

Language Technology

<http://cs.lth.se/edan20/>

Chapter 8: Part-of-Speech Tagging Using Machine-Learning Techniques

Pierre Nugues

Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

September 15, 2022



Motivation

The analysis of sentences often involves the analysis of words.

We can divide it in three main tasks:

- ➊ Identify the type of word, for instance noun or verb using the classical grammar;
- ➋ Identify a group or segment, a noun group for instance:
The big table,
- ➌ Identify a name (a proper noun) for instance are these three words,
Kjell Olof Andersson,

This lecture will show you how to solve the first one, part-of-speech tagging, and you will write a program for the second one, chunking, in a next laboratory assignment.



Part-of-Speech Annotation (CoNLL 2000)

Annotation of: *He reckons the current account deficit will narrow to only # 1.8 billion in September.* We set aside the last column for now.

| | | |
|-----------|-----|------|
| He | PRP | B-NP |
| reckons | VBZ | B-VP |
| the | DT | B-NP |
| current | JJ | I-NP |
| account | NN | I-NP |
| deficit | NN | I-NP |
| will | MD | B-VP |
| narrow | VB | I-VP |
| to | TO | B-PP |
| only | RB | B-NP |
| # | # | I-NP |
| 1.8 | CD | I-NP |
| billion | CD | I-NP |
| in | IN | B-PP |
| September | NNP | B-NP |
| . | . | O |



Designing a Part-of-Speech Tagger

We will now create part-of-speech taggers, where we will examine three architectures:

- ① A feed-forward pipeline with a one-hot encoding of the words;
- ② A feed-forward pipeline with word embeddings: We will replace the one-hot vectors with GloVe embeddings;
- ③ A recurrent neural network, either a simple RNN or a LSTM, with word embeddings.



Features for Part-of-Speech Tagging

The word *visit* is ambiguous in English:

*I paid a **visit** to a friend* → *noun*

*I went to **visit** a friend* → *verb*

The context of the word enables us to tell, here an article or the infinitive marker

To train and apply the model, the tagger extracts a set of features from the surrounding words, for example, a sliding window spanning five words and centered on the current word.

We then associate the feature vector $(w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2})$ with the part-of-speech tag t_i at index i .



Training Set

Part-of-speech taggers use a training set where every word is hand-annotated (Penn Treebank and CoNLL 2008).

| Index | Word | Hand annotation | Index | Word | Hand annotation |
|-------|------------|-----------------|-------|------------|-----------------|
| 1 | Battle | JJ | 19 | of | IN |
| 2 | - | HYPH | 20 | their | PRP\$ |
| 3 | tested | JJ | 21 | countrymen | NNS |
| 4 | Japanese | JJ | 22 | to | TO |
| 5 | industrial | JJ | 23 | visit | VB |
| 6 | managers | NNS | 24 | Mexico | NNP |
| 7 | here | RB | 25 | , | , |
| 8 | always | RB | 26 | a | DT |
| 9 | buck | VBP | 27 | boatload | NN |
| 10 | up | RP | 28 | of | IN |
| 11 | nervous | JJ | 29 | samurai | FW |
| 12 | newcomers | NNS | 30 | warriors | NNS |
| 13 | with | IN | 31 | blown | VBN |
| 14 | the | DT | 32 | ashore | RB |
| 15 | tale | NN | 33 | 375 | CD |
| 16 | of | IN | 34 | years | NNS |
| 17 | the | DT | 35 | ago | RB |
| 18 | first | JJ | 36 | . | . |



Architecture 1: Part-of-Speech Tagging with Linear Classifiers

Linear classifiers are efficient devices to carry out part-of-speech tagging:

- ① The lexical values are the input data to the tagger.
- ② The parts of speech are assigned from left to right by the tagger.

