Comparing machine learning approaches to space, capitalization, and punctuation restoration

Anthony Hughes, Dhwani Shah, Laurence Dyer
University of Wolverhampton
Wolverhampton, United Kingdom
{ A.J.Hughes2, D.R.Shah2, L.J.Dyer } @ wlv.ac.uk

***Abstract*—In this study we implement and evaluate a range of machine learning methods for restoration of spaces, capitalization and punctuation to unformatted English text. We implement a total of five models, including two models for space restoration only, two for capitalization and punctuation restoration, and one end-to-end model that restores all three types of formatting in a single inference step. Some of the models are based closely on existing literature whereas others are our own original implementation. We are not aware of any previous work that has implemented restoration of spaces, capitalization, and punctuation in a single end-to-end BiLSTM model, so to the best of our knowledge Model 5 is a completely novel creation. We combine the partial models to form pipelines and find that a pipeline composed of a Naive Bayes-based model for space restoration and a BiLSTM-based model with pretrained word embeddings for capitalization and punctuation restoration gives the most accurate output for our test dataset.**

Keywords—space restoration, capitalization restoration, punctuation restoration, machine learning, deep learning, Naive Bayes, bidirectional LSTM, continuous random field, word embeddings

I. Introduction

The task of restoring space, punctuation and capitalization is important for automatic speech recognition (ASR) systems. Converting strings of unseparated characters into readable and presentable text is a difficult task. Like every problem, we need to break big problems into smaller bits and tackle them one by one. Recent research on this task based on BiLSTM and GRUs models has been quite successful [1].

In addition, these restoration tasks are not only useful for ASR but also for hashtag segmentation in social media text. A great abundance and variety of data can be found on social media platforms. However, it is well known that we cannot harness text in hashtags with simple pre-processing. Not only are hashtags very popular and can have a lot of information, but people hide hate speech in hashtags.

These two applications were the main motivation behind our model creation. We ultimately decided to focus on data related to speech transcription instead of hashtags from social media.

II. The Data

A. Choice of dataset

We chose a dataset of TED Talks available on Kaggle [2] to train and test our models. Upon researching datasets related to speech transcription, we found two datasets—the TED Talks dataset and the Cornell Movie Dialogs Corpus—that suited our needs. The Cornell Movie dataset contains transcription of movie dialogue from over 600 movies. The data reflects day-to-day conversation but is not generalized and doesn't cover as many topics as the TED Talks dataset.

The TED Talks dataset is also larger than the Cornell corpus. We extracted only the text information from the TED Talks dataset, discarding unnecessary information like title, speaker names, occupation, dates related to each event, etc. We then began cleaning the transcript data.

B. Cleaning the dataset

As part of our work cleaning the data for this project, we developed a Python library of functions to help with cleaning text data in pandas data frames in Google Colab or other notebook-based environments. The library is available at https://github.com/ljdyer/text-data-cleaner.

The library has a function called *show_prohibited_characters* that displays all the unwanted characters in the dataset. By default, these are every character other than alpha-numeric characters, periods and commas. We found many non-English characters along with a wide variety of punctuation symbols. Once we had the list of all the non-essential characters in the corpus, we started working on removing them using regular expressions (RegEx) in Python. We carried out a total of 48 RegEx replacement operations, a small subset of which are presented in Table I.

TABLE I. Examples of RegEx replacements used for data cleaning

RegEx replacement	Description
`('…', "...")`	Convert ellipsis characters to regular full stops
`(r'(([A-Z][a-z]+ )+([A-Z][a-z]+)+:)', '')`	Remove speaker indicators
`(r'([0-9]+):([0-9]+)', r'\1 \2')`	Remove colon from times
`(r'%', r' percent')`	Replace % symbol with word
`(r'(\w)@(\w)', r'\1 at \2')`	Deal with @ symbol used in email addresses

C. Preparing the data

Once we had finished cleaning the data, we saved the results in another csv file. This file has four columns. The first represents the ideal data we would like to get from our models (the 'gold standard'). The second column has been stripped of the punctuation (i.e. periods and commas). The third has been stripped of capitalization. The final column has been stripped of spaces and we are left with sequences of letters.

all_cleaned	`Thank you so much, Chris. And...`
no_punctuation	`Thank you so much Chris And i...`
lower	`thank you so much chris and i...`
no_spaces	`thankyousomuchchrisanditstrul...`

Fig. 1. Example row from the cleaned & prepared dataset

TABLE II. STATISTICS FOR THE TED TALKS DATASET AFTER CLEANING

Number of documents	3,997
Number of sentences	448,783
Number of words	8,138,002
Number of characters	31,336,245

Following cleaning, the documents were randomly shuffled and split into 80% training and 20% test datasets (3,197 and 800 documents respectively). This were saved in separate csv files and the test dataset was not touched during model training—only in the evaluation stage for each model.

