SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

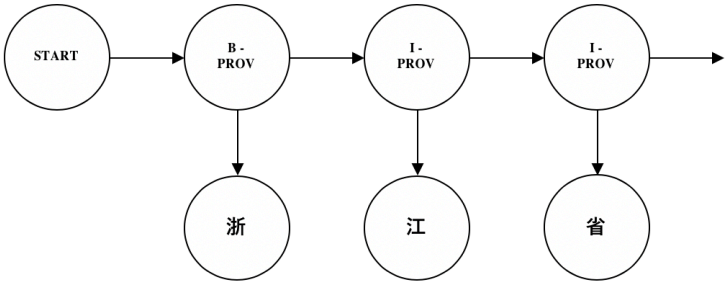
# 01.112: Machine Learning Project

Timothy Liu Kaihui (1002653), See Wan Yi Faith (1002851), Zhang Ruihan (1003784)

10 December 2019

**Abstract**

In this report, we detail our 01.112: Machine Learning project approaches and results where we implemented a sequence labelling system using a hidden markov model (HMM). We train the system on the provided training data before we used the system to predict tag sequences for new sentences on the development and test sets. In addition, we attempt various strategies to improve the performance of the HMM.



# Contents

# 1 Code and Output Files

Our code and raw output files (`dev.out` for the various datasets) can be found on this GitHub Repository.

Our output from the test data can be found in `data/TEST/EN` and `data/TEST/AL` folders as `test.out`, and can be found by running `part_5_en_test_set.ipynb` and `part_5_al_test_set.ipynb` respectively.

# 2 Part 2: Emission Parameters

## 2.1 Running the Code

To run our code, run each cell in the file, `part_2.ipynb` after changing the `dataset_folder` directory to the respective directories in the second cell.

## 2.2 Our Approach

Our approach is detailed as follows:

1. We estimated the emission parameters from the training set using Maximum Likelihood Estimation (MLE). This function can be found in `emission.py`.
2. We smoothed the emission parameters by creating a modified training set by replacing words that appear less than 3 times in the training set with a special word token `##UNK##` before training. This significantly reduces the number of words in the modified training set. This function can be found in `utils.py`.
3. We implemented a simple sentiment analysis system and then proceeded to learn the parameters for each of our dataset before evaluating them using the evaluation script, `evalResult.py` to obtain our results detailed below.

## 2.3 Results

Comparing our outputs and the gold-standard outputs in `dev.out` for the respective datasets by running our `evalResult.py` to calculate precision, recall and F score, our results on the four datasets are as follows:

### 2.3.1 Entity

| Dataset | Precision | Recall | F Score |
|---|---|---|---|
| **EN** | 0.4955 | 0.7049 | 0.5819 |
| **AL** | 0.1467 | 0.3403 | 0.2050 |
| **CN** | 0.0794 | 0.2896 | 0.1246 |
| **SG** | 0.1764 | 0.3734 | 0.2396 |

### 2.3.2 Sentiment

| Dataset | Precision | Recall | F Score |
|---|---|---|---|
| **EN** | 0.4296 | 0.6112 | 0.5046 |
| **AL** | 0.1158 | 0.2688 | 0.1619 |
| **CN** | 0.0473 | 0.1725 | 0.0742 |
| **SG** | 0.1003 | 0.2123 | 0.1362 |

# 3 Part 3: Transition Parameters

## 3.1 Running the Code

To run our code, run each cell in the file, `part_3.ipynb` after changing the `dataset_folder` directory to the respective directories in the second cell.

## 3.2 Our Approach

Our approach is detailed as follows:

1. We estimated the transition parameters from the training set using Maximum Likelihood Estimation (MLE). This function can be found in `transition.py`.
2. For all datasets, we learned the model parameters with `train`.
3. Using the estimated transition and emission parameters, we implemented the Viterbi algorithm for each of our dataset by running it on the development set, `dev.in` using the learned models, before evaluating the output in `dev.p3.out` using the evaluation script, `evalResult.py` to obtain our results detailed below.

### 3.2.1 Dealing with Numerical Underflow

To eliminate any numerical underflow or other unanticipated numerical effects (such as math domain errors), we used a number of strategies:

- When multiplying probabilities, we did it via a summation of the `log` of all probabilities.
- Scale the value of `pi` by a large number (we empirically used a `1e4` which eliminates some edge cases where underflow still occurred).
- Add a small value epsilon ($\varepsilon$ `= 1e-100`) to values before `log` to avoid math domain since `log(0)` is not defined.

## 3.3 Results

Running the Viterbi algorithm on the development set `dev.in` using the learned models for the respective datasets with our output in `dev.p3.out`:

### 3.3.1 Entity

| Dataset | Precision | Recall | F Score |
|---|---|---|---|
| **EN** | 0.8489 | 0.8410 | 0.8449 |
| **AL** | 0.6146 | 0.6927 | 0.6513 |
| **CN** | 0.4288 | 0.2057 | 0.2780 |
| **SG** | 0.5439 | 0.3540 | 0.4288 |

### 3.3.2 Sentiment

| Dataset | Precision | Recall | F Score |
|---|---|---|---|
| **EN** | 0.8166 | 0.8089 | 0.8128 |
| **AL** | 0.5249 | 0.5916 | 0.5563 |
| **CN** | 0.2934 | 0.1407 | 0.1902 |
| **SG** | 0.3386 | 0.2204 | 0.2670 |

# 4 Part 4: Our Algorithm

## 4.1 Running the Code

To run our code, run each cell in the file, `part_4.ipynb` after changing the `dataset_folder` directory to the respective directories in the second cell.

## 4.2 Our Approach

Using the estimated transition and emission parameters, the following details our algorithm to find the $k$-th best output sequences, where $k = 7$. Instead of only taking $\arg\max_v\{\pi(j, v) \cdot a_{v,STOP}\}$ as per the original Viterbi algorithm, we modify the original Viterbi algorithm by adding all the entries to a pool, sorting the pool and then truncating it to $k$ entries where we can subsequently find the $k$-th best output sequences.

### 4.2.1 General Approach

1. Perform a slightly modified version of the Viterbi algorithm and append all possible output sequences and their probabilities to a pool, instead of just selecting the best sequence as per the original Viterbi algorithm
2. All the entries in the pool are sorted by probability.
3. With $n$ representing the number of words in the sentence, we repeat this process for $j = \{1, ..., n + 1\}$, where $x_{n+1} =$ `STOP`.
4. The final result presents the $k$-th best output sequences.

The figure below illustrates the first 2 steps of our modified Viterbi algorithm where $k = 2$:

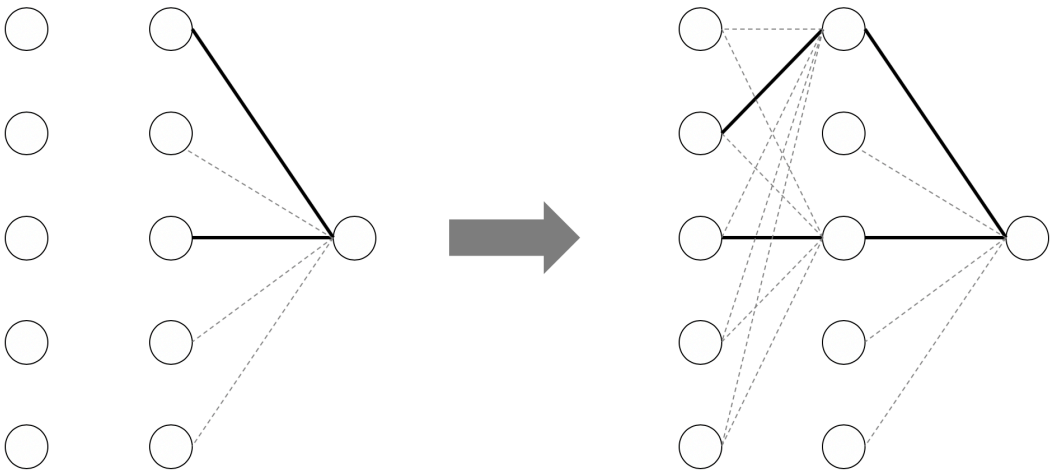


Figure 1: First 2 steps of our modified Viterbi algorithm when $k = 2$

#### 4.2.2 Detailed Steps

**Initialization Step**

We began by initializing $\pi(0, \text{START}) = 1$ and by generating a pool of $k$ seed sequences of STOP. This is the starting point for the recursion in our modified Viterbi algorithm.

