Web Mining

Web Content Mining:
Named Entity Recognition

Simone Ponzetto

FSS 2023
Information Extraction

− **Information extraction (IE)** is the **automatic identification of selected types of entities, relations, or events in free text**

− **Traditionally, IE tasks are the following:**
  • Named entity recognition and classification (NERC)
  • Coreference resolution
  • Relation extraction
  • Event extraction

− **The following tasks loosely belong to IE:**
  • Keywords/keyphrase extraction
  • Terminology extraction
  • Collocation extraction
Outline

1. Named Entity Recognition
2. Evaluation
3. RNNs
4. BERT
Supervised Named Entity Recognition

- In information extraction, a named entity is a real-world object, such as a person, location, organization, product, etc., that can be denoted with a proper name. It can be abstract or have a physical existence.

- Named-entity recognition (NER) is a subtask of information extraction that seeks to locate and classify named entities mentioned in unstructured text into pre-defined categories such as person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc.

Wikipedia, released under Creative Commons Attribution-ShareAlike License 3.0 (https://creativecommons.org/licenses/by-sa/3.0/)
Eastern Ukraine is gripped by an armed separatist uprising, with pro-Russian protesters occupying government buildings in more than a dozen towns and cities, despite an ongoing "anti-terror" operation launched by the Ukrainian military. Vyacheslav Ponomaryov is the self-proclaimed pro-Russian mayor of Sloviansk, Donetsk region, the stronghold of the separatist movement in eastern Ukraine. He was involved in the seizure of a group of military observers from the Organization for Security and Co-operation in Europe (OSCE). One of the best-known leaders of the uprising, Igor Strelkov directs armed pro-Russian activists in eastern Ukraine, especially in Sloviansk. The word is he works for the GRU (Russian military intelligence agency), and his real name is Igor Girkin. He was born in 1970 and registered in Moscow.

– **PERson, LOCation, ORGanization, TIME**

– **Q: What type of NLP task would NER be (from the machine learning perspective)?**
Rule-Based Named Entity Recognition

- Large number of extraction patterns / rules
- Each pattern detects some type of named entities

\[\text{[capitalized-word]+['C'orp.']} \Rightarrow \text{Organization}\]
\[\text{['Mr.'][capitalized-word]+} \Rightarrow \text{Person}\]
\[\text{[in|at|on][capitalized-word]+} \Rightarrow \text{Location}\]

- Unfortunately, most rules have exceptions...

“She lost hope she would ever meet Mr. Right One.” (Person?)
“God only knows what goes on in Putin’s mind.” (Location?)
Building a Named-Entity Tagger

- We can add additional rules to handle exceptions
- E.g., gazetteers: word lists for each of the NER categories
- Some potential gazetteer rules:

```
[cap-word-names-gazetteer]+[cap-word-surnames-gazetteer]+
```

- Personal names: Aaliyah, Aaron, Abbey, ..., Zygmunt, Zyta
- Surnames: Abbott, Abney, Abraham, ..., Zysett, Zyskowsky
- Organizations: Abbott Laboratories, Abercrombie & Fitch, Association for Computational Linguistics, ..., WorldCom, World Help Foundation
- Locations: Alabama, Arkansas, ..., Zimbabwe

- Problem: Gazetteers are always incomplete
- Generally, too many rules, difficult to maintain, etc.
Supervised Named Entity Recognition

- **We need:** a corpus *manually annotated* with named entities
- **Annotations done according to annotation standard**
  - The most renowned annotation standard: MUC-7
    - Chinchor & Robinson, 1997
- **MUC-7 named entity types**
  - Entity names (ENAMEX) – Person, Organization, Location
  - Temporal expressions (TIMEX) – Date, Time
  - Quantities (NUMEX) – Monetary value, Percentage
- **Annotation of named entities is not particularly demanding**
  - No need to hire experts (e.g., linguists)
  - Virtually any native speaker can annotate (after training)
Supervised Named Entity Recognition

- **NER** is a prototypical **sequence labelling** task
  - But named entities are generally multi-token expressions
  - **Q:** What labels do we assign to individual tokens?