III. THE MODELS

A. *Model 1: Naive Bayes-based model for space restoration*

The first model we implemented is a Naive Bayes-based model for space restoration. Our implementation is based closely on the description and Python code in [3]. We initially implemented the model with unigrams only, and later extended it to take bigrams into account. The description below is for the bigram version of the model, with comparison with the unigram model included where relevant. The two versions of the model are compared in the evaluation section.

1) Training: Training the model is very simple. The training data is split at space characters, and the resulting unigrams and bigrams are counted. These counts are then used to calculate probabilities for each unigram/bigram based on the following formula:

$$P(x) = \frac{F(x)}{\sum_{x \in T} F(x)} \quad (1)$$

Where X is either a unigram or bigram and T is the concatenation of all documents in the training dataset.

2) Inference: The model works by first attempting to segment input strings into words (defined here as any continuous sequence of non-space characters, such as 'banana' or '1986'), and then joining the final list of words with space characters to generate the space-restored string. For each new input string, the model calculates probabilities for several candidate words starting from the first character of the input sequence and returns the word with the highest probability.

The unigram version of the model assumes complete independence between words, so that the probability a candidate word is just the product of the probabilities of that word and of all the characters that follow it. In the bigram version of the model, the previous word is considered together with each input, and the probability of each candidate word is calculated as the conditional probability of that word given the previous word w_{prev}, which is:

$$P(w|w_{prev}) = \frac{P(w_{prev}w)}{P(w_{prev})} \quad (2)$$

Stupid backoff is implemented so that the model reverts to unigram probabilities for unseen bigrams. Further, smoothing is implemented so that the model assigns a non-zero probability for unseen unigrams using the following formula to penalize longer candidate words:

$$P(w) = \frac{10}{\sum_{x \in T} F(x) \times 10^{len(w)}} \quad (3)$$

where $\sum_{x \in T} F(x)$ is the total number of unigram tokens in the training dataset. The number 10 in the numerator of equation 3 is arbitrary and could be adjusted to fine-tune the model.

The candidate word selected at each step is the word with the highest probability, that is:

$$argmax_{w \in candidates} P(w) \quad (4)$$

Calculating the probability of every possible segmentation of every input sequence would require an impossibly large number of calculations, so several steps are taken to make the model computationally feasible. First, an upper limit L is placed on the number of characters in each word. We used $L = 20$ for our implementation, but the value of L could be adjusted to fine-tune the balance between accuracy and model performance.

When we began running inference on documents in the test dataset, we discovered that the model could only handle inputs of length up to around 100 characters due to issues with maximum recursion depth. This was resolved by breaking the input string up into smaller chunks and iterating over the chunks, but this led to the further issue that words may be split in the middle. The second issue was resolved by adapting the method used in [4], where the final five words output at each step are concatenated to the input in the next step.

3) Evaluation: Qualititative evaluation after running inference on the test dataset indicated that even the unigram model was very accurate. In fact, there were entire documents in the test dataset for which the model output exactly matched the gold standard. Two types of error were observed in the outputs from the unigram model: those caused by unseen vocabulary (indicated in **red** below), and those caused by lack of contextual information (indicated in **orange** below).

```
on november 5th 1990 a man named
elsayyidnosair walked into a hotel in
manhattan and assassinated rabbimeirkahane
the leader of the jewish defense league
nos air was initially found notguilty of
```

Fig. 2. Sample output from unigram version of Model 1

Adding bigrams to the model significantly reduced the frequency of the second type of error, and also reduced the frequency of the first type of error to some extent:

```
on november 5th 1990 a man named
elsayyidnosair walked into a hotel in
manhattan and assassinated rabbi meir
kahane the leader of the jewish defense
league nos air was initially found not
guilty of
```

Fig. 3. Sample output from bigram version of Model 1

See Table III below for a comparison of the unigram and bigram versions of the model across a range of performance metrics. The accuracy metric for the space restoration step is

Damerau-Levenshtein character error rate over the whole training dataset.

TABLE III. MODEL 1 EVALUATION METRICS

	Unigram version	**Bigram version**
Time to train (s)	2.59	8.41
Pickle size (MB)	2.1	47.5
Inference speed (s/10,000chars)	1.10	**15.59**
Accuracy (%)	99.69	**99.79**

The bigram version of the model is more accurate, but also takes significantly longer to run on new inputs. It is worth noting that inference speed was measured on freshly trained model instances with no memoization cache, so faster inference can be expected if the same instance of the model is reused on multiple inputs.