| ID | FORM | PPOS | |
|-----|------------|-------|----------------|
| | BOS | BOS | Padding |
| | BOS | BOS | |
| 1 | Battle | NN | |
| 2 | - | HYPH | |
| 3 | tested | NN | |
| ... | ... | ... | |
| 17 | the | DT | |
| 18 | first | JJ | |
| 19 | of | IN | |
| 20 | their | PRP\$ | |
| 21 | countrymen | NNS | Input features |
| 22 | to | TO | |
| 23 | visit | VB | Predicted tag |
| 24 | Mexico | | |
| 25 | , | | ↓ |
| 26 | a | | |
| 27 | boatload | | |
| ... | ... | ... | |
| 34 | years | | |
| 35 | ago | | |
| 36 | . | | |
| | EOS | | Padding |
| | EOS | | |



Feed-Forward Structure

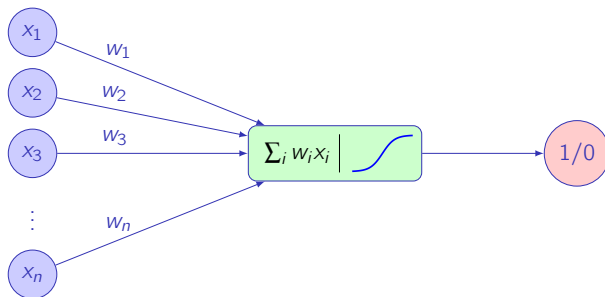
As input, the classifier uses:
 $(w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2})$ to
 predict the part-of-speech tag t_i at
 index i .

Here:
 (countrymen, to, visit, Mexico, ",")
 to predict VB

| ID | FORM | PPOS | |
|-----|------------|-------|----------------|
| | BOS | BOS | Padding |
| | BOS | BOS | |
| 1 | Battle | NN | |
| 2 | - | HYPH | |
| 3 | tested | NN | |
| ... | ... | ... | |
| 17 | the | DT | |
| 18 | first | JJ | |
| 19 | of | IN | |
| 20 | their | PRP\$ | |
| 21 | countrymen | NNS | Input features |
| 22 | to | TO | |
| 23 | visit | VB | Predicted tag |
| 24 | Mexico | | |
| 25 | , | | ↓ |
| 26 | a | | |
| 27 | boatload | | |
| ... | ... | ... | |
| 34 | years | | |
| 35 | ago | | |
| 36 | . | | |
| | EOS | | Padding |
| | EOS | | |



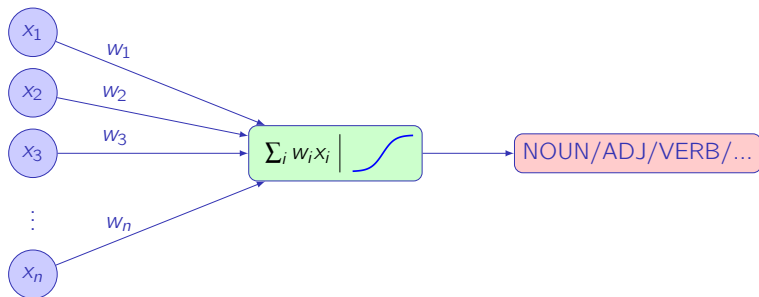
Feed Forward (Binary)



Logistic function



Feed Forward (Multinomial)



Softmax



Feature Vectors

| ID | Feature vectors: X | | | | | PPOS: y |
|-----|--------------------|------------|------------|------------|------------|---------|
| | w_{i-2} | w_{i-1} | w_i | w_{i+1} | w_{i+2} | |
| 1 | BOS | BOS | Battle | - | tested | NN |
| 2 | BOS | Battle | - | tested | Japanese | HYPH |
| 3 | Battle | - | tested | Japanese | industrial | JJ |
| ... | ... | ... | ... | ... | ... | ... |
| 19 | the | first | of | their | countrymen | IN |
| 20 | first | of | their | countrymen | to | PRP\$ |
| 21 | of | their | countrymen | to | visit | NNS |
| 22 | their | countrymen | to | visit | Mexico | TO |
| 23 | countrymen | to | visit | Mexico | , | VB |
| 24 | to | visit | Mexico | , | a | NNP |
| 25 | visit | Mexico | , | a | boatload | , |
| ... | ... | ... | ... | ... | ... | ... |
| 34 | ashore | 375 | years | ago | . | NNS |
| 35 | 375 | years | ago | . | EOS | RB |
| 36 | years | ago | . | EOS | EOS | . |