- We need to make a **distinction** between the first token of a named entity and all other tokens
  - **Q:** Why?

Barcelona’s/ORG draw/O with/O Atletico/ORG Madrid/ORG at/O Camp/LOC Nou/LOC was/O not/O expected/O, says/O British/ORG Broadcast/ORG Channel’s/ORG La/ORG Liga/ORG football expert Andy/PER West/PER.

- „**British Broadcast Channel’s La Liga**“ – one or two organizations?
Supervised Named Entity Recognition

- **NER is a prototypical sequence labelling task**
  - But named entities are generally multi-token expressions

- **B-I-O annotation scheme**
  - B – Begins a named entity (i.e., first NE token)
  - I – Inside a named entity (i.e., second and subsequent NE tokens)
  - O – Outside of a named entity (i.e., token is not part of any NE)

Barcelona’s/B-ORG draw/O with/O Atletico/B-ORG Madrid/I-ORG at/O Camp/B-LOC Nou/I-LOC was/O not/O expected/O, says/O British/B-ORG Broadcast/I-ORG Channel’s/I-ORG La/B-ORG Liga/I-ORG football expert Andy/B-PER West/I-PER.

- „British Broadcast Channel’s La Liga” – two organizations!
Supervised approaches to NER:

1. **Token-level classification**
   - Naive Bayes, SVM, Logistic regression, Feed-forward NN
   - Cannot use labels from both token sides as features

2. **Sequence labelling**
   - Hidden Markov Models (HMM), Conditional Random Fields (CRF)
     - Require manual feature design
   - Recurrent (or gated convolutional) neural networks
     - Word embeddings as input, no feature design
     - State-of-the-art results

Common features (for feature-based learning algorithms):

- Linguistic features: word, lemma, POS-tag, sentence start, capitalization, ...
- Gazetteer features: is gazetteer entry, starts gazetteer entry, inside of a gazetteer entry (for all gazetteers)
Named Entity Recognition – Document Level

- **Sequence models predict BIO labels at the sentence level**
- **Thus, it’s possible to have different labels for the same named entity at the document level**

Eastern Ukraine is gripped by an armed separatist uprising. Vyacheslav Ponomaryov is the self-proclaimed pro-Russian mayor of Sloviansk, Donetsk region, the stronghold of the separatist movement in eastern Ukraine. He was involved in the seizure of a group of military observers from the Organization for Security and Co-operation in Europe (OSCE). One of the best-known leaders of the uprising, Igor Strelkov directs armed pro-Russian activists in eastern Ukraine, especially in Sloviansk.

- **Enforcing document-level consistency improves NER performance**
Outline

1. Named Entity Recognition
2. Evaluation
3. RNNs
4. BERT
Named Entity Recognition Evaluation

- Comparing system predicted Named Entities (NEs) with gold-annotated Nes
  - In terms of precision, recall, and F-score

1. Lenient (aka MUC) evaluation
   - System NE and gold NE need to be of the same type and overlap in token spans in order to count as a match (i.e., true positive)

2. Strict (aka Exact) evaluation
   - System NE and gold NE need to be of the same type and exactly the same token span order to count as a match (i.e., true positive)

Gold: „The Faculty of Business Informatics and Mathematics issued a diploma...”
Sys1: „The Faculty of Business Informatics and Mathematics issued a diploma...”
Sys2: „The Faculty of Business Informatics and Mathematics issued a diploma...”

- State-of-the-art NER performance (coarse-grained entity types) is around 94% F-score for English, and significantly less for other languages
Named Entity Recognition Evaluation

The F1 score is the harmonic mean of the precision and recall.