4) Improving the model: The model is very dependent on training data, so training with more data is likely to lead to a significant improvement in accuracy. Since only unigram and bigram frequencies are required, pre-compiled lists of ngram frequencies such as those available at https://norvig.com/ngrams/ could be leveraged in order to save time spent sourcing enormous volumes of continuous text data.

Providing more contextual information to the model—for example by implementing trigrams—could also improve model accuracy but can be expected to slow down inference even further.

B. Model 2: BiLSTM-based model for space restoration

Following investigation into the various deep learning architectures that have been applied to the space restoration task, we decided to implement a model using bidirectional LSTMs (BiLSTMs). This type of model was chosen because it appears frequently in the literature and well-documented libraries are available to help with implementation.

[5] was used as a basic reference when getting started on the sequence classification task, but the vast majority of the code is our own. After writing the code to generate training sequences, we trained the model on random strings generated from small vocabularies of around 5 words as a proof of concept, and then gradually increased the size of the vocabulary. We also trained a single (unidirectional) LSTM version of the model and observed that performance differences in favor of the BiLSTM architecture could be observed even for small vocabularies.

1) Model design: The task is framed as a binary sequence classification task. For each position in an input sequence of characters x, a 0 or 1 is assigned to indicate whether or not that character in that position is followed by a space, as illustrated in Fig. 4.

```
Original text  to give me an idea of how m
             x togivemeanideaofhowm
             y 01000101010001010010
```

Fig. 4. An example training sequence for Model 2

We used sequences of 100 characters for training, but this choice was arbitrary and could be adjusted. Training samples were generated using a sliding window over the training data with a step size of one word. This generated around 6 million training samples, which were split into 80% training and 20% validation sets.

The model architecture is illustrated in Fig. 5. The forward LSTM layer is fed with the input sequence in its original order, so intuitively it 'remembers' information about characters that came before any given input character. The backward LSTM is fed with the reverse of the input sequence, so it 'remembers' information about characters that will come after an input character. The outputs of the two LSTM layers are concatenated and fed into a sigmoid activation function to generate predictions for the output classes. In evaluation and inference stages, the list of predicted classes is combined with the original input string to generate the space-restored output string.

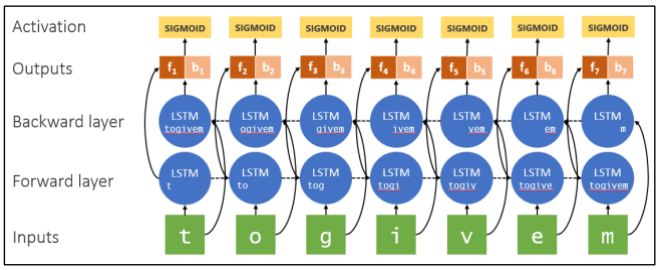

Fig. 5. Model 2 architecture

2) Training: Before training, we carried out two rounds of grid search on a subset (approximately 10%) of the training data to determine optimal hyperparameter values. The hyperparameter values included in each round of the grid search are as indicated in Table 2 below, with the combinations that resulted in the highest test accuracy highlighted in **bold**. The batch size was fixed at 5000. It is thought that a smaller batch size would improve performance, but we ran into errors due to computational resources being overloaded when setting smaller batch sizes. Similarly, it is possible that further increases to the BiLSTM output dimension would benefit model performance, but these were found to be unfeasible from a capacity perspective.

TABLE IV. MODEL 2 GRID SEARCH RESULTS

Hyperparameter	**1st grid search**	**2nd grid search**
Output dimension	{ 25, 50, **100** }	{ 100, **200** }
Dropout	{ **0**, 0.2, 0.4 }	{ 0, **0.1**, 0.2 }
Recurrent dropout	{ 0, **0.2**, 0.4 }	{ 0.1, **0.2**, 0.3 }

Validation loss gains began to stagnate around the 10th epoch, but validation loss continued to decrease past the 40th epoch. The fact that validation loss dropped consistently and was lower than training loss throughout the training process suggests that overfitting did not occur. The decision to stop training after the 41st epoch was arbitrary and was taken as we were still getting used to using GPU resources and due to time pressures to move on to developing and training other models. If we were to train the model again, we would implement early stopping and save the model more frequently so that we could leave it to train for long periods without the need for manual monitoring.