Word Encoding: One hot encoding

The feature space is defined by all the word values and a word has one dimension

This is a sparse representation

We use DictVectorizer() to encode them:

```
from sklearn.feature_extraction import DictVectorizer
v = DictVectorizer(sparse=False)
X_cat = [{ 'w_1': 'the', 'w_2': 'first', 'w_3': 'of' },
          { 'w_1': 'first', 'w_2': 'of', 'w_3': 'the' },
          { 'w_1': 'of', 'w_2': 'the', 'w_3': 'countrymen' },
          { 'w_1': 'the', 'w_2': 'countrymen', 'w_3': 'to' }]
X = v.fit_transform(X_cat)
X
array([[0., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0.],
       [1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 0.],
       [0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 1.]])
```



Word Encoding

```
v.transform([{'w_1': 'the', 'w_2': 'of', 'w_3': 'Mexico'}])
array([[0., 0., 1., 0., 0., 1., 0., 0., 0., 0.]])
```

```
v.get_feature_names()
['w_1=first',
 'w_1=of',
 'w_1=the',
 'w_2=countrymen',
 'w_2=first',
 'w_2=of',
 'w_2=the',
 'w_3=countrymen',
 'w_3=of',
 'w_3=the',
 'w_3=to']
```

Notebook on GitHub:
[ilppp/programs/ch08/python/lr_pos_tagger_simple.ipynb](https://github.com/ilppp/programs/blob/master/ch08/python/lr_pos_tagger_simple.ipynb)



Architecture 1: A Feed-Forward Neural Network

We use a Keras feed-forward architecture corresponding to a logistic regression:

```
np.random.seed(0)

model = models.Sequential([Dense(NB_CLASSES,
                                  input_dim=X.shape[1],
                                  activation='softmax')])

model.compile(loss='sparse_categorical_crossentropy',
              optimizer=OPTIMIZER,
              metrics=['accuracy'])

model.summary()

model.fit(X, y, epochs=EPOCHS, batch_size=BATCH_SIZE)

model.save('out.model')
```



Encoding the \mathbf{y} Vector

In the previous examples, we used `categorical_crossentropy`. This requires that all the targets are encoded with one-hot vectors. For instance:

- determiner: [1, 0, 0, 0]
- noun: [0, 1, 0, 0]
- verb: [0, 0, 1, 0]
- adjective: [0, 0, 0, 1]

With `sparse_categorical_crossentropy`, we can use numerical indices:

- determiner: 1
- noun: 2
- verb: 3
- adjective: 4

We do not need to use the `to_categorical` function.



Preprocessing

Preprocessing is more complex though: Four steps:

- 1 Read the corpus

```
train_sentences, dev_sentences, test_sentences, \
    column_names = load_ud_en_ewt()
```

- 2 Store the rows of the CoNLL corpus in dictionaries

```
conll_dict = CoNLLDictorizer(column_names, col_sep='\t')
train_dict = conll_dict.transform(train_sentences)
test_dict = conll_dict.transform(test_sentences)
```

- 3 Extract the features and store them in dictionaries

```
context_dictorizer = ContextDictorizer()
context_dictorizer.fit(train_dict)
X_dict, y_cat = context_dictorizer.transform(train_dict)
```

- 4 Vectorize the symbols

```
# We transform the X symbols into numbers
dict_vectorizer = DictVectorizer()
X_num = dict_vectorizer.fit_transform(X_dict)
```



Code Example

Jupyter Notebook: <https://github.com/pnugues/edan95/blob/master/programs/4.1-nn-pos-tagger.ipynb>

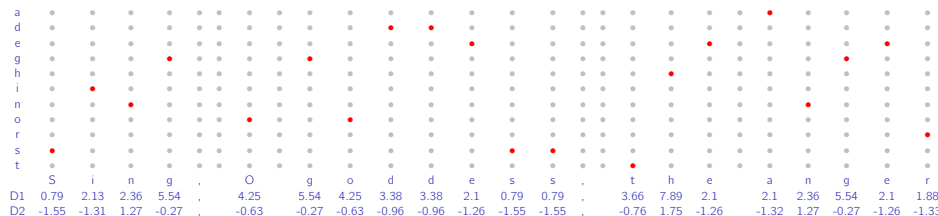


Architecture 2: Dense Word Encoding

One-hot encoding is results in sparse matrices

An alternative is to use dense vectors, for instance from a principal component analysis. Compare the character encodings of:

Sing, O Goddess, the anger...