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

\[ F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} \]

\[ F_1 = \frac{tp}{tp + \frac{1}{2}(fp + fn)} \]

Wikipedia, released under Creative Commons Attribution-ShareAlike License 3.0 (https://creativecommons.org/licenses/by-sa/3.0/)
## I. Surface string and entity type match

<table>
<thead>
<tr>
<th>Golden Standard</th>
<th>System Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Surface String</strong></td>
<td><strong>Entity Type</strong></td>
</tr>
<tr>
<td>in</td>
<td>O</td>
</tr>
<tr>
<td>New</td>
<td>B-LOC</td>
</tr>
<tr>
<td>York</td>
<td>I-LOC</td>
</tr>
<tr>
<td>.</td>
<td>O</td>
</tr>
</tbody>
</table>

By David S. Batista (https://www.davidsbatista.net/blog/2018/05/09/Named_Entity_Evaluation/), licensed under a [CC BY-NC-SA 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).
II. System hypothesized an entity

<table>
<thead>
<tr>
<th>Golden Standard</th>
<th>System Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Surface String</strong></td>
<td><strong>Entity Type</strong></td>
</tr>
<tr>
<td>an</td>
<td>O</td>
</tr>
<tr>
<td>Awful</td>
<td>O</td>
</tr>
<tr>
<td>Headache</td>
<td>O</td>
</tr>
<tr>
<td>in</td>
<td>O</td>
</tr>
</tbody>
</table>

By David S. Batista (https://www.davidsbatista.net/blog/2018/05/09/Named_Entity_Evaluation/), licensed under a [CC BY-NC-SA 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/)
### III. System misses an entity

<table>
<thead>
<tr>
<th>Golden Standard</th>
<th>System Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface String</td>
<td>Entity Type</td>
</tr>
<tr>
<td>in</td>
<td>O</td>
</tr>
<tr>
<td>Palo</td>
<td>B-LOC</td>
</tr>
<tr>
<td>Alto</td>
<td>I-LOC</td>
</tr>
<tr>
<td>,</td>
<td>O</td>
</tr>
</tbody>
</table>

By David S. Batista (https://www.davidsbatista.net/blog/2018/05/09/Named_Entity_Evaluation/), licensed under a [CC BY-NC-SA 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).
Note

- Note that considering only this 3 scenarios, and discarding every other possible scenario we have a simple classification evaluation that can be measured in terms of false negatives, true positives and false positives, and subsequently compute precision, recall and f1-score for each named-entity type.

- But of course we are discarding partial matches, or other scenarios when the NER system gets the named-entity surface string correct but the type wrong, and we might also want to evaluate these scenarios again at a full-entity level.

By David S. Batista (https://www.davidsbatista.net/blog/2018/05/09/Named_Entity_Evaluation/), licensed under a CC BY-NC-SA 4.0 International License
### IV. System assigns the wrong entity type

<table>
<thead>
<tr>
<th>Golden Standard</th>
<th>System Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Surface String</strong></td>
<td><strong>Entity Type</strong></td>
</tr>
<tr>
<td>I</td>
<td>O</td>
</tr>
<tr>
<td>live</td>
<td>O</td>
</tr>
<tr>
<td>in</td>
<td>O</td>
</tr>
<tr>
<td>Palo</td>
<td>B-LOC</td>
</tr>
<tr>
<td>Alto</td>
<td>I-LOC</td>
</tr>
<tr>
<td>,</td>
<td>O</td>
</tr>
</tbody>
</table>
V. System gets the boundaries of the surface string wrong

<table>
<thead>
<tr>
<th>Golden Standard</th>
<th>System Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Surface String</strong></td>
<td><strong>Entity Type</strong></td>
</tr>
<tr>
<td>Unless</td>
<td>O</td>
</tr>
<tr>
<td>Karl</td>
<td>B-PER</td>
</tr>
<tr>
<td>Smith</td>
<td>I-PER</td>
</tr>
<tr>
<td>resigns</td>
<td>O</td>
</tr>
</tbody>
</table>

By David S. Batista (https://www.davidsbatista.net/blog/2018/05/09/Named_Entity_Evaluation/), licensed under a [CC BY-NC-SA 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/)
VI. System gets the boundaries and entity type wrong