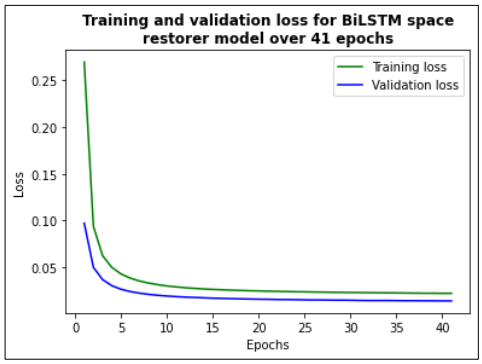

Fig. 6. Model 2 learning curves

We observed high validation and training accuracy after just a few epochs, so we kept a 'short training' version of the model in addition to the 'long training' version in order to understand the relative benefits of training the model for longer. The performance metrics (Table V) show that over 99% accuracy can be obtained in just 43 minutes, but that training the model for much longer led to a significant increase in accuracy.

TABLE V. MODEL 2 EVALUATION METRICS

	Short training	**Long training**
Time to train	43 mins 18 secs (3 epochs)	**11 hrs 32 mins (41 epochs)**
Pickle size (MB)	5.91	5.95
Inference speed (s/10,000chars)	14.23	12.44
Accuracy (%)	99.13	**99.66**

3) Comparison with Model 1: Accuracy for the long training version of Model 2 was almost a high as the unigram version of Model 1, but still significantly lower than the bigram version. It was interesting to see a very straightforward statistical model outperform a sophisticated deep learning architecture with the same training data.

Qualitative analysis of the output from each of the models led to some interesting insights. While errors made by Model 1 can easily be diagnosed as having been caused by either unknown vocabulary or lack of contextual information at the word level, errors made by Model 2 appear more random. This could be seen as a weakness of Model 2 and makes it more challenging to identify ways to improve this model.

On the other hand, Model 2 appeared to cope better with unknown vocabulary. This was confirmed by selecting some samples from Wikipedia with obscure words from different domains and observing that Model 2 inserted spaces in the correct positions—while Model 1 failed to do so—in every case (see Fig. 7).

Model 1 (Naive Bayes)	**Model 2 (BiLSTM)**
…largest **rum in ant** on earth…	…largest **ruminant** on earth…
…distinguished from **canoe ing** by…	…is distinguished from **canoeing** by…
…position of the **paddle r** and…	…position of the **paddler** and…

Fig. 7. Comparison of outputs of Model 1 and 2 for obscure vocabulary

This behaviour is unsurprising when we consider that Model 1 works at the word level and Model 2 works at the character level. Model 2 implicitly learns about the structure of words themselves, and can identify common prefixes and suffixes to judge how likely it is that a string of unseen characters makes up a word, whereas Model 1 has no means to be able to make such a judgement. While no evaluation with other datasets was carried out in this study, the observations above lead us to speculate that Model 2 would be likely to outperform Model 1 on data from different domains, and may be more capable of learning in situations where there is less training data available.

4) Improving the model: Given more time to improve Model 2, we would attempt to identify patterns in the errors produced to help inform the next steps. It is possible that improvements could be made in the training data creation stage. Perhaps changing the sequence length would impact the model performance. Also, it may be better to use padding to generate samples of equal length, rather than cutting words off in the middle. We would test out any changes on restricted vocabularies to assess their effect before taking the time to retrain the entire dataset.

C. *Model 3: CRF-based model for capitalization and punctuation restoration*

The literature review into statistical techniques for space restoration led to the discovery of the use of Conditional Random Fields (CRF). The decision to implement a model using CRF was based on existing literature [6]. This multi-class classification model was chosen because it is well established in NLP tasks. A novel implementation was built based upon the documentation for the python-crfsuite library [7]. Being able to assign multiple classes to each word within a given sequence is useful for this task, where we are attempting to restore both capitalization and punctuation in a single inference step.

1) Model design: The task is framed as a multi-class sequence classification task. Each set of features x corresponding to a single word in the input sequence has 1 or more assigned restoration classes y (Table VI) as illustrated in Fig. 8.

TABLE VI. MODEL 3 TARGET CLASSES

Class	**Label**
u	Uppercase
l	Lowercase
t	Titlecase
c	Comma
p	Period
n	No punctuation

Original text	`To …`
x	`['bias', 'word=to', 'pos_tag=TO', 'SOD', '+1:word=give', '+1:pos_tag=VB', '+2:word=me', '+2:pos_tag=PRP']`
Y	`['tn']`

Fig. 8. Example feature input for the start of a sequence (Model 3)

The features used to train the model were the words themselves and part of speech (POS) tags. The current word and its POS tag were fed to the model as part of each feature set. If available, the previous and next two words and their POS tags were also included. In the situation where the word is at the start or end of a document, a special **SOD** or **EOD** feature was added. See Fig. 8 as an example.

2) Training: Once the data was ready training was simple. The python-crfsuite library has a training mechanism that takes in all training data and stores the model as a Python serialized object (pickle). The CRF parameters used are presented in Table VII.