Notebook, where we use GloVe:

[ilppp/programs/ch08/python/lr_pos_tagger_emb.ipynb](https://github.com/pnugues/programs/blob/master/ch08/python/lr_pos_tagger_emb.ipynb)



Architecture 2: Using Embeddings

We replace the one-hot vectors with embeddings, the rest being the same
Word embeddings are dense vectors obtained by a principal component analysis or another method.

They can be trained by the neural network or pretrained

In this implementation:

- 1 We use pretrained embeddings from the GloVe project;
- 2 Our version of GloVe is lowercased, so we set all the characters in lowercase;
- 3 We add the embeddings as an `Embedding` layer at the start of the network;
- 4 We initialize the embedding layer with GloVe and make it trainable or not.

It would be possible to use a randomly initialized matrix as embeddings instead



The Embedding Layer

```
embedding_matrix = np.random.random(  
    (len(vocabulary_words) + 1,  
     EMBEDDING_DIM))  
  
for word in vocabulary_words:  
    if word in embeddings_dict:  
        embedding_matrix[word_idx[word]] = embeddings_dict[word]  
  
model = models.Sequential([  
    Embedding(cnt_uniq,  
              EMBEDDING_DIM,  
              trainable=True,  
              embeddings_initializer=  
                  initializers.Constant(embedding_matrix),  
              input_length=2 * W_SIZE + 1),  
    Flatten(),  
    Dense(NB_CLASSES, activation='softmax')  
])
```



Code Example

Jupyter Notebook: <https://github.com/pnugues/edan95/blob/master/programs/4.2-nn-pos-tagger-embeddings.ipynb>



Architecture 3: Recurrent Structure

As input, the classifier uses:

$(w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}, t_{i-2}, t_{i-1})$,
to predict the part-of-speech tag t_i
at index i .

Here:

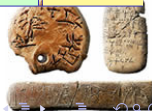
(countrymen, to, visit, Mexico, ",",
NNS, TO) to predict VB

| ID | FORM | PPOS | |
|-----|------------|-------|----------------|
| | BOS | BOS | Padding |
| | BOS | BOS | |
| 1 | Battle | NN | |
| 2 | - | HYPH | |
| 3 | tested | NN | |
| ... | ... | ... | |
| 17 | the | DT | |
| 18 | first | JJ | |
| 19 | of | IN | |
| 20 | their | PRP\$ | |
| 21 | countrymen | NNS | Input features |
| 22 | to | TO | |
| 23 | visit | VB | Predicted tag |
| 24 | Mexico | | |
| 25 | , | | ↓ |
| 26 | a | | |
| 27 | boatload | | |
| ... | ... | ... | |
| 34 | years | | |
| 35 | ago | | |
| 36 | . | | |
| | EOS | | Padding |
| | EOS | | |