<table>
<thead>
<tr>
<th>Golden Standard</th>
<th>System Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Surface String</td>
<td>Entity Type</td>
</tr>
<tr>
<td>Unless</td>
<td>O</td>
</tr>
<tr>
<td>Karl</td>
<td>B-PER</td>
</tr>
<tr>
<td>Smith</td>
<td>I-PER</td>
</tr>
<tr>
<td>resigns</td>
<td>O</td>
</tr>
</tbody>
</table>

By David S. Batista (https://www.davidsbatista.net/blog/2018/05/09/Named_Entity_Evaluation/), licensed under a [CC BY-NC-SA 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).
CoNLL: NER task

- The Language-Independent Named Entity Recognition task introduced at CoNLL-2003 measures the performance of the systems in terms of precision, recall and f1-score, where:
  - "precision is the percentage of named entities found by the learning system that are correct. Recall is the percentage of named entities present in the corpus that are found by the system. A named entity is correct only if it is an exact match of the corresponding entity in the data file."
  - so basically it only considers scenarios I, II and III, the others described scenarios are not considered for evaluation.

By David S. Batista (https://www.davidsbatista.net/blog/2018/05/09/Named_Entity_Evaluation/), licensed under a CC BY-NC-SA 4.0 International License
Outline

1. Named Entity Recognition
2. Evaluation
3. RNNs
4. BERT
Recurrent neural networks

- **Recurrent neural networks** are neural models that explicitly take into account the sequences
  - Sequences of words in a sentence, sentences in a paragraph, etc.

**General RNN model:**

- **Input:** sequence of input vectors (e.g., word embeddings): \( x_1, \ldots, x_n \)
- **RNN** is a function that converts an arbitrary size sequence \( x_1, \ldots, x_n \) into a fixed size output vector \( y_n \)
  - Analogously, the subsequence \( x_1, \ldots, x_i \) will produce the output \( y_i \)
- The output vector \( y_{i-1} \) of the previous step (\( i-1 \)) is combined with the current input \( x_i \) to produce the output \( y_i \)
- The RNN network is, at time step \( i \), represented with its current state \( s_i \)
General RNN model:

- Defined by two functions:
  - Function $R$ defines how the next state $s_i$ is computed from the previous state $s_{i-1}$ and current input $x_i$
  - Function $O$ defines how the current output $y_i$ is computed from the current state $s_i$

- Obviously, RNN is defined recursively

$$y_n = O(s_n)$$

$$s_n = R(x_n, s_{n-1})$$

$$= R(x_n, R(x_{n-1}, s_{n-2})$$

$$...$$

$$= R(x_n, R...(R(x_1, s_0)))$$

- $\theta$ are RNN parameters (in $R$)
Recurrent neural networks

- For some input sequence of finite length, we can “unroll” the RNN recursion

- $s_n$ (and $y_n$) can be thought of as the encoding of the whole sequence

- This general RNN model is instantiated into concrete models by defining functions $R$ and $O$
  - Two models we will cover: Simple (Elman) RNN and LSTM
Simple (Elman) RNN

- The **simplest** RNN formulation that still captures the ordering of the elements in the sequence

- **Model:**
  - $R = \text{Non-linear transformation } g$ (usually hyperbolic tangent or sigmoid) applied to a linear combination of the input and previous state
    \[
    s_i = R(x_i, s_{i-1}) \\
    = g(x_i W^x + s_{i-1} W^s + b);
    \]
  - $O = \text{identity function}$
    \[
    y_i = O(s_i) \\
    = s_i
    \]

- **Parameters**
  - $\theta = (W^x, W^s, b)$ if inputs are fixed (e.g., pre-trained word embeddings)
  - $\theta = (W^x, W^s, b, X)$ if we are also learning the word representations
Gated architectures

- Simple (Elman) architecture suffers from a problem known as vanishing gradients
  - Error signals from later steps in the sequence diminish quickly in the backpropagation algorithm
  - Thus the updates for early inputs that come from errors for in later steps are very small
    - Essentially, Simple RNN has difficulties capturing long-distance dependencies
  - At each step, the whole RNN state is rewritten