TABLE VII. MODEL 3 CRF PARAMETERS

Parameter	Value
C1	1.0
C2	1e-3
Max iterations	50
Possible transitions	True

The C values are used to regularize the model and the training process is halted after 50 iterations. 'Possible transitions' is a feature of the CRF suite library that generates transitions between x and y pairs that are not available in the training set. The model took around 11 minutes to train on a CPU and the size of the model was 10MB.

3) Evaluation: An anecdotal qualititative evaluation was undertaken on the model to observe how well the test dataset was performing.

Fig. 9. identifies situations where the model has inserted punctuation in the middle of a named entity, and where a named entity has not been capitalized. Although issues arose with named entities for these documents the word error rate was below 4% when compared to the original transcripts.

Gold Standard	Model 3 (CRF)
…a man named **El Sayyid Nosair** walked…	…a man named **El Sayyid. Nosair** walked …
A **cold January** day…	A **cold. January Day**…
…assassinated **Rabbi Meir Kahane**…	…assassinated **rabbi meir kahane**…

Fig. 9. Comparison of Model 3 output with gold standard text

Since the qualitative evaluation was promising, a systematic evaluation was undertaken to test the model. Precision and recall were used to test the model.

TABLE VIII. MODEL 3 CLASSIFICATION REPORT

Class	Precision (%)	Recall (%)	F1 (%)
Uppercase	93	98	30
Lowercase	46	22	96
Titlecase	58	36	95
Comma	62	30	44
Period	100	82	40
No punctuation	92	98	90

A macro F1 score of 75% suggests that the model is highly accurate. When accounting for the weighted average the model scores 89%. This suggests that the model is highly accurate when accounting for the contribution of each class in the model.

D. Model 4: BiLSTM-based model for capitalization and punctuation restoration

The literature contains a large amount of research using deep learning architectures that have been applied to NLP tasks. Research has demonstrated that neural network models used in conjunction with transformer-based architectures can yield highly accurate models [8].

[8] was used as the basis for this sequence classification task. The authors had fine-tuned multiple transformer-based language models, such as BERT and RoBERTa, to classify words in a sequence as having either question marks, commas, or full stops. The authors' model was altered so that the classification of capitalization was included, and question marks removed.

1) Model design: The task is framed as a sequence classification task. For each position in an input sequence of words x, a single class y (Table IX) is assigned to indicate whether or not that word requires punctuation and/or capitalization, as illustrated in Fig. 10.

TABLE IX. MODEL 4 TARGET CLASSES

Class	Label
0	No restoration
1	Comma
2	Full stop
3	Capitalization
4	Capitalization + Full stop
5	Capitalization

Original text	To	give	me	an	idea,
x	to	give	me	an	idea
y	3	0	0	0	1

Fig. 10. Sample training example for Model 4.

Samples were generated from the source data. This generated around 8 million training samples and 1 million samples in both validation and test sets.

The high-level model architecture is illustrated in Fig. 11. A word from a BERT tokenized sequence is passed to each node in the pretrained layer. A hidden layer is then fed into the forward LSTM layer. The pretrained hidden layer is also attached to the backward LSTM. The outputs of the two LSTM layers are concatenated and fed to a fully connected layer with six nodes, each representing a target class. A softmax activation function is used to generate predictions for each of the output classes. In evaluation and inference stages the argmax function is used to return the class with the highest predicted probability.

2) Training: A single pretrained transformer, bert-based-uncased, was implemented for this task. A sequence length of 256 was used as defined by the model-specific tokenizer. Start and end sequence tokens are also applied. Where a sequence falls short of the length padding is applied. A batch size of 8 was used. Adam was the optimisation function and a learning rate of 1e-5 was applied.

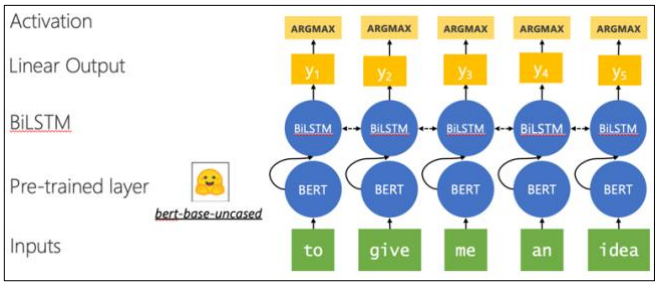

Fig. 11. Model 4 architecture

Training time took around 35 mins per epoch and the validation performance was tracked at each completed epoch (see Table X).