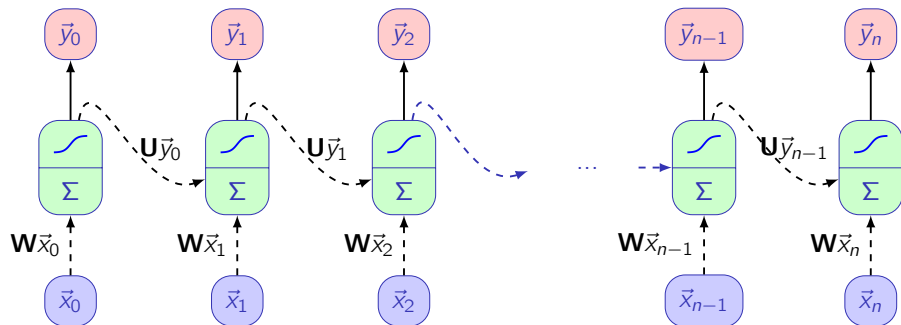


Feature Vectors

| ID | Feature vectors: X | | | | | | | PPOS: y |
|-----|----------------------|------------|------------|------------|------------|-----------|-----------|-----------|
| | w_{i-2} | w_{i-1} | w_i | w_{i+1} | w_{i+2} | t_{i-2} | t_{i-1} | |
| 1 | BOS | BOS | Battle | - | tested | BOS | BOS | NN |
| 2 | BOS | Battle | - | tested | Japanese | BOS | NN | HYPH |
| 3 | Battle | - | tested | Japanese | industrial | NN | HYPH | JJ |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 19 | the | first | of | their | countrymen | DT | JJ | IN |
| 20 | first | of | their | countrymen | to | JJ | IN | PRP\$ |
| 21 | of | their | countrymen | to | visit | IN | PRP\$ | NNS |
| 22 | their | countrymen | to | visit | Mexico | PRP\$ | NNS | TO |
| 23 | countrymen | to | visit | Mexico | , | NNS | TO | VB |
| 24 | to | visit | Mexico | , | a | TO | VB | NNP |
| 25 | visit | Mexico | , | a | boatload | VB | NNP | , |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 34 | ashore | 375 | years | ago | . | RB | CD | NNS |
| 35 | 375 | years | ago | . | EOS | CD | NNS | RB |
| 36 | years | ago | . | EOS | EOS | NNS | RB | . |



The RNN Architecture



Input Format for RNNs

The input format is different from feed forward networks.

We need to build two lists: one for the input and the other for the output

| | | | | | |
|----------|-----|--------|---------|-----|------|
| y | DET | NOUN | VERB | DET | NOUN |
| x | The | waiter | brought | the | meal |

All the vectors in a same batch must have the same length. We pad them:

| | | | | | | | | |
|----------|-----|-----|-----|-----|--------|---------|-----|------|
| y | PAD | PAD | PAD | DET | NOUN | VERB | DET | NOUN |
| x | PAD | PAD | PAD | The | waiter | brought | the | meal |

We could apply the padding after



Building the Sequences

```
def build_sequences(corpus_dict, key_x='form', key_y='pos',
                    tolower=True):
    X, Y = [], []
    for sentence in corpus_dict:
        x, y = [], []
        for word in sentence:
            x += [word[key_x]]
            y += [word[key_y]]
        if tolower:
            x = list(map(str.lower, x))
        X += [x]
        Y += [y]
    return X, Y
```

At this point, we have **x** and **y** vectors of symbols



Building Index Sequences

0 is for the padding symbol and 1 for the unknown words

```
idx_word = dict(enumerate(vocabulary_words, start=2))  
idx_pos = dict(enumerate(pos, start=2))  
word_idx = {v: k for k, v in idx_word.items()}  
pos_idx = {v: k for k, v in idx_pos.items()}
```

At this point, we have **x** and **y** vectors of numbers



Padding the Index Sequences

We build the complete **X_idx** and **Y_idx** matrices for the whole corpus
And we pad the matrices:

```
X = pad_sequences(X_idx)
```

```
Y = pad_sequences(Y_idx)
```

```
# The number of POS classes and 0 (padding symbol)
```

```
Y_train = to_categorical(Y, num_classes=len(pos) + 2)
```

`pad_sequences` can have an argument that specifies the maximal length
`maxlen (MAX_SEQUENCE_LENGTH)`.

The padded sentences must have the same length in a batch. This is
automatically computed by Keras



Recurrent Neural Networks (RNN)

```
model = models.Sequential([
    Embedding(len(vocabulary_words) + 2,
              EMBEDDING_DIM,
              mask_zero=True,
              embeddings_initializer=
                  initializers.Constant(embedding_matrix),
              trainable=True,
              input_length=None),
    SimpleRNN(100, return_sequences=True),
    # Bidirectional(SimpleRNN(100, return_sequences=True)),
    Dense(NB_CLASSES + 2, activation='softmax')])
```



Parameters

Keras functions have many parameters.

In case of doubt, read the documentation

A few useful parameters:

- 1 `mask_zero=True` is to tell whether or not the input value 0 is a special “padding” value;
- 2 `return_sequences=True` tells whether to return the last output in the output sequence, or the full sequence. In sequences, it is essential;
- 3 `recurrent_dropout=0.3` tells how much to drop for the linear transformation of the recurrent state.



Code Example

Jupyter Notebook: `4.3-rnn-pos-tagger.ipynb`



Long Short-Term Memory (LSTM)

```
model = models.Sequential([
    Embedding(len(vocabulary_words) + 2,
              EMBEDDING_DIM,
              mask_zero=True,
              embeddings_initializer=
                  initializers.Constant(embedding_matrix),
              trainable=True,
              input_length=None),
    Bidirectional(LSTM(100, return_sequences=True)),
    Dense(NB_CLASSES + 2, activation='softmax')])
```



Segment Recognition

Group detection – chunking –:

Brackets: [_{NG} The government _{NG}] has [_{NG} other agencies and instruments _{NG}] for pursuing [_{NG} these other objectives _{NG}] .

Tags: *The/I government/I has/O other/I agencies/I and/I instruments/I for/O pursuing/O these/I other/I objectives/I ./O*

Brackets: Even [_{NG} Mao Tse-tung _{NG}] [_{NG} 's China _{NG}] began in [_{NG} 1949 _{NG}] with [_{NG} a partnership _{NG}] between [_{NG} the communists _{NG}] and [_{NG} a number _{NG}] of [_{NG} smaller, non-communists parties _{NG}] .

Tags: *Even/O Mao/I Tse-tung/I 's/B China/I began/O in/O 1949/I with/O a/I partnership/I between/O the/I communists/I and/O a/I number/I of/O smaller/I ,/I non-communists/I parties/I ./O*



Segment Categorization

Tages extendible to any type of chunks: nominal, verbal, etc.

For the IOB scheme, this means tags such as I.Type, O.Type, and B.Type, Types being NG, VG, PG, etc.

In CoNLL 2000, ten types of chunks

| Word | POS | Group | Word | POS | Group |
|----------------|-----|-------|------------------|-----|-------|
| <i>He</i> | PRP | B-NP | <i>to</i> | TO | B-PP |
| <i>reckons</i> | VBZ | B-VP | <i>only</i> | RB | B-NP |
| <i>the</i> | DT | B-NP | <i>£</i> | # | I-NP |
| <i>current</i> | JJ | I-NP | <i>1.8</i> | CD | I-NP |
| <i>account</i> | NN | I-NP | <i>billion</i> | CD | I-NP |
| <i>deficit</i> | NN | I-NP | <i>in</i> | IN | B-PP |
| <i>will</i> | MD | B-VP | <i>September</i> | NNP | B-NP |
| <i>narrow</i> | VB | I-VP | <i>.</i> | . | O |

Noun groups (NP) are in red and verb groups (VP) are in blue.



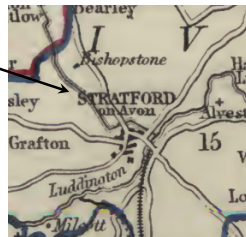
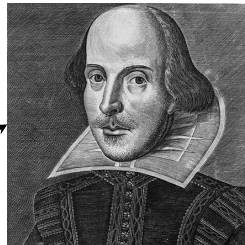
IOB Annotation for Named Entities

| CoNLL 2002 | | CoNLL 2003 | | | |
|------------|----------------|------------|-----|--------|----------------|
| Words | Named entities | Words | POS | Groups | Named entities |
| Wolff | B-PER | U.N. | NNP | I-NP | I-ORG |
| , | O | official | NN | I-NP | O |
| currently | O | Ekeus | NNP | I-NP | I-PER |
| a | O | heads | VBZ | I-VP | O |
| journalist | O | for | IN | I-PP | O |
| in | O | Baghdad | NNP | I-NP | I-LOC |
| Argentina | B-LOC | . | . | O | O |
| , | O | | | | |
| played | O | | | | |
| with | O | | | | |
| Del | B-PER | | | | |
| Bosque | I-PER | | | | |
| in | O | | | | |
| the | O | | | | |
| final | O | | | | |
| years | O | | | | |
| of | O | | | | |
| the | O | | | | |
| seventies | O | | | | |
| in | O | | | | |
| Real | B-ORG | | | | |
| Madrid | I-ORG | | | | |
| . | O | | | | |