- Gated architectures idea
  - Do not update the whole state at every step
  - Introduce parameters that decide which parts of the state to update
  - Introducing gate vectors:
    - They define which parts of the new state are taken from previous state and which from the current input
  - Models: Long short-term memory (LSTM), Gated Recurrent Unit (GRU)
Long Short-Term Memory Networks

Long Short-Term Memory (LSTM) is an RNN architecture that:

1. Explicitly splits the RNN state into two halves: $s_i = [c_i; h_i]$
   - $c_i$ is the „memory cell”, whereas $h_i$ is the „working memory”

2. Introduces differentiable gating mechanisms – smooth functions that simulate logical gates. There are three gate vectors:
   - Input gate vector: decides how much of the current input $x_i$ should be written to the memory cell $c_i$
   - Forget gate vector: decides which parts of the memory cell should be forgotten, due to new input
   - Output gate vector: decides which parts of the memory cell should be copied to the current memory
   - Gate vectors themselves are computed from the current input $x_i$ and previous state of the working memory $h_{i-1}$
Long Short-Term Memory Networks

- **Concrete computation**
  1. State: $s_i = R(x_i, s_{i-1}) = [c_i; h_i]$
  2. Input gate: $i_g = \text{sigmoid}(x_i W^x_{ig} + h_{i-1} W^h_{ig} + b^g)$
  3. Forget gate: $f_g = \text{sigmoid}(x_i W^x_{fg} + h_{i-1} W^h_{fg} + b^f)$
  4. Output gate: $o_g = \text{sigmoid}(x_i W^x_{og} + h_{i-1} W^h_{og} + b^o)$
  5. Next step raw: $z = \tanh(x_i W^x_z + h_{i-1} W^h_z + b^z)$
  6. Next step memory cell: $c_i = f_g [\cdot_{el}] c_{i-1} + i_g [\cdot_{el}] z$
  7. Next step working memory: $h_i = o_g [\cdot_{el}] \tanh(c_i)$

- $[\cdot_{el}]$ is the element-wise (Hadamard) product of the vectors

- The output at each step is simply the working memory at that step:
  - $y_i = O(s_i) = h_i$

- LSTM has to learn **many more parameters** than Simple RNN:
  - $\theta = [W^x_{ig}, W^h_{ig}, b^g, W^x_{fg}, W^h_{fg}, b^f, W^x_{og}, W^h_{og}, b^o, W^x_z, W^h_z, b^z]$
LSTM

Forget gate
Input gate
Output gate

Cell

Legend:

Layer
ComponentwiseCopy
Concatenate

By Guillaume Chevalier, released under Creative Commons Attribution 4.0 International (CC BY 4.0): https://creativecommons.org/licenses/by/4.0/
Bi-directional RNNs

- Standard RNN at each step only encodes the sequence from one side of the current token
- For many NLP tasks (remember sequence labelling!) we want to incorporate knowledge about the context from both sides
- Bidirectional RNN is a model that combines two unidirectional RNNs encoding in opposite directions
  - Output at step $i$ is the concatenation of output vectors of both RNNs
Multi-Layer (Stacked) RNNs

- RNNs can be stacked in layers, forming a grid
- The input for the first RNN are the actual input $x_1, ..., x_n$
- The input for all other layers are the outputs of previous layer RNN
- This architecture is called Deep RNN
RNN usage patterns

**RNN as encoder**

- **Assumptions:**
  - the last state vector ($s_n$) encodes the **whole sequence**
  - Thus, if we need to make a prediction for the whole sequence, we can use the last state as its representation

- The representation of the whole sequence, the last state, is fed into a classifier
  - E.g., a ternary classifier determining whether the sentence has positive, negative, or neutral sentiment
  - The classifier is usually a feed-forward network with its own set of parameters ($\theta_{cl}$)

$$y = \text{softmax}(s_n W_{cl} + b)$$
RNN usage patterns

RNNs as encoders
- Bidirectional RNN consists of two unidirectional RNN
- We have two states \((s_n^f, s_n^b)\) that encode the whole sequence
  - \(s_n^f\) encodes left-to-right
  - \(s_n^b\) encodes right-to-left

Q: How do we create a sentence representation for classification?
A: We concatenate the two final states: \(s_n = [s_n^f; s_n^b]\)
RNN usage patterns – attention mechanism

RNNs as encoders

- By using the last state to predict the class for the whole sequence, we’re putting more emphasis on the inputs closer to the end
- This is somewhat remedied by bidirectional RNN
  - More emphasis on tokens at the beginning and the end of the sequence
- Still, some inputs (e.g., in the middle of the sequence) might be more important for the overall sequence representation than the others
  - E.g., tokens „good” or „bad” for sentence sentiment classification

Attention mechanism

- Sequence representation should be a weighted sum of outputs at different time steps (all positions in the sequence)
- Weights are determined via special parameters that also need to be learned
Attention mechanism

- Sequence representation is a weighted sum \([+\]_w) of the step outputs
  \[ s = \sum_{i=1}^{n} w_i y_i \]

- Weights \(w_i\) are determined via attention parameter vector \(v_i\):
  - Unnormalized weight for each output can be computed as dot product with attention parameter vector: \(w_i' = v_a \cdot y_i\)
  - Final weights are determined by applying the softmax function on the vector of unnormalized weights:
    \[ [w_1, w_2, ..., w_n] = \text{softmax}([w_1', w_2', ..., w_n']) \]
  - The attention parameter vector is another parameter of the whole model which is being learned (together with RNN parameters and \(\theta_{cl}\))
RNN usage patterns

RNN as transducer

- When RNN is used for sequence labelling (e.g., POS-tagging)
- We need to predict the class at every time step of the RNN

- Again, we couple the RNN with a feed-forward classifier
  - But now we predict the class at every position in the sequence, instead of only from the final sequence state as in the encoder usage pattern
  - The prediction loss for the whole sequence is simply the sum of prediction losses of all token-level predictions
Bi-LSTM-CRF

By Guillaume Chevalier, released under Creative Commons Attribution 4.0 International (CC BY 4.0): https://creativecommons.org/licenses/by/4.0/
Outline

1. Named Entity Recognition
2. Evaluation
3. RNNs
4. BERT
Sequential transfer learning

- Core idea: pretrain the language/text encoder on large amounts of text, so that it learns ”the language”
  - Structure of the language (i.e., syntax)
  - Compositionality of meaning in the language (i.e., semantics)

- If we could „pre-train” such an encoder, it would be generally useful for a wide spectrum of NLP tasks

- On which data do we pretrain such an encoder?
  - Large annotated task-specific datasets? Is it going to be general enough?
  - Large unlabeled datasets? But what is our (pre)training objective then?
Sequential transfer learning

Pretraining

- word2vec
- GloVe
- skip-thought
- InferSent
- ELMo
- ULMFiT
- GPT
- BERT

Adaptation

- classification
- sequence labeling
- Q&A
- ...

Diagram showing the process of sequential transfer learning with pretraining models and adaptation tasks.
Self-supervised pretraining

- We have access to enormous amounts of raw unannotated texts (at least for major language)
- Can we somehow pre-train the encoder using raw text?
  - Yes, via language modeling! Task is to predict the word from the text based on the encoding of the surrounding context

LM-pretraining
- Causal (unidirectional) language modeling: GPT(I, II, and III)
- Bidirectional language modeling: ELMo
- Masked language modeling: BERT

- **Pretraining:** Masked language modeling, MLM (and next sentence prediction, NSP)
- **Encoder architecture:** deep Transformer (attention-based) network
- **After fine-tuning, we have a task-specific encoder**

Image from [Devlin et al., NAACL 19]
BERT

By Jimmy Lin (jimmylin@uwaterloo.ca), released under Creative Commons Attribution 4.0 International (CC BY 4.0): https://creativecommons.org/licenses/by/4.0/
BERT

By Jimmy Lin (jimmylin@uwaterloo.ca), released under Creative Commons Attribution 4.0 International (CC BY 4.0): https://creativecommons.org/licenses/by/4.0/
Web Content Mining

What we covered today

- Named Entity Recognition
- Evaluation
- RNNs
- BERT