TABLE X. MODEL 4 VALIDATION PERFORMANCE

Epoch	Val. loss	Val. accuracy
1	0.207	0.83
2	0.185	0.86
3	0.179	0.88
…	...	…
9	0.1622	0.91
10	0.16	0.91

3) Evaluation: A systematic evaluation was undertaken to test the model. Precision and recall were the metrics used to test the model (see Table XI).

TABLE XI. MODEL 4 CLASSIFICATION REPORT

Class	Precision (%)	Recall (%)	F1 (%)
No restoration	97	98	98
Comma	75	66	71
Full stop	81	82	81
Capitalization	86	84	85
Capitalization + full stop	78	76	77
Capitalization + comma	74	66	70

A macro F1 score of 80% suggests that the model is highly accurate. When accounting for the weighted average the model scores 93%. This suggests that the model is highly accurate when accounting for the contribution of each class in the model.

4) Further work: The current accuracy rates are the result of using a single pretrained model. The first reommended experiment is to use a larger language model such as RoBERTa. RoBERTa [9] offers significant performance improvements over BERT and achieves state-of-the-art results on benchmarks such as GLUE and SQuAD. A cased version of BERT [10] should also be investigated. This version of BERT may contain better representations of cased words and therefore may improve the accuracy of this classification task.

E. Model 5: BiLSTM-based end-to-end model

Having already trained models that could accurately restore spaces and capitalisation/punctuation separately, we developed Model 5 as an experiment to see whether all of this formatting could be restored using a single model. We observed that Model 2 could be extended to complete all of the restoration tasks by capturing information about capitalisation and punctuation in the classes assigned to each character. The implementation is entirely our own, and we are unaware of any previous research that has used character-level BiLSTMs in this way.

1) Model design: The main difference between this model and Model 2 is the classes assigned to each character. In Model 2, only two classes were required because the only information that needed to be captured was the presence or absence of a space after each character, but Model 5 needed multiple classes to capture the various different types of formatting and combinations of them.

With more types of formatting to consider than for previous models, in order simplify the labelling process and avoid any confusion or ambiguity in naming (does *c* stand for *capital* or *comma*? Does *n* signify *no space* or *no punctuation*?), we came up with a simple system using prime numbers. The choice of prime numbers was motivated by the fundamental theorem of arithmetic [11, p. 799], which states that every number can be written as a product of prime factors and implies that it will always be possible to retrieve the prime numbers that were multiplied to produce a number (which in this case correspond to our various formatting features, as we shall see) without any risk of ambiguity.

The first 4 prime numbers (2, 3, 5 and 7) were assigned to the properties of being capitalised, having a full stop, having a comma and having a space respectively. For example, the *U* in the text "in the U.S., and to make bett…" was assigned the class label 6 (2 × 3) because it is upper case and has a full stop after it. The *S* in the same sentence was assigned the class label 210 (2 × 3 × 5 × 7).

Input: A gold standard text sample with spaces, capitals, and punctuation (e.g. "in the U.S., and to make bett…"). This will always begin with an upper or lowercase letter.

Initialize an empty list *letters*
Initialize an empty list *classes*

For each character *c* in the text sample:

If *c* == '.':
Multiply the last element of *classes* by 3

If *c* == ',':
Multiply the last element of *classes* by 5

If *c* == ' ':
Multiply the last element of *classes* by 7

If *c* is a letter:
Append *c* to *letters*
If *c* is lowercase, append 1 to *classes*
If c is uppercase, append 2 to *classes*

Fig. 12. Pseudocode for algorithm to generate Model 5 training samples

The task of generating training samples (parallel lists of letter characters and class labels) was slightly more complex that for the previous models because the number of characters in the gold standard input text corresponding to a single letter/class pair in the sequence could be one or many (e.g. `s` or `S.,`), and there was no single character that indicated the start of the next letter (so the *split* function would not work).

The algorithm in Fig. 12 was designed to assign classes in a simple and computationally efficient manner. We added an unknown class to the list of classes after generating training samples to account for any combinations of punctuation that may not have been present during training, resulting in a total of 14 classes: {1, 2, 3, 5, 6, 7, 70, 14, 210, 21, 35, 105, 42, '<UNK>'}.

We decided to limit the number of periods, commas, or full stops following each letter to no more than one (by placing an extra *if* statement inside each of the *if* statements in the algorithm to check that the last class was not already divisible by the prime number before multiplying) to keep the number of classes to a minimum, but this restriction could be lifted in situations where multiple instances of a punctuation character or (some other property) can follow a letter, such as to allow for multiple exclamation marks. The class labels do not capture information about the order of the features, so an alternative method would have to be used in situations where this is important.

When restoring formatting to input strings based on the classes predicted by the model, each type of formatting was restored to a letter if its predicted class was divisible by the number assigned to that type of formatting. Formatting was restored in the order capital → full stop → comma → space, which is the only order generally permitted by the rules of English punctuation.

The architecture of the bidirectional LSTM layer is identical to Model 2. The dimensionality of the output layer is the number of classes (14 in this instance), and softmax activation is used in place of sigmoid to accommodate multiple classes. Categorical cross entropy is used in place of binary cross entropy for the loss function.

2) Training: In order to establish the feasibility of the model design before beginning to prepare for training on the full dataset, we ran some initial tests using random strings generated from small vocabularies (10-100 words). We then carried out grid search on a subset (approximately 10%) of the training data to determine optimal hyperparameter values. Perhaps due to the additional complexity introduced by having more than 2 classes, we found that setting output dimensionality greater than 100 led to errors, so this was fixed at 100 in addition to batch size being fixed at 5000 as for Model 2. The parameters included in the grid search were dropout and recurrent dropout, and combinations of values in the set {0, 0.1, 0.2} were tested. The optimal hyperparameters for this model were found to be *dropout=0.2* and *recur_dropout=0*.

The learning curves for this model (shown in Fig. 13) were very similar to those for Model 2 but learning was generally slower, with significant drops in validation loss observed past the 20th epoch and small but steady gains continuing as we approached 100 epochs. Training was stopped following around 24 hours of training (96 epochs). The decision to stop training was based on manual monitoring and motivated to a large extent by restrictions on computational resources and the time pressures on the authors. If we were to train the model again we would decide on conditions for stopping in advance and implement these in the code.

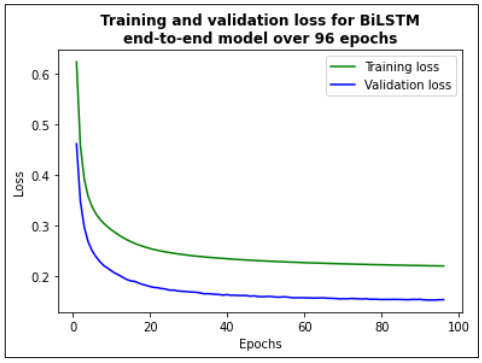

Fig. 13. Model 5 learning curves

3) Evaluation: Qualitative observation of the output from the model indicates that the model succeeds to a large extent in appropriately restoring all of the types of formatting, although the output is far from perfect.

```
on November 5th 1990, a man Na medel
say yid no sair walked into a hotel in
manhattan and assassina ted
Rabbimeirkahane the leader of the
Jewish defense league no sair was
initially found not guilty of
```

Fig. 14. Model 5 sample output

Since this model performs the same task and the pipelines models composed of combinations of the previous models, quantitative evaluation of the model was carried out at the same time and using the same metric as those models.

IV. Final Evaluation and Comparison of Models

By joining models which work on space restoration with models that work on punctuation and capitalization restoration we have four pipeline models, in addition to which we have one end-to-end model. Each model on its own has accuracy of at least 80% based on the evaluation metrics used for each stage in the process. But the final goal is to restore everything, so we need to calculate the accuracy after we have created the pipeline models. We also needed to evaluate these models on a single metric to enable like-for-like comparison.

We chose word error rate (WER) as the metric for the final evaluation and comparison of our models. WER is the standard metric used to evaluate the output of ASR systems, which is fitting given that our goal was to restore spaces, capitalization, and punctuation to clean up the output of ASR systems.

Word error rate (WER) is based on how many words in the output of the restorer model (the hypothesized word string) differ from those in a reference transcription. The first step in computing WER is to compute the minimum edit distance in words between the hypothesis and reference strings, giving us the minimum numbers of word substitutions (S), word insertions (I), and word deletions (D) necessary to map between the reference and hypothesis strings. The WER is then calculated using the following formula:

$$WER = \frac{I+S+D}{len(reference)} \quad (5)$$

As we are aiming to present the accuracy of model rather than the error itself, we subtract the error from 1 and multiply by 100.

$$accuracy = (1 - WER) \times 100 \quad (6)$$

The lists of words in the reference and hypothesis texts are generated by splitting at space characters, so incorrect word splits are flagged as errors, as is failure to restore punctuation or capitalization correctly because, for example, the string 'Manhattan' is not the same as 'manhattan', and 'US' is not the same as 'U.S.'.

We were unable to encounter any convenient ready-made libraries for calculating WER for our data, so we adapted code from https://github.com/ljdyer/wer-calculator, which was created as part of LD's work for the 7LN004: Computational Linguistics module and is based on the description of the Levenshtein algorithm in [12, pp. 74-77]. We used this method to measure the accuracy for each pipeline/end-to-end model, which is shown in Table XII. The first model in each pipeline is for space restoration and the second is for capitalization and punctuation restoration. The end-to-end model scored lowest. The highest-performing model overall is the Naive Bayes and BiLSTM pipeline model. This shows that traditional statistical models can perform well alongside state-of-the-art deep learning models such as BiLSTM.