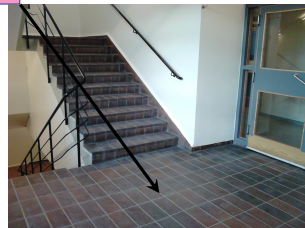
Named Entities: Proper Nouns

William Shakespeare was born and brought
up in Stratford-upon-Avon



Others Entities: Common Nouns

Meeting with our guest on the landing at
lunchtime



Evaluation

There are different kinds of measures to evaluate the performance of machine learning techniques, for instance:

- Precision and recall in information retrieval and natural language processing;
- The *receiver operating characteristic* (ROC) in medicine.

| | Positive examples: P | Negative examples: N |
|-------------------|------------------------|------------------------|
| Classified as P | True positives: A | False positives: B |
| Classified as N | False negatives: C | True negatives: D |

More on the receiver operating characteristic here: http://en.wikipedia.org/wiki/Receiver_operating_characteristic



Recall, Precision, and the F-Measure

The **accuracy** is $\frac{|AUD|}{|PUN|}$.

Recall measures how much relevant examples the system has classified correctly, for P :

$$\text{Recall} = \frac{|A|}{|A \cup C|}.$$

Precision is the accuracy of what has been returned, for P :

$$\text{Precision} = \frac{|A|}{|A \cup B|}.$$

Recall and precision are combined into the **F-measure**, which is defined as the harmonic mean of both numbers:

$$F = \frac{2 \cdot \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$



Evaluation

Accuracy, precision, and recall.

For noun groups with the predicted output:

| Word | POS | Group | Word | POS | Group |
|----------------|-----|-------------|------------------|-----|-------------|
| <i>He</i> | PRP | B-NP | <i>to</i> | TO | B-PP |
| <i>reckons</i> | VBZ | B-VP | <i>only</i> | RB | B-NP |
| <i>the</i> | DT | B-NP | <i>£</i> | # | I-NP |
| <i>current</i> | JJ | B-NP | <i>1.8</i> | CD | B-NP |
| <i>account</i> | NN | I-NP | <i>billion</i> | CD | I-NP |
| <i>deficit</i> | NN | I-NP | <i>in</i> | IN | B-PP |
| <i>will</i> | MD | B-VP | <i>September</i> | NNP | B-NP |
| <i>narrow</i> | VB | I-VP | <i>.</i> | . | O |

Accuracy = $\frac{14}{16}$, recall = $\frac{2}{4} = 0.5$, precision = $\frac{2}{6} = 0.33$
 harmonic mean = $2 \times \frac{0.33 \times 0.5}{0.33 + 0.5} = 0.4$



Course Content: 2022

The rest of the slides in this document is not part of the course in 2022.

You can nonetheless read it if you are curious.



Conditional Random Fields

A tagger produces a sequence of tags, where a given tag obviously depends on the previous ones.

For instance, a preposition cannot follow a determiner

We can model the tag transitions probabilities using **conditional random fields** (CRF)

The simplest form is the **linear chain**.

If \mathbf{y} denotes the output, here a sequence of tags, and \mathbf{x} , a sequence of inputs, consisting for instance of the words and the characters, we try to maximize

$$P(\mathbf{y}|\mathbf{x}),$$

i.e.

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}).$$



Conditional Random Fields (II)

As we want the output sequence to depend on the input and on previously predicted output, we rewrite $P(\mathbf{y}|\mathbf{x})$ as a joint probability, $P(\mathbf{y}, \mathbf{x})$, and, more precisