn and CLAWS7 POS tags of the preceding, target and proceeding words; and (3) trigrams of the CLAWS7 tags of the target word and two preceding words.

5 Classifier Architecture

We experimented with a range of classifier architectures, decomposing the task into different sub-tasks and combining the results differently.

5.1 Class decomposition

We first experimented with a 3-way class decomposition, performing dedicated classification for each of: (1) acronym detection, (2) case restoration, and (3) punctuation restoration. We then take the predictions of the three classifiers for a given instance, and combine them into a fully-specified class. Note that acronym detection cross-cuts both punctuation and case restoration, but is a well-defined standalone sub-task.

For the acronym detection task, we focus exclusively on the three classes of CAP1+FULLSTOP, ALLCAPS+FULLSTOP and NOCHANGE (the three most frequent classes). This was achieved through simple class translation over the training/development instances, by stripping off any extra classes from the original data to form a modified class set.

To perform case restoration, we again strip all but the case information from the class labels. There will inevitably be some overlap with the acronym classifier, so we exclude ALLCAPS+FULLSTOP instances from classification with this classifier (i.e. transform all ALLCAPS+FULLSTOP instances into NOCHANGE instances).

Finally, for the punctuation restoration sub-task, we strip off any case information from class labels, leaving only the punctuation-related labels.

To combine the predictions for the classifiers associated with each of the three sub-tasks, we tested two approaches: (1) using heuristics to directly combine the class labels of the three classifiers, and (2) performing meta-classification across the classifier outputs. In the heuristic approach, the class label produced by the abbreviation sub-task overrides the predictions of the other two classifiers if both predict that case restoration is necessary. For example, if ALLCAPS+FULLSTOP was predicted by the abbreviation classifier and CAP1 was predicted by the case restoration classifier, we would accept ALLCAPS+FULLSTOP as the final case prediction. If the punctuation classifier then predicted COMMA, the final class would be ALLCAPS+FULLSTOP+COMMA. If, on the other hand, the abbreviation classifier predicted NOCHANGE, the prediction from the case restoration classifier would be accepted.

In the meta-classification approach, the three classifiers are run over both the test and development datasets, and the outputs over the development data are used to train a meta-classifier. The outputs from the three classifiers for each test instance are fed into the meta-classifier to generate the final class.

5.2 Retagging

As stated in Section 3, the base features were generated using a sliding window approach (without case or punctuation information). We expect the performance of the preprocessors to improve with correct case and punctuation information, and sentence-tokenised inputs. We thus experiment with a feedback mechanism, where we iteratively: (a) classify the instances, and restore the (training, development and test) text on the basis of the predicted classes; and (b) sentence tokenise, re-tag, lemmatise and chunk the data, and then feed the updated tags

Table 3. Classification results (*italicised* numbers indicate gold-standard data used; **bold** numbers are the best achieved without gold-standard data)

Classifier Description	Accuracy	Precision	Recall	F-score
Baseline (IB1)	.784	.516	.301	.381
Base features, with only first tag	.790	.571	.282	.378
+ extra lexical features	.837	.637	.534	.581
+ all extra features	.839	.620	.604	.611
Heuristic combination	.834	.596	.612	.604
Meta-classifier	**.840**	**.639**	.554	.594
Iterative retagging	**.840**	.615	**.622**	**.619**
Retagging (based on original text)	*.885*	*.715*	*.766*	*.740*
With gold-standard punct labels	*.926*	*.887*	*.793*	*.837*
With gold-standard case/abbrev labels	*.912*	*.793*	*.813*	*.803*

Table 4. Best-10 performing classes for the iterative retagger (ranked based on F-score)

Class	Accuracy	Precision	Recall	F-score
ALLCAPS+FULLSTOP	.657	.787	.799	.793
ALLCAPS	.561	.853	.621	.719
CAP1	.523	.719	.658	.687
CAP1+FULLSTOP	.312	.607	.391	.476
CAP1+COMMA	.276	.523	.369	.433
NOCHANGE+FULLSTOP	.251	.450	.361	.401
NOCHANGE+COMMA	.191	.351	.294	.320
CAP1-3	.143	.750	.150	.250
CAP1+FULLSTOP+COMMA	.100	.667	.105	.182
CAP1+COLON	.082	1.000	.082	.151

back into the data as new feature values. As our sentence tokeniser, we used the NLTK implementation of PUNKT [9]; the lemma, POS and chunk tags are generated in the same way as mentioned in Section 3. We stop the iterative process when the relative change in tags from one iteration to the next becomes sufficiently small (gauged as a proportion of test instances which undergo change in their predicted class).

6 Results

All evaluation was carried out using token-level accuracy, precision, recall and F-score ($\beta = 1$).

As our baseline classifier, we ran the TiMBL implementation of the IB1 learner [6] over the base set of features (which actually outperformed an SVM learner over the same features). All other classifiers are based a multi-class support vector machine, implemented as part of the BSVM toolkit [8].[3] We used a linear kernel in all experiments described in this paper, because we found that it performed better than the Radial Basis Function (RBF) and polynomial kernels.

The results for all the experiments are presented in Table 3.

First, we can see that the strategy of using only the first POS and chunk tag improves accuracy and precision, but actually leads to a drop in recall and F-score over the baseline. The addition of lexical features (Section 4.2) appreciably improved results in all cases, and had the greatest impact of any one of the feature sets described in Section 4. The incorporation of all the extra features brought precision down slightly, but improved recall and led to an overall improvement in F-score.

Using the same set of expanded features with the 3-way task decomposition and either direct heuristic combination or meta-classification, actually led to a slight drop in F-score in both cases relative to the monolithic classification strategy. The meta-classifier generated the highest precision and equal-best accuracy of all the classifiers using only automatically-generated features, but precision dropped considerably.

The retagging method, in combination with the monolithic classifier architecture, resulted in the best accuracy, recall and F-score of all automatic methods tried. The indicated F-score is based on three iterations, as the number of changes dropped exponentially across iterations to only 335 at this point. Error analysis of this final classifier revealed that the performance over case restoration actually deteriorated (to below-baseline levels for the class ALLCAPS+FULLSTOP, e.g.), but the performance over punctuation restoration picks up considerably. Results for the top-10 classes (based on F-score[4]) are presented in Table 4.

To investigate the potential for the retagging method to improve results, we separately ran the lemmatiser, taggers and chunker over the original text (with correct case and punctuation information, and sentence tokenisation), and re-ran the classifier. This caused the F-score to jump up to .740, suggesting that this approach could lead to much greater improvement given higher performance of the base classifier.

Finally, we investigated the co-dependency of the case and punctuation restoration tasks in the context of the meta-classification approach, by combining gold-standard case labels with automatically-generated punctuation labels, and vice versa. This resulted in the final two lines of Table 3, which clearly show that if we can get one of the two tasks correct, the other becomes considerably easier.

7 Future Work

Our research focussed on a small sub-set of punctuation. Punctuation such as question marks and colons was not explored here, and features targeting these could be considered to further improve the performance of the classifier.

[3] `http://mloss.org/software/view/62/`

[4] Excluding the NOCHANGE class.

Another area for future investigation is instance selection [11]. The distribution of instances over the set of classes is skewed heavily in favour of the NOCHANGE class. Instance filtering could have helped in alleviating this bias, forcing the classifier to look at the other classes. This could help especially when looking at the sub-tasks, where the number of NOCHANGE instances increased because of the stripping off of the case or punctuation class information.

The original motivation for this research was in applications such as ASR and OCR, but all of our results are based on the artificially-generated dataset, which lacks case and punctuation but is otherwise clean. We are keen to investigate the applicability of the proposed method to noisy outputs from ASR and OCR in more realistic settings.

8 Conclusion

We have explored the task of case and punctuation restoration over English text. First, we established a benchmark dataset for the task, complete with a base feature set, and then we proposed an expanded feature set, and a range of classifier architectures based on decomposition of the overall task. The best results were achieved with the expanded feature set, a monolithic classifier architecture and iterative retagging of the text.

Acknowledgements

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

1. S. P. Abney. Parsing by chunks. In R. C. Berwick, S. P. Abney, and C. Tenny, editors, *Principle-Based Parsing: Computation and Psycholinguistics*, pages 257–278. Kluwer, Dordrecht, Netherlands, 1991.
2. D. Beeferman, A. Berger, and J. Lafferty. Cyberpunc: A lightweight punctuation annotation system for speech. In *Proceedings of 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'98)*, Seattle, USA, 1998.
3. S. Bird, E. Klein, and E. Loper. *Natural Language Processing with Python — Analyzing Text with the Natural Language Toolkit.* O'Reilly Media, Sebastopol, USA, 2009.
4. E. Briscoe, J. Carroll, and R. Watson. The second release of the RASP system. In *Proceedings of the COLING/ACL 2006 Interactive Poster System*, pages 77–80, Sydney, Australia, 2006.
5. L. Burnard. *User Reference Guide for the British National Corpus.* Technical report, Oxford University Computing Services, 2000.

6. W. Daelemans, J. Zavrel, K. van der Sloot, and A. van den Bosch. *TiMBL: Tilburg Memory Based Learner, version 5.1, Reference Guide.* ILK Technical Report 04-02, 2004.
7. C. Fellbaum. *Wordnet: An Electronic Lexical Database.* MIT Press, Cambridge, USA, 1998.
8. C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science National Taiwan University, 2008.
9. T. Kiss and J. Strunk. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525, 2006.
10. L. V. Lita, A. Ittycheriah, S. Roukos, and N. Kambhatla. tRuEcasIng. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 152–159, Sapporo, Japan, 2003.
11. H. Liu and H. Motoda. *Feature Extraction, Construction and Selection: A Data Mining Perspective.* Kluwer Academic Publishers, Dordrecht, Netherlands, 1988.
12. M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: the Penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.
13. G. Minnen, J. Carroll, and D. Pearce. Applied morphological processing of English. *Natural Language Engineering*, 7(3):207–223, 2001.
14. G. Ngai and R. Florian. Transformation-based learning in the fast lane. In *Proceedings of the 2nd Annual Meeting of the North American Chapter of Association for Computational Linguistics (NAACL2001)*, pages 40–47, Pittsburgh, USA, 2001.
15. S. M. Shieber and X. Tao. Comma restoration using constituency information. In *Proceedings of the 3rd International Conference on Human Language Technology Research and 4th Annual Meeting of the NAACL (HLT-NAACL 2003)*, pages 142–148, Edmonton, Canada, 2003.
16. M. Wynne. *A post-editor's guide to CLAWS7 tagging.* UCREL University of Lancaster, Lancaster, England, 1996.