**Modified Recursion Step**

For the first $k$ sequences in the pool, perform the following Viterbi algorithm and append the individual results to a new pool:

$$\pi(j, v) = \pi(j-1, u) \cdot a_{u,v} \cdot b_v(x_j), \ \forall \ u \in \mathcal{T}$$

This operation is repeated for all $u \in \mathcal{T}$ and $j = \{1, ..., n+1\}$, where $n$ represents the number of words in the sentence and $x_{n+1} = \texttt{STOP}$.

Sort the pool by probability, and truncate it to leave only the top-$k$ entries.

The algorithm terminates after performing the last step (where $j = 1$ and $u = \text{START}$), providing $k$ sequences of labels.

### 4.3 Results

#### 4.3.1 EN

Running our algorithm on the development set `EN/dev.in`:

| EN | Precision | Recall | F Score |
|---|---|---|---|
| **Entity** | 0.7746 | 0.7645 | 0.7695 |
| **Sentiment** | 0.7437 | 0.734 | 0.7388 |

#### 4.3.2 AL

Running our algorithm on the development set `AL/dev.in`:

| AL | Precision | Recall | F Score |
|---|---|---|---|
| **Entity** | 0.6182 | 0.6376 | 0.6278 |
| **Sentiment** | 0.5437 | 0.5608 | 0.5521 |

# 5 Part 5: Design Challenge

We aim to develop an improved tagging system for the dataset using variations of the HMM model, and also by applying preprocessing strategies. We train our new models on the training set and evaluate on the development set. We use this to evaluate our strategies independently and select the best combination for the final test set.

## 5.1 Separate HMM for Entity and Sentiment

### 5.1.1 Hypothesis

If we use separate HMM to predict the entities and sentiment separately, we might be able to improve the results by reducing the complexity of the task required of each HMM.

### 5.1.2 Method

Through this "divide-and-conquer" approach, we will use two independent HMM with separate parameters to solve the two separate sub-problems: entity and sentiment.

**Entity HMM** This HMM is trained to predict the labels `B`, `I` and `O` based on the input sequence. This reduces the parameter count for both emission and transition, as well as output space of the HMM, resulting in a simpler model and task.

**Sentiment HMM** This HMM is trained to predict other labels, such as `NP`, `VP` etc based on the input sequence. This again reduces the parameter count for both emission and transition, as well as output space of the HMM, resulting in a simpler model and task.

The output of both HMM are combined to give the final label. For example, the final label `B-NP` is given by the Entity HMM producing `B` and the Sentiment HMM producing `NP`.

### 5.1.3 Results

To run our code, run each cell in the file, `part_5_split.ipynb` after changing the `dataset_folder` directory to the respective directories in the second cell.

The results of using the separate HMM for entities and sentiment resulted in significantly poorer F scores. We hypothesize the reason for this being that the entity and sentiment are not independent, and as such we cannot aim to predict them independently.

We used $k = 3$ for all experiments below:

**Entity**

| Dataset | Precision | Recall | F Score |
|---|---|---|---|
| **EN** | 0.7403 | 0.8209 | 0.7785 |
| **AL** | 0.4392 | 0.6070 | 0.5097 |

**Sentiment**

| Dataset | Precision | Recall | F Score |
|---|---|---|---|
| **EN** | 0.7003 | 0.7765 | 0.7365 |
| **AL** | 0.3823 | 0.5283 | 0.4436 |

The code used for the experiment can be found in the notebook `part_5_split.ipynb`.

## 5.2 Extending HMM "Context"

### 5.2.1 Hypothesis

An unmodified HMM generates an output variable `x` from a hidden variable (state) `y` independently of `x` and `y` in previous time steps. That is, the output variable is only dependent on the current state `y`.

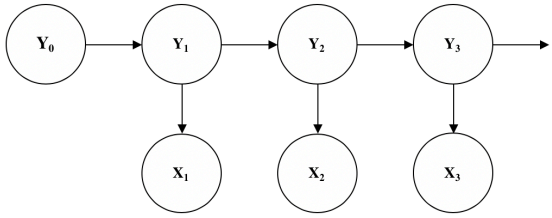



Figure 2: Unmodified "vanilla" Hidden Markov Model (HMM)

We perceive this to be a weakness intuitively that language understanding is very contextual, where the semantic meaning of each word is conditional on its surrounding words (context).

### 5.2.2 Method

We attempt to improve the "vanilla" HMM by forcing the output variable to be conditional on one of two variables that can indicate context: either the previous output variable `x` or the previous hidden variable `y`. We show this in the diagram below:

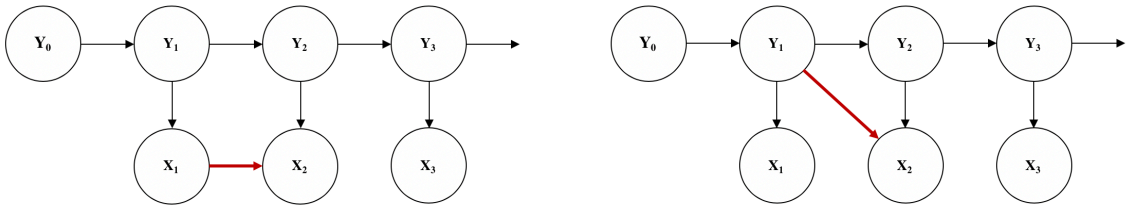



Figure 3: Modified HMM with additional conditioning on previous output variable (left) or previous hidden variable (right).

To achieve this, we model an additional emission parameter $b2$, which is also trained using Maximum Likelihood Estimation (MLE).

**Modelling Previous Word** MLE of output variable given previous output variable:

$$b2_{x_{j-1}}(x_j) = \frac{\text{count}(x_{j-1}, x_j)}{\text{count}(x_j)}$$

**Modelling Previous POS** MLE of output variable given previous hidden variable:

$$b2_{y_{j-1}}(x_j) = \frac{\text{count}(y_{j-1}, x_j)}{\text{count}(x_j)}$$

We then modify the Viterbi algorithm equation by adding in the $b2$ parameter via a multiplication product with the existing expression. This implies that we now wish to find the most probable sequence, taking into account the current output variable's relationship with either the previous output or hidden variable, which is independent of the current output variable's relationship with the current hidden variable.

$$\pi(j, v) = \max_{u \in \mathcal{T}} \{\pi(j-1, u) \cdot a_{u,v} \cdot b_v(x_j) \cdot b2\}, \; \forall \; v \in \mathcal{T}$$

where $b2 = b_{x_{(j-1)}}(x_j)$ if we are modelling previous word or $b2 = b_u(x_j)$ if we are modelling previous POS.

#### 5.2.3 Results

To run our code, run each cell in the file, `part_5_viterbi_context.ipynb` after changing the `dataset_folder` directory to the respective directories in the second cell.

This method caused results to be poorer, hence we will not use it for submission on the test set.

**HMM with Previous Word Context**
**Entity**

| Dataset | Precision | Recall | F Score |
|---|---|---|---|
| **EN** | 0.4549 | 0.3415 | 0.3901 |
| **AL** | 0.4998 | 0.5905 | 0.5414 |

**Sentiment**

| Dataset | Precision | Recall | F Score |
|---|---|---|---|
| **EN** | 0.3409 | 0.2559 | 0.2923 |
| **AL** | 0.3865 | 0.4566 | 0.4186 |

**HMM with Previous POS Context**
**Entity**

| Dataset | Precision | Recall | F Score |
|---|---|---|---|
| **EN** | 0.3434 | 0.1916 | 0.2459 |
| **AL** | 0.093 | 0.166 | 0.1192 |

**Sentiment**

| Dataset | Precision | Recall | F Score |
|---|---|---|---|
| **EN** | 0.0942 | 0.0526 | 0.0675 |
| **AL** | 0.0067 | 0.0119 | 0.0085 |

## 5.3 Tuning k value

### 5.3.1 Hypothesis

We tune the value of k to select the ideal balance between replacing rare words and unseen words with the `##UNK##` token and reducing the amount of vocabulary that our HMM model can capture in its parameters. In this case, we simply select the value of k that produces the best results on the development set.

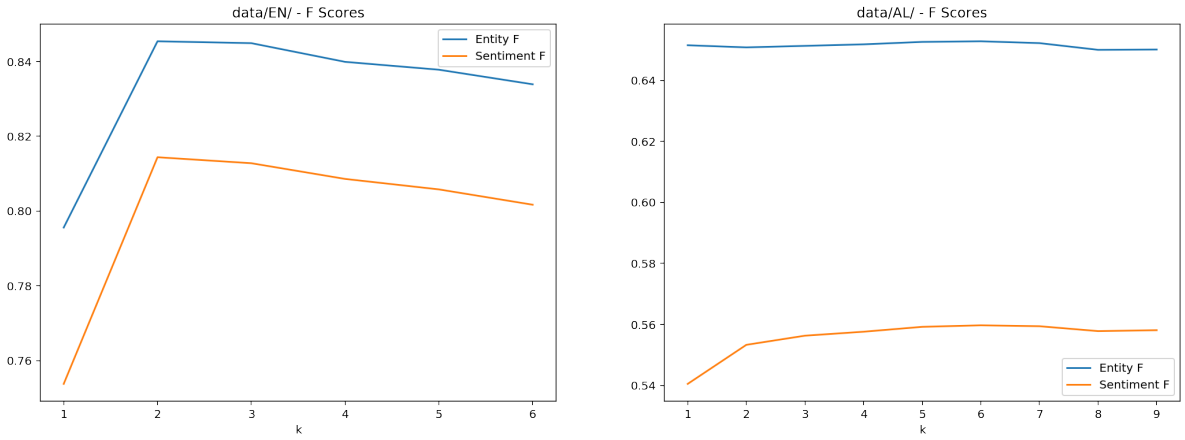
### 5.3.2 Method

We perform a grid search over the values of $k$ ranging from $k = 1$ to $k = 9$.

### 5.3.3 Results

To run our code, run each cell in the file, `part_5_search_k.ipynb` after changing the `dataset_folder` directory to the respective directories in the second cell.

In the process of the grid search, we observe a general trend where the F score increases with $k$, and then decreases. The F scores over the `EN` dataset peaked at $k = 2$ before it worsened. On the other hand, the F scores over the AL dataset peaked at $k = 6$ before it worsened.




(a) Grid search of k over EN dataset.

(b) Grid search of k over AL dataset.

Figure 4: Grid search to select ideal value of k

As such, we will choose the ideal value of $k$ as 2 for the `EN` dataset and 6 for the `AL` dataset.

## 5.4 Preprocessing for EN Dataset

### 5.4.1 Hypothesis

Upon inspection of the vocabulary of the unmodified `EN` dataset, we notice several trends:

- Varied presence of contractions or abbreviations.
- Large number of unique numerical "words".
- Mixture of upper and lower case words.
- Mixture of word forms (such as tense).

We hypothesize that we might be able to reduce the amount of unique vocabulary in the dataset without affecting the semantic information captured in the word sequences through certain preprocessing strategies. In doing so, we reduce the number of parameters and also hopefully increase the generalisability of our model.

### 5.4.2 Method

**Strategies that Improved Results**

- Replaced short-forms or inconsistent abbreviation with its full forms. *(e.g. "a.m" was replaced with "a.m." and "dec." was replaced with "december")*
- Replaced numerical words with numerical values *(e.g. "one" was replaced with "1".)*
- Replaced all uppercase alphabets with lowercase
- Replaced all common punctuation (e.g. "." and ",") with a common token
- Removed certain suffixes (removed plural "s" and "ing")
- Replaced certain words. *(e.g. "was" was replaced with "is" and "were" was replaced with "are")*

**Strategies that Worsened Results**

- Replaced all numbers with `##NUM##` tokens
- Replaced years with `##YEAR##` *(e.g. "1930s" was replaced with* `##YEAR##`*)*
- Removed suffix that indicated past tense ("d" and "ed")

**General Strategy to Remove Suffixes**

It is not trivial to remove a suffix from a word and ensure that it still results in the valid base form of the word. For example, removing the "ing" from "Beijing" does not result in a valid word. In this section, we describe a simple strategy to remove suffixes in a way which results in a **valid** word without the need of an external data source. We employ the following strategy:

1. Remove suffix from word (e.g. "s" or "ing")
2. Check if resulting word is present in dataset
3. For certain suffixes like "ing", optionally check if resulting word is present in dataset if we append an "e". (e.g. "housing" → "house")

### 5.4.3 Results

To run our code, run each cell in the file, `part_5_search_k_prepro_en.ipynb`.

This improves the results slightly. We do another grid search, showing that the improvement is present across all values of $k$, and that the ideal value of $k$ remains the same.

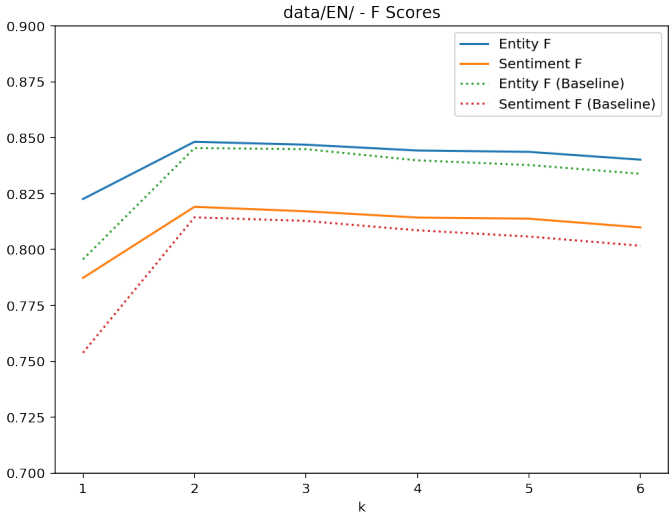



Figure 5: Grid search of k over preprocessed EN dataset.

We therefore combine the ideal value of k found by grid search and our preprocessing strategy, and arrive at our improved result. **We use this final configuration** ($preprocessing + k = 2$) **on the test data.**

| Dataset | Precision | Recall | F Score |
|---|---|---|---|
| **EN-Entity** | 0.8500 | 0.8470 | 0.8485 |
| **EN-Sentiment** | 0.8209 | 0.8180 | 0.8194 |

## 5.5 Preprocessing for AL Dataset

### 5.5.1 Hypothesis

Similar to the `EN` dataset, we might be able to improve our `AL` results through the use of preprocessing. However, the strategies we can use are significantly more restricted as it is harder for us to work with the Chinese characters.

### 5.5.2 Method

**Strategies that Improved Results**

- Normalised separator symbols such as "–", "_" to a common "-"

**Strategies that Worsened Results**

- Replaced all uppercase alphabets with lowercase.
- Replaced all numbers and Chinese numerals with `##NUM##` tokens.

**Strategies that had No Effect**

**Data Augmentation** We attempted data augmentation in order to create more training data by swapping around characters found in various names (identified by POS tags such as `CITY`, `COMMUNITY`, `COUNTRY` etc.). The code can be found in `part_5_al_data_augmentation_test.ipynb`.

An example of a newly generated training example via data augmentation is as follows:

- Original: 和义路 1-5 号汇金大厦 1-6
- Augmented: 度义路 1-5 号汇示大厦 1-6

### 5.5.3 Results

To run our code, run each cell in the file, `part_5_search_k_prepro_al.ipynb`.

Preprocessing improved the results slightly. We again perform a grid search to see if the ideal value of $k$ changes.

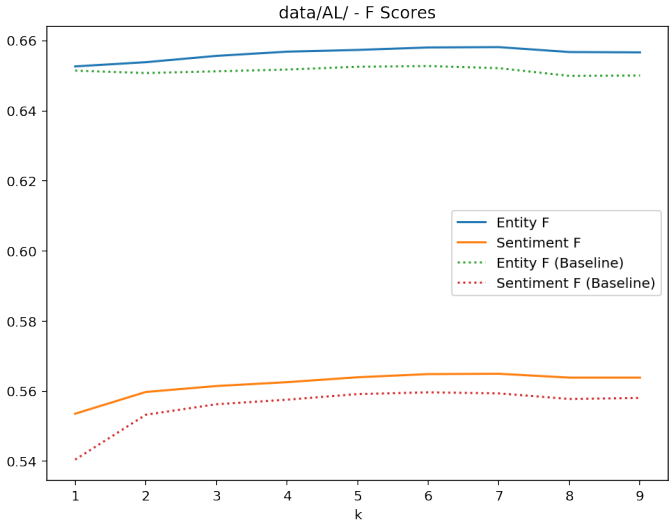



Figure 6: Grid search of k over preprocessed AL dataset.

In this case, the ideal $k$ increased to 7. Our final results for the `AL` dataset are shown below. **We use this final configuration** (*preprocessing* + $k = 7$) **on the test data.**

| Dataset | Precision | Recall | F Score |
|---|---|---|---|
| **AL-Entity** | 0.6197 | 0.7015 | 0.6582 |
| **AL-Sentiment** | 0.5320 | 0.6022 | 0.5650 |