Machine Learning for Named Entity Recognition and Classification Report

Yunchong Huang

Abstract

This report examines Named Entity Recognition and Classification (NERC) using classic machine learning algorithms with feature engineering, alongside a fine-tuned BERT model. We highlight the importance of feature engineering for improving classic systems by experiments and feature ablation, while demonstrating that SVMs with pre-trained embeddings achieve competitive performance along with error analysis. We also conclude that he fine-tuned BERT, leveraging the modern Transformer structure, achieves satisfactory performance.

1 Introduction

This report focuses on the Named Entity Recognition and Classification (NERC) task utilizing several machine learning methods. We first examine the data set and propose reasonable evaluation metrics for such a task. Then we experiment with classic algorithms of Logistic Regression (LogReg), Naive Bayes (NB) and Support Vector Machine (SVM) for the classifier design, to which various feature engineering measures are implemented to improve the performance. Additionally, we utilized a fine-tuned BERT (HuggingFace, 2023) for the same task. We conclude that extensive feature engineering is vital for improving the performance of classic machine learning systems, and the SVM system combining handcrafted features and pre-trained word embeddings with hyperparameters tuned has the best performance among classic algorithms. The fine-tuned BERT, on the other hand, provides a satisfactory performance.

2 Task and Data

2.1 Task

The experiments focus on the NERC task on structured English textual data based on the Reuters Corpora stored in `.conll` annotation files, taken from Tjong Kim Sang and De Meulder (2003). On the operational level, this is equivalent to a classification task mapping word tokens in the structured textual data to different named entity tags with Beginning-Inside-Outside (BIO) annotations. The classification is implemented using machine learning methods with input features encoded from annotated structural data or loaded from pre-trained word embeddings. The named entity tags as classification targets are listed in Table 1 with descriptions.

Category	Description
B-ORG	the beginning of an organizational named entity
I-ORG	being inside of an organizational named entity
B-PER	the beginning of a personal named entity
I-PER	being inside a personal named entity
B-LOC	the beginning of a locational named entity
I-LOC	being inside of a locational named entity
B-MISC	the beginning of a named entity that is not a location, organization or person
I-MISC	being inside of a named entity that is not a location, organization or person

Table 1: The categories of named entity.

2.2 Dataset and Distribution

The task-specific structured data includes tokenized texts and annotations of their corresponding Part-of-Speech (PoS) tags, chunk tags and named entity tags. Both chunk tags and named entity tags are annotated based on the Beginning-Inside-Outsie (BIO) annotation scheme. The whole data is split into three parts for training, development and testing and stored in three `.conll` files respectively.

The training dataset. The training dataset contains 203,621 annotated tokens. The specific distribution of named entity tags in the training data by named entity tags is shown in Table 2. The majority of training instances are with the “O" tag, taking up a percentage of 83.30%, whereas tags of “I-LOC" and “I-MISC" are significantly rare, both taking up percentages lower than 1%. Other tags have proportions ranging from 1.80% to 3.50%.

The development dataset. The development dataset contains 51,362 annotated tokens. The distribution by named entity categories in the development data is shown in Table 2. The development dataset shows a distribution similar to the training dataset, with instances of the “O" category occupying the majority while instances of “I-LOC" and “I-MISC" being the rarest.

The test dataset. The test dataset contains 46,435 annotated tokens. The distribution by named entity categories in the test data is shown in Table 2. The test dataset still shows a distribution similar to both the training dataset and the development set, with instances of the “O" category occupying the majority while instances of “I-LOC" and “I-MISC" being the rarest.

	Training Data		Development Data		Test Data	
	Number	Percentage	Number	Percentage	Number	Percentage
B-LOC	7140	3.50%	1837	3.58%	1668	3.59%
B-MISC	3438	1.70%	922	1.80%	702	1.51%
B-ORG	6321	3.10%	1341	2.61%	1661	2.61%
B-PER	6600	3.24%	1842	3.59%	1617	3.58%
I-LOC	1157	0.57%	257	0.50%	257	0.55%
I-MISC	1155	0.57%	346	0.67%	216	0.47%
I-ORG	3704	1.80%	751	1.46%	835	1.80%
I-PER	4528	2.22%	307	2.54%	1156	2.49%
O	169578	83.30%	42579	83.25%	38323	82.53%

Table 2: Data distribution across training, development, and test datasets

2.3 Preprocessing

As already mentioned, the structural textual data used for the current task are already tokenized and annotated with PoS tags and chunk tags. Preprocessing measures common in other tasks such as lower-casing, stemming and removing punctuations are not taken, as they can bring unwanted information loss for the task due to linguistic traits of named entity expressions in English.

On the other hand, a preprocessing pipeline for feature extraction is designed to add feature columns to the original data files. These columns are extracted as dictionary vectorizer inputs for the machine learning systems. The detailed choices of these features are introduced in 3.2.

2.4 Evaluation Metrics

Standard token-level evaluation system. Directly inspecting the counts of token-level matching and mismatching between predicted named entity tags and gold named entity tags. Precision, recall and F1 scores are calculated for each named entity tag, along with their macro average and weighted average. Since the distribution of named entity tags is highly unbalanced and leans heavily toward the “O" tag (non-entity), we mainly focus on the macro average scores to evaluate the overall performance, reducing the bias brought by the “O" tag.

Span-based evaluation system. Named entity spans, represented as tuples of start index, end index, and entity type (e.g., "PER", "ORG"), are extracted from annotated data. Gold and predicted spans are compared to compute exact matches as the intersection of their sets, given the uniqueness of encoded span tuples. Partial matches are calculated as the overlap proportions between gold and predicted spans, ignoring the entity type. Final scores of precision, recall and F1 are derived by summing exact match counts and partial match proportions, normalized by the total number of predicted or gold spans. This method focuses on the overall detection of named entities as complete spans, accounting for both complete and partial recognition success.

3 Models and Features

3.1 Models

Logistic Regression. A machine learning model for classification tasks by using the Logistic (Sigmoid) function to map linear combinations of input features to probabilities, enabling prediction of class labels. In this task, two logistic regression models are trained: LogReg-basic with only one-hot features directly available in the structured data (word tokens, PoS tags and syntactic parsing tags); LogReg-1hot with additional engineered one-hot features (to be specified in 3.2). Generally, one-hot features are encapsulated into dictionaries for vectorization, and class labels are named entity tags. The maximum iteration is set to 1000 in both models to avoid convergence failure.

Multinomial Naive Bayes. A machine learning model for classification tasks that assumes that features follow a multinomial distribution and conditional independence between feature values. It calculates the probability of each class based on Bayes’ theorem, making predictions by selecting the class with the highest posterior probability $P(\text{class}|\text{features})$. The Multinomial version is chosen because it can take (non-binary) discrete features in the dictionary form as input. One-hot input features are encapsulated as dictionaries for vectorization and class labels are named entity tags in this task.

Support Vector Machine (SVM). A machine learning model for classification tasks with the

mechanism to find the optimal hyperplane that best separates different classes by maximizing the margin between data points and the hyperplane to ensure accurate predictions. For experiments of the current task, two SVM classifiers are trained: SVM-1hot uses only one-hot features encapsulated into dictionaries as inputs, while SVM-wmb uses a concatenated feature matrix formed by pre-trained word embeddings and one-hot features as inputs.

Fine-tuned BERT. BERT is an encoder-based Transformer language model pre-trained on large-scale data through masked token prediction and next-sentence prediction tasks. Utilizing a bidirectional self-attention mechanism, it captures rich general linguistic knowledge during pre-training. With fine-tuning based on a small amount of labelled data, BERT can achieve strong performance on specific downstream tasks. For the current NERC task, which is a token-level classification problem, we leverage the *AutoModelForTokenClassification* class from HuggingFace's Transformers library to implement a 3-round fine-tuning on BERT with different seeds.

3.2 Features

Word tokens. Word tokens included in named entity expressions themselves are generally differentiable from other ordinary expressions, thus they can directly be encoded as discrete-value entries in feature dictionaries for the current task.

PoS tags. Most tokens annotated as parts of named entities have nominal PoS tags (around 90% in both training and development data), revealing that certain PoS tags are more likely to be correlated with named entity expressions. Thus information on PoS tags can be directly extracted and encoded as discrete-value entries in feature dictionaries for the current task.

Chunk tags. The majority of tokens annotated as named entity tokens are syntactically within noun phrases (NP) (96% - 97% in both training and development data) with chunk tags of "B-NP" or "I-NP", revealing that certain chunk tags are more likely to correlate with named entity expressions. Thus information on chunk tags can be directly extracted and encoded as discrete-value entries in input feature dictionaries for the current task.

Orthographic information of the current token. Most tokens annotated as named entity tokens contain at least one upper-case letter (97% - 98% in both training and development data). Thus the orthographic information of a token can be indicative of its membership of a named entity. This information can be encoded as binary-value entries (i.e. "Whether the current token contains an uppercase letter) in input feature dictionaries.

Affixes of word tokens. Prefixes and suffixes of a word token potentially indicate a named entity's existence. For instance, a prefix like "Mr.-" is strongly correlated to a person entity; while suffixes like "-land" and "-burg" are strongly correlated to a location entity. Following the practice in Tkachenko and Simanovsky 2012, the maximum length of affix extraction is set to 6 in the current task, and they are encoded as two discrete-value entries (i.e. prefix and suffix) in the input feature dictionaries.

Orthographic information of neighbouring tokens (o_{n-1} and o_{n+1}). Tkachenko and Simanovsky 2012 mentioned that extracting orthographic information of neighbouring tokens is also helpful to improve the NERC task performance. Thus "whether the previous/following token contains upper case" can be additionally encoded as two binary-value entries in the feature dictionaries.

Neighbouring tokens (t_{n-1} and t_{n+1}). Since most expressions of named entities are composed of more than one token, directly introducing neighbouring tokens of the current token as contextual information to serve as features are also potentially helpful for a model to judge whether the current token is part of a named entity. Thus for each token, its previous token (t_{n-1}) and next token (t_{n+1}) are extracted and encoded as two string-value entries in the input feature dictionaries.

Pre-trained word embeddings. An alternative for encoding word tokens and neighbouring tokens as features is to utilize pre-trained word embeddings. Pre-trained word embeddings are dense word vectors with 100–300 dimensions trained on large-scale corpora using neural networks. These embeddings capture rich semantic information about words based on their contextual relationships, often leveraging tasks like predicting surrounding words or classifying words in a given context. For the current task, we make use of 300-dimension Google word embeddings ("GoogleNews-vectors-negative300") in the word2vec format to form al-

ternative feature representations for “word tokens" and “neighbouring tokens". Then these word embeddings are combined with other one-hot features introduced above through matrix concatenation (horizontal stack), forming a feature matrix as the input for the SVM-wmb model.

4 Experiments and Results

4.1 Hyperparameter Tuning

A hyperparameter tuning was carried out on the SVM-wmb model by implementing the exhaustive grid search and 3-fold validation over the training data. This is motivated by the fact that the size of training data is bigger than the development data, while cross-fold validation can guarantee the diversity of data used for evaluations and avoid potential overfitting compared to simply using the development data. The grid search space is defined by the range of C (the regularization parameter) value [0.1, 1] and the range of maximal iterations [5000, 10000][1]. The result shows that $C = 0.1, \text{max\_iter} = 5000$ leads to the best performance on standard tag-wise macro F1-score (0.858). We further verified this by comparing the development data with the tuned and default settings. The tuned setting indeed led to a slightly better overall performance (tag-wise macro F1: 0.880, span-based macro F1: 0.943) than that of the default setting (label-wise macro F1: 0.879, span-based macro F1: 0.942). We then applied the tuned setting to the SVM-wmb model on the test data experiment.

4.2 Evaluation

Trained models are first experimented on the development set. Results show that the LogReg-basic model using only self-contained features directly available in the data has a significant performance gap compared to other models with more extensive feature engineering. Please refer to Appendix A for tag-wise and span-based macro average scores shown in Table 8, along with detailed tag-wise standard evaluations for these models (except for LogReg-basic)[2] provided in Table 9.

[1]The grid search space is rather limited due to the limitation of my current device’s computation power.

[2]LogReg-basic is excluded due to its significantly worse performance and the space limitation

	Standard Macro Average			Span Macro Average		
	Precision	Recall	F1-score	Precision	Recall	F1-score
LogReg-1hot	0.806	0.782	0.791	0.911	0.873	0.894
Multinomial NB-1hot	0.803	0.687	0.720	0.837	0.813	0.825
SVM-1hot	0.797	0.802	0.799	0.915	0.895	0.905
SVM-wmb	0.821	0.829	0.824	0.918	0.921	0.920

Table 4: Comparison of different models on the test data using two macro average metrics)

The overall performance of experiments on the testing data is shown in Table 4. The LogReg-basic model is excluded as its performance has already proven to be obviously unsatisfactory in the development stage. Among the remaining models, SVM-wmb has the best performance, with all figures in both standard and span-based metrics scoring the highest, but its advantage over LogReg-1hot and SVM-1hot is not significant. The only model with a clearly worse performance is Multinomial NB-1hot. Detailed tag-wise standard evaluations on the token level for these models are provided in Table 3.

Additionally, the performance results from fine-tuned BERT on the development data with three different seeds are provided in Table 11 in Appendix A. The overall performance remains consistent with changes of seed (average F1-score between 0.946-0.947), which is significantly better than traditional machine learning methods from the perspective of standard metrics and slightly better than their span-based performances on the numeric level. But further efforts confirming the computational methods of the `SeqEval` library should be made to finalize this comparison.[3]

4.3 Feature Ablation

A feature ablation analysis is implemented on SVM-1hot based on the development data with the feature combinations shown in Table 5. The choice is motivated by the fact that SVM-1hot is the second-best performing model and its feature ablation is more straightforward to implement than SVM-wmb.

Only a subset of possible feature combinations is selected for the ablation analysis, as an exhaustive analysis would require training models on $2^{10} - 1$ possible feature combinations and many of them are not meaningful. The choice of this subset is motivated by the need to show how much performance improvement a single feature can bring with key incremental comparisons, while different sets of features can also be compared on the effectiveness of the performance improvement. Specific

[3]I didn’t find detailed documentation of this metric on the internet.

	LogReg-1hot			Multinomial NB-1hot			SVM-1hot			SVM-wmb		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
B-LOC	0.827	0.835	0.831	0.706	0.872	0.780	0.829	0.868	0.848	0.843	0.901	0.871
B-MISC	0.822	0.752	0.786	0.833	0.697	0.759	0.809	0.785	0.797	0.794	0.802	0.798
B-ORG	0.827	0.713	0.766	0.781	0.650	0.709	0.839	0.723	0.777	0.843	0.765	0.802
B-PER	0.812	0.848	0.830	0.822	0.742	0.780	0.821	0.845	0.833	0.894	0.922	0.908
I-LOC	0.823	0.634	0.716	0.915	0.253	**0.396**	0.727	0.716	0.722	0.734	0.708	0.721
I-MISC	0.731	0.653	0.689	0.706	0.532	0.607	0.645	0.648	**0.648**	0.647	0.667	**0.646**
I-ORG	0.616	0.684	**0.648**	0.642	0.641	0.641	0.664	0.707	0.684	0.734	0.741	0.738
I-PER	0.804	0.933	**0.863**	0.845	0.810	**0.827**	0.847	0.942	**0.892**	0.926	0.959	**0.942**
O	0.987	0.987	0.987	0.973	0.985	0.979	0.989	0.988	0.988	0.993	0.991	0.992

Table 3: Tag-wise standard token-level evaluations of feature-engineered models on the test set

	Token	PoS tag	Chunk tag	Ortho	Prev ortho	Next ortho	Prev token	Next token	Prefix	Suffix
f_0	✓	✓	✓							
f_1	✓	✓	✓	✓						
f_2	✓	✓	✓	✓				✓		
f_3	✓	✓	✓	✓	✓			✓	✓	
f_4	✓	✓	✓	✓	✓	✓		✓	✓	
f_5	✓	✓	✓	✓	✓	✓		✓	✓	✓
f_6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
f_7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
f_8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
f_9	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 5: Feature combinations selected for the ablation analysis (f_0 to f_9)

insights drawn from this ablation analysis (results shown in Figure 1 and Table 10) can be summarized as follows: 1) Involving basic information PoS tags, chunk tags and orthographical information of the current token as features can already improve performance significantly than using only tokens themselves (comparing f_0-f_1 and f_0-f_2); 2) 6-digit prefixes and suffixes of tokens are not features as valuable as expected, which is shown in pairwise comparisons of f_2-f_3, f_4-f_5 and f_6-f_7; 3) Involving neighbouring tokens as features can significantly improve the performance, shown in pairs of f_2-f_4, f_3-f_5 and f_7-f_8; 4) Involving orthographic information of neighbouring tokens as features can also improve the performance, shown by comparing f_2-f_6 and f_3-f_7, but it's not effective as involving neighbouring tokens themselves (shown in pairs of f_2-f_4 and f_3-f_5); 5) Involving all 10 features (f_9) extracted leads to the best performance, with both tag-wise metrics and span-based metrics reaching a macro average F1 above 0.85.

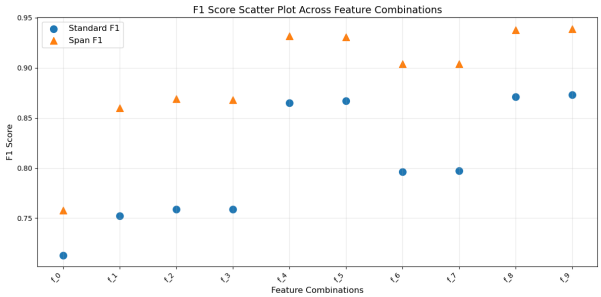


Figure 1: The tag-wise and span-based macro average F1 scores of chosen feature combinations for the ablation analysis

5 Error Analysis

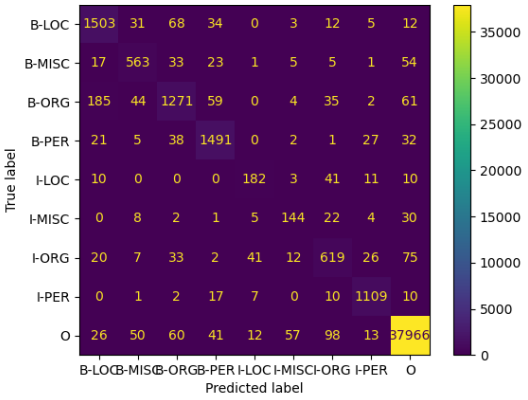


Figure 2: The macro average F1 scores of chosen feature combinations for the ablation analysis

An error analysis is carried out on the SVM-wmb model on the test data, which is with the best overall performance among classic machine learning models.

Table 3 from the standard evaluation metrics shows that besides the "O" tag, "I-PER" and "B-PER" are the two tokens with highest F1-scores (0.942 and 0.908), followed by "B-LOC" (0.871) and "B-ORG" (0.802). F1-scores of "B-MISC" (0.798), "I-ORG"(0.738) and "I-LOC" (0.721) are less satisfactory, while "I-MISC" scores the lowest (0.646). Several observations can be drawn from this: 1) The SVM-wmb model excels at recognizing "PER" entities; 2) For other types of entities, the SVM-wmb model is better at recognizing the beginning of the named entity than other parts fol-

lowing; 3) The SVM-wmb model performs the worst on MISC entities.

The confusion matrix shown in Figure 2 provides more insights into specific types of errors made by the SVM-wmb model. We report the most frequent confused pairs in the form of "(gold, prediction)" starting from each entity tag as gold (except for "O"): (B-LOC,B-ORG), (B-MISC, O), (B-ORG, B-LOC), (B-PER,B-ORG), (I-LOC,I-ORG), (I-MISC,O), (I-ORG, O), (I-PER, B-PER). It can be observed that the confusion between entity types is the most common error among these pairs (5/8), followed by the failure of recognition (misjudged as "O"). The errors of the boundary are less witnessed, except for the (I-PER, B-PER) pair.

The confusion between "ORG" and "LOC" is rather outstanding since the three most commonly confused pairs are related to it: (B-LOC,B-ORG), (B-ORG, B-LOC), (I-LOC,I-ORG). Therefore, we further investigate confusion between entity types of "LOC" and "ORG" in both directions. We first extract all error instances with "LOC" and "ORG" as gold from the output prediction file, discarding boundary markings of "I" and "O". Then we retrieve the descriptive statistics of the confusion types, summarized in Table 6:

ORG			LOC		
Confusion Pair	Number	Percentage	Confusion Pair	Number	Percentage
ORG-LOC	246	**45.73%**	LOC-ORG	121	**52.6%**
ORG-O	136	25.28%	LOC-PER	50	21.74%
ORG-PER	89	16.54%	LOC-MISC	37	16.09%
ORG-MISC	67	12.45%	LOC-O	22	9.57%
Total	538	**100%**	**Total**	230	**100%**

Table 6: Confusion pair statistics for ORG and LOC entity types

The statistics proved that confusion between "ORG" and "LOC" is indeed the most common error in both cases where the gold is "ORG" and "LOC" respectively. A potential reason is that it's common for organization names to include geographical word tokens, and the model's contextual sensitivity to differentiate them is not sufficient.

Beyond this, we also carry out a high-level error analysis on the relationship between whether capitalization is involved in word tokens and the detection of named entities. The entity types and the I/O boundary distinction are discarded, and the "detection" is defined by assigning any non-"O" tag to named entity tokens. We compare the overall detection rate, the detection rate when entity tokens contain capitalization and the detection rate when entity tokens contain no capitalization:

	Total Number	**Detected Number**	**Detection Rate**
No Capitalization	107	55	**51.4%**
With Capitalization	8005	7773	97.1%
Total	**8112**	**7828**	**96.5%**

Table 7: Detection rates for cases with and without capitalization

From the results above, we can observe that when capitalized characters are involved in the entity tokens, the classifier has a detection rate above 95%. But when entity tokens are without capitalized characters, the classifier can only detect around half of them. This reveals that the classifier depends heavily on the capitalization feature and has a limited detecting capacity for minority cases without it.

6 Discussion

Performance summaries of development and test data show that the SVM-wmb model achieves the highest tag-wise and span-based macro F1 scores among classic machine learning models, highlighting the benefits of combining handcrafted feature engineering with pre-trained word embeddings. However, error analysis reveals weaknesses in differentiating similar entity types and a detection bias toward major features. While feature ablation analysis should ideally be performed on SVM-wmb for consistency, time constraints led to its implementation on SVM-1hot, still offering insights into the role of handcrafted features. Additionally, a fine-tuned BERT model achieved satisfactory results, and further comparisons with neural models (e.g., LSTM) could enhance the analysis.

7 Conclusion

Based on experiments, we can conclude that manual feature engineering is vital in the design of classic machine learning systems. For the specific NERC task, explicitly encoding PoS tags, chunk tags, orthographical information and contextual tokens brings particularly significant performance improvement. However, the best model among classic machine learning systems still suffers from the confusion between entity types with similar features and the insensitivity towards minority instances without a "mainstream" feature representation. Last but not least, a fine-tuned BERT model was implemented for the task and obtained satisfactory results.

References

HuggingFace. 2023. Fine-tuning a model on a token classification task.

Heeyoung Lee, Angel Chang, Yves Peirsman, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. 2013. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational Linguistics*, 39(4):885–916.

Preslav Nakov, Sara Rosenthal, Zornitsa Kozareva, Veselin Stoyanov, Alan Ritter, and Theresa Wilson. 2013. SemEval-2013 task 2: Sentiment analysis in Twitter. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 312–320, Atlanta, Georgia, USA. Association for Computational Linguistics.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Maksim Tkachenko and Andrey Simanovsky. 2012. Named entity recognition: Exploring features. In *KONVENS*, volume 292, pages 118–127.

Theoretical Component

Assignment 1

Machine Learning is a process in which a machine is not explicitly programmed to implement a task based on a set of pre-defined rules, but implicitly learns patterns from experience with instances/examples provided for it to improve its performance on this task.

Assignment 2

Sentiment Analysis in Twitter

The discussion of this part is based on Nakov et al. (2013).

1.A.1 Subtask B is defined as using pre-processed structural data acquired from tweet messages as input. The basic pre-processing can include tokenization, PoS tagging, syntactic parsing and orthological lowercasing. The desired output is sentiment labels of "positive", "negative" or "neutral" attached to each complete unit of Tweet message.

1.A.2 **Words**: The word tokens themselves. They reflect the choice of words that can be indicative of expressions of a certain sentiment.
Neighbouring words: A specified number of neighbouring tokens around the target token. They can help to grasp the contextual information, especially degree adverbs and negative words that can help to identify sentiments more accurately.
Punctuations: Punctuation marks in the language. Some punctuations can be indicative of certain sentiment labels. For example, exclamation marks and question marks are clearly indicative of non-neutral sentiments.
PoS tags: The word class markings of word tokens. They can help to differentiate different uses of the same word token form and help the system focus on the ones that are genuinely useful for sentiment analysis. For example, it can differentiate "good" as an adjective and "good" as a noun, and the sentiment analysis should only focus on the former.
Shape of words: The specific shapes that word tokens are in (e.g. upper/lower case). Specific shapes of word tokens appearing in the text can be informative for sentiment analysis. For instance, the frequent use of capitalized letters is a signal for strong and non-neutral sentiment.
Sentiment lexicon (dictionary): An external lexicon resource mapping from certain tokens to certain sentiment labels. For tokens included in this lexicon, the mappings can be used as features to set up a clear relationship between tokens and the sentiment label.
Emoticons(emojis): Non-linguistic symbols supplementary to messages on social media. They provide important non-verbal clues for sentiment analysis, as they are mainly used as supplementary tools to mark the sentiment of a written expression. For example, emojis related to "angry" are directly related to the negative sentiment.

1.A.3 **Words**: If sticking to the dictionary method introduced in Assignment 1, they can be encoded as a discrete-value entry for the feature dictionary, transformed from text tokens to the main body of the input feature vectors.
Neighbouring words: Use projective mapping in the data to retrieve neighbouring words/tokens and add them discrete-value entries to feature dictionaries, then finally transform them as a concatenated part of input vectors.
Shape of words: Before the lowercase operation during the preprocessing, use condition checking to decide if a token contains no capital letter / 1 capital letter / 2 capital letters. Map the three cases into 3 binary-value entries in the feature dictionaries, then transform them as a concatenated part of input vectors.
PoS tags: Add token-wise PoS tags (retrieved from structural data or a PoS tagger) as a discrete-value entry in feature dictionaries, then transform them as a concatenated part of input vectors. There is also an alternative to combine the string of PoS tags and word token strings and use their combination as a discrete-value entry for feature dictionaries. Then the machine learning system can learn to distinguish allomorphs (e.g. good-ADJ, good-NN).
Punctuations: Use a tokenizer that doesn't remove punctuation marks and preferably treat them as independent tokens. Then they can be directly encoded as a discrete-value entry for the feature dictionary and concatenated into the vectors.
Sentiment lexicon (dictionary): For each word token that occurs in the training data,

map it into the sentiment lexicon and see if it has an entry there; meanwhile in the feature dictionary of each token, set up a discrete-value entry with the value of sentiment label in the lexicon (if the token is included), then transform them as a concatenated part of input vectors. When a differentiation between the general sentiment label and the domain-specific sentiment label is introduced for some tokens, the mapping should guarantee the domain-specific label is prioritized when this domain is relevant to the current task (e.g. using rules based on logical conditions).

Emoticons (emojis): Use an emoji processing module to transform emojis in the text as Unicode representations, and then encode them directly as discrete-value entries for feature dictionaries and transform them into the concatenated vectors. **Tweet representation**: Each token's feature dictionary can be aggregated across the entire tweet. For example, numerical features such as sentiment scores can be averaged or summed to create tweet-level numerical features. Similarly, categorical features like PoS tags can be represented by their distributions across the tweet (e.g., the proportion of nouns, verbs, or adjectives).

Coreference Resolution

The discussion of this part is based on Lee et al. (2013).

1.B.1 The general task definition of coreference resolution is identifying and collecting expressions referring to the same entity in a textual discourse. On the level of system design, this task preferably takes pre-processed structured textual data as input. Specifically, the pre-processing can include tokenization, PoS tagging, sentence splitting, syntactic parsing and named entity recognition (mention detection). The desired output should be a complete list of lists/sets/tuples containing expressions (mentions) referring to the same entity.

1.B.2 **Features for speaker identification:** implementing rules of speaker identification would require extracting features of *textual strings of recognized named entities/detected mentions* (raw materials for rule-based comparisons), *whether the sentence has a reporting verb* (as an important standard to activate speaker identification), *whether the sentence has quotation marks* (as an important standard to activate speaker identification for data directly quoting conversations), *the subject of the reporting verb* (one potential co-referent party of coreference ready for rule-based comparison), *the pronouns involved in the sentence* (the other potential co-referent party of coreference ready for rule-based comparison), *whether two pronouns are within a pair of quotation marks* (an important basis for pronoun disambiguation from speaker's relative perspective).

Features for exact match and relaxed string match: *textual strings of recognized named entities/detected mentions* are core features serving as materials for the matching of strings among detected entities/mentions. Specifically for the sieve of relaxed string match, it's also vital to retrieve the *syntactic headwords* of detected entities/mentions.

Features for precise constructs: 1) *Syntactic parsing tags (chunk tags)* and *PoS tags* of tokens are necessary features to grasp the (NP, NP) structure of appositives and detect coreferences from them. 2) *Whether a sentence has a copulative structure* would be an important binary feature for identifying coreferences with predictive nominative. 3) *Textual strings of recognized named entities/detected mentions*, the *named entity labels* of detected entities/mentions *syntactic headwords* of detected entities/mentions, the *animacy status* of detected entities/mentions and the *gender* of detected entities/mentions are all necessary features to implement the rules proposed for role appositive. For example, a named entity expression of location should not be of coreference with a personal entity/pronoun in most cases. 4) *Whether a detected entity/mention is a relative pronoun* is the core feature for the relative-pronoun rule. 5) *Textual strings of recognized named entities/detected mentions* and *PoS tags* of tokens are important for rules of acronym coreference detection.

Features for head match and its variants (sieves 5-9): *syntactic headwords* of detected entities/mentions, *textual strings of recognized named entities/detected mentions* and *Syntactic parsing tags (chunk tags)* are necessary for a variety of rule-based comparisons

concerning entity head match, word inclusion, compatible modifiers and not i-within-i under the category of head match and its variants. For example, if a coreferent relationship is to be set up between two expressions of "the Prime Minister of the United Kingdom" and "the Prime Minister", the effective feature extraction of *syntactic headwords* is really important.

Features for pronominal coreference resolution: Apart from two features of *gender*, *animacy*, *named entity tags* of recognized named entities/detected mentions already mentioned above, features of *number*, *person* and *the distance between pronouns* from them are also valuable for checking grammatical and semantic agreements to retrieve coreferences more precisely. For example, the mention of the name of an individual government official should not be coreferent with the expression "the officials".

1.B.3 ***Syntactic headwords*** **of detected entities/mentions**: It's possible to extract syntactic headwords from detected entities/mentions in two ways. Firstly, due to the syntactic characteristics of English, we can assume that the heads of noun phrases (the phrase type that named entities are mostly of) are mostly located at the final position. With this assumption, based on syntactic parsing tags (chunk tags), we can retrieve the last word token of every chunk labelled as NP and regard them as heads. But one should be aware that this phrase-final assumption is not always true and it may bring many errors. Secondly, based on both syntactic parsing tags (chunk tags) and token-wise PoS tags, efforts can be made to retrieve the word token with a nominal PoS tag in noun phrase chunks and collect them as heads. But this is also problematic in cases where there is more than one noun in a noun phrase chunk. For instance, some nouns can be used as adjective components in a noun phrase construction, but their PoS tags are still nominal and they would be wrongly collected as heads under this method. Either way, the extracted headwords can be encoded as a discrete-value entry for a feature dictionary serving as the basis of the vectorizer.

The *gender* feature of detected entities/mentions: The construction of the *gender* feature of detected entities/mentions may rely on the combination of some rules based on cultural traditions (e.g. "if a person name ends with a consonant, then it's a name for a male.") and the mapping based on an external lexicon as the basis of one-hot encoding in feature dictionaries. Since this feature is not strictly hard-wired in linguistic forms, there may be exceptions that cannot be captured by extraction rules based on cultural traditions (e.g. some traditional male names are adapted by females). Additionally, the external lexicon may not be exhaustive and its quality has a great influence on the performance of the coreference detection task. Therefore, reliable external resources defining male and female names are important for the extraction of this feature. Once the gender feature is determined, it can be encoded as a discrete-value entry for a feature dictionary serving as the basis of the vectorizer.

Assignment 3

The KNN algorithm

(1) The algorithm behind the CNN classifier is a type of so-called "lazy learning algorithm", which means that it doesn't have an independent training procedure but makes predictions based on known data directly. The whole working process of KNN classification can be summarized as follows: 1) Collecting some instances with their class already known; 2) Given a new data instance with the class unknown, calculate its distance from every other instance with a known class; 3) Set up a k value, rank the instances with class known by their distances to the newly given instance from the closest to the farthest, then select k closest known instances; 4) Predict the unknown instance to belong to the class that appears most frequently among the k closest known instances.

(2) As mentioned above, for the classification problem, the unknown instance is predicted by KNN to belong to the class that appears most frequently among the k closest known instances, thus the choice of k can affect the prediction differently. In this example in Figure 1, if $k = 3$ as the figure shows, then the prediction would be a green triangle. If the value k is increased to the extent where a larger number

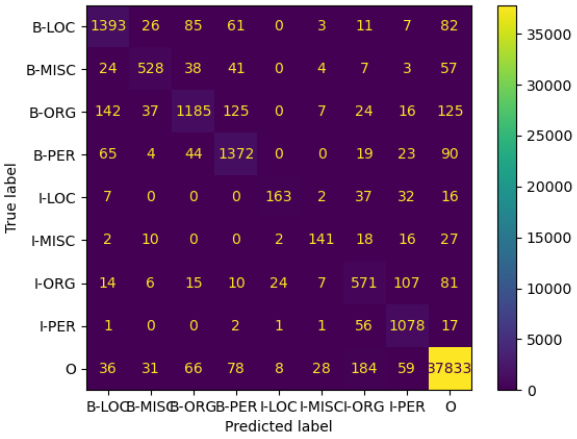
of yellow square instances are included, the prediction may change to the yellow square if the number of its instances suppresses the number of green triangle instances.

(3) If the k value is set to be very small, then the classification of a new data point can be heavily sensitive to one or two non-representative peculiar data points nearby, which would lead the classifier to predict its class to be the same as the peculiar data point instead of the class of the majority of data points around. If the k value is set to be extremely large, then the classification of a new data point would be heavily dependent on the global class distribution. This results in a failure to reveal the "local" characteristic of the new data point, which essentially ignores the specific traits of it when making a prediction.

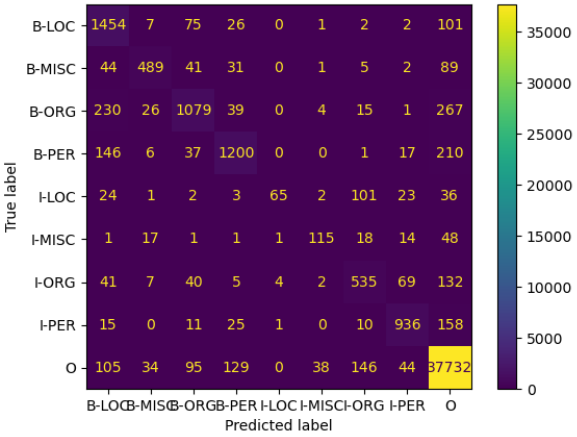
A Appendix

This is an appendix. You can include additional analyses and tables here (e.g. samples you analyzed for your error analysis).

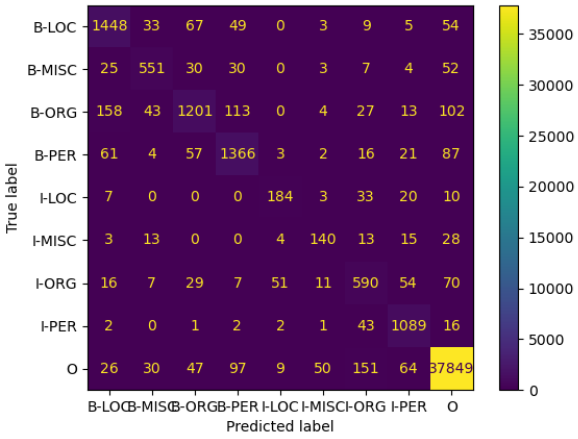
A.1 The confusion matrix of the standard token-level evaluation on the test data - LogReg-1hot



A.2 The confusion matrix of the standard token-level evaluation on the test data - Multinomial NB-1hot



A.3 The confusion matrix of the standard token-level evaluation on the test data - SVM-1hot



A.4 The performance summary of different models on the development data

A.5 Tag-wise standard token-level evaluations on the development set

A.6 The detailed performance data of the feature ablation analysis

A.7 The performance of fine-tuned BERT with various seeds

	Standard Macro Average			Span Macro Average		
	Precision	Recall	F1-score	Precision	Recall	F1-score
LogReg-basic	0.791	0.702	0.730	0.895	0.781	0.834
LogReg-1hot	0.896	0.835	0.861	0.944	0.915	0.929
Multi NB-1hot	0.866	0.728	0.773	0.875	0.861	0.869
SVM-1hot	0.894	0.856	0.873	0.949	0.929	0.939
SVM-wmb	0.899	0.864	0.880	0.946	0.939	0.943

Table 8: Comparison of different models on the development data using two macro average metrics

	LogReg-1hot			Multinomial NB-1hot			SVM-1hot			SVM-wmb		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
B-LOC	0.883	0.880	0.881	0.769	0.893	0.826	0.896	0.911	0.903	0.916	0.911	0.914
B-MISC	0.899	0.810	0.852	0.885	0.755	0.815	0.907	0.838	0.871	0.898	0.857	0.877
B-ORG	0.874	0.797	0.834	0.825	0.758	0.790	0.887	0.811	0.847	0.879	0.852	0.865
B-PER	0.884	0.901	0.892	0.894	0.823	0.857	0.900	0.914	0.907	0.930	0.952	0.941
I-LOC	0.920	0.763	0.834	0.901	0.354	**0.508**	0.872	0.821	0.846	0.874	0.786	0.828
I-MISC	0.910	0.671	**0.772**	0.933	0.442	0.678	0.869	0.688	**0.768**	0.821	0.691	**0.750**
I-ORG	0.814	0.740	0.775	0.708	0.651	0.678	0.818	0.768	0.793	0.816	0.762	0.778
I-PER	0.886	0.953	**0.919**	0.905	0.836	**0.893**	0.905	0.959	**0.931**	0.963	0.967	**0.965**
O	0.990	0.996	0.993	0.981	0.993	0.987	0.992	0.997	0.995	0.993	0.997	0.995

Table 9: Tag-wise standard token-level evaluations of feature-engineered models on the development set

	Standard Macro Average			Span Macro Average		
	Precision	Recall	F1-score	Precision	Recall	F1-score
f_0	0.806	0.650	0.713	0.838	0.691	0.758
f_1	0.792	0.734	0.752	0.893	0.829	0.860
f_2	0.796	0.740	0.759	0.898	0.841	0.869
f_3	0.795	0.741	0.759	0.896	0.841	0.868
f_4	0.889	0.846	0.865	0.941	0.923	0.932
f_5	0.891	0.848	0.867	0.943	0.920	0.931
f_6	0.825	0.776	0.796	0.913	0.894	0.904
f_7	0.826	0.777	0.797	0.916	0.892	0.904
f_8	0.893	0.853	0.871	0.948	0.927	0.938
f_9	0.894	0.856	**0.873**	0.949	0.929	**0.939**

Table 10: Performance metrics for different ablation configurations (f_0 to f_9) using macro average evaluations

	Seed=0			Seed=21			Seed=42		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
LOC	0.956	0.971	0.963	0.958	0.967	0.962	0.958	0.966	0.961
MISC	0.861	0.861	0.861	0.861	0.861	0.861	0.867	0.857	0.862
ORG	0.913	0.939	0.926	0.913	0.939	0.926	0.918	0.935	0.927
PER	0.980	0.982	0.981	0.980	0.982	0.980	0.978	0.985	0.982
overall avg	0.941	0.951	0.946	0.941	0.951	0.946	0.943	0.950	**0.947**

Table 11: Fine-tuned BERT performances under different seeds

NERC-research preparation

Finding out the Basics

1(1) Categories used are the following:
B-ORG: the beginning of an organizational named entity.
I-ORG: being inside of an organizational named entity.
B-PER: the beginning of a personal named entity.
I-PER: being inside a personal named entity.
B-LOC: the beginning of a locational named entity.
I-LOC: being inside of a locational named entity.
B-MISC: the beginning of a named entity that is not a location, organization or person.
I-MISC: being inside of a named entity that is not a location, organization or person.

1(2) The annotation files follow the Computational Natural Language Learning (CoNLL) format. Every token is on a separate line followed by TAB-separated columns with annotations. The annotations include Part-of-Speech (PoS) tagging (column 2), syntactical parsing (column 3) and named entity marking (column 4). Annotations of PoS tagging use the Penn Treebank P.O.S tags. Annotations of both syntactic parsing and named entity marking use the Beginning-Inside-Outside (BIO) style.

2(1) Each line represents a single-word token.

2(2) They are empty lines for separation. They are used between the title and author information, author information and place and date, the place and date and the main body texts, and most importantly, between paragraphs of the main body texts.

2(3) The word token itself - its PoS tag - the syntactic category in which it is situated- its named entity status.

Set-up Evaluation

3(1) The precision is the proportion of true positives predicted by a system on a certain label in all instances predicted to be positive on a certain label by this system; the recall is the proportion of true positives on a certain label predicted by a system in all instances that are indeed positive on this label; the F-score is a comprehensive performance evaluating measure considering both precision and recall.

$$\text{Precision} = \frac{\text{true positive}}{\text{true positive}+\text{false positive}}$$

$$\text{Recall} = \frac{\text{true positive}}{\text{true positive}+\text{false negative}}$$

$$F_1 = 2 \cdot \frac{\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}}$$

A scenario where precision is more important: In a NERC system for legal documents, high precision is more important because false positives unchecked can lead to serious legal consequences, such as wrongly attributing an action to the wrong person or organization. False negatives are also not ideal, but it's easier to be perceived and legal professionals can still manually identify the overlooked entities during review.

A scenario where recall is more important: In a medical NERC system designed to extract mentions of diseases, drugs, or symptoms from clinical notes, high recall is more important. Missing critical medical entities (false negatives) could cause incomplete diagnoses or treatment plans, potentially endangering patients' lives. False positives, such as retrieving irrelevant terms, are comparatively tolerable since they are easier to observe and can be filtered out in post-processing or by medical professionals.