TABLE XII. FINAL EVALUATION RESULTS

Pipeline/model	Accuracy (%)
Model 1 (Naive Bayes) + Model 3 (CRF)	81.6
Model 1 (Naive Bayes) + Model 4 (BiLSTM)	91.4
Model 2 (BiLSTM) + Model 3 (CRF)	80.5
Model 2 (BiLSTM) + Model 4 (BiLSTM)	90.2
Model 5 (end-to-end BiLSTM)	75.8

V. PUTTING THE MODELS INTO PRODUCTION

In order to demonstrate that our models could be deployed to a production environment and to make the results of our work available to the wider public, we developed APIs for each of the models using Microsoft Azure Functions and uploaded a website to act as a portal for those APIs. We invite readers to visit https://multi-restore.netlify.app/spaces.html to try our models out for themselves with any text of their choice. At the time of writing, Models 1, 2, 3, and 5 (as well as pipelines composed of combinations of those models) are available, and we are currently working on making Model 4 available.

INDIVIDUAL CONTRIBUTIONS

This project was a truly collaborative effort, with every member of the group actively involved in every part of the study. The authors met up in person at intervals of around once per week throughout the period that we were working on the project to share findings, results and feedback, and to come to a group decision on how to proceed with the next phase of work. The following is a broad summary of what each group member was mainly focused on in each phase of the project:

In Phase 1, each team member carried out research into existing approaches to the problem and available datasets.

In Phase 2, AH focused on performance metrics, and DS and LD focused on cleaning and preparing the dataset.

In Phase 3, AH focused on approaches with word embeddings, DH focused on deep learning approaches, and LD focused on statistical machine learning approaches.

In Phase 4, AH focused on statistical machine learning approaches, DH focused on approaches with word embeddings, and LD focused on deep learning approaches.

In Phase 5, all team members focused on final evaluation results for each of the models and on preparation of the presentation. For the presentation, each team member was allocated sections based on the areas they had accumulated the most knowledge about.

In Phase 6, all team members worked on the preparation of this report. Sections assigned were the same as for the presentation.

REFERENCES

[1] J. Sivakumar, J. Muga, F. Spadavecchia, D. White and B. Can, "A GRU-based pipeline approach for word-sentence segmentation and punctuation restoration in English," In 2021 International Conference on Asian Language Processing (IALP), 2021, pp. 268-273.

[2] V. Gupta, "TED Talk: Complete Metadata and Transcript of TED.com as of 24-JUN-20", Jun. 24, 2020. [Online]. Available: https://www.kaggle.com/datasets/thegupta/ted-talk. [Accessed: May 14, 2022].

[3] P. Norvig, "Natural language corpus data," in *Beautiful Data*, T. Segaran and J. Hammerbacher, Eds. Sebastopol: O'Reilly, 2009, pp. 219-242.

[4] G. Jenks, "python-wordsegment," July, 2018. [Online]. Available: https://github.com/grantjenks/python-wordsegment. [Accessed May 2, 2022].

[5] J. Brownlee, "How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras," machinelearningmastery.com, June 16, 2022. [Online]. Available: https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/. [Accessed May 2, 2022].

[6] M. Lui and L. Wang. "Recovering casing and punctuation using conditional random fields", In Proceedings of the Australasian Language Technology Association Workshop, 2013, pp. 137-141.

[7] "python-crfsuite". Mar. 14, 20220. [Online]. Available: https://python-crfsuite.readthedocs.io/en/latest/. [Accessed May 12, 2022].

[8] T. Alam, A. Khan and F. Alam, "Punctuation restoration using transformer models for high-and low-resource languages", In Proceedings of the Sixth Workshop on Noisy User-generated Text, 2020, pp. 132-142.

[9] J. Devlin, W. C. Ming, K. Lee and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," arXiv: 1908.08962v2 [cs.CL], 2019.

[10] Y. Liu, O. Myle, N. Goyal, M. M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer and V. Stoyanov. "RoBERTa: a robustly optimized BERT pretraining approach," arXiv:1907.11692 [cs.CL], 2019.

[11] G. H. Hardy, "An introduction to the theory of numbers", *Bulletin of the American Mathematical Society*, vol. 35, no. 6, pp. 778-818, 1929.

[12] D. Jurafsky and J. H. Martin, *Speech and language processing*. 2nd ed., New Jersey: Prentice Hall, 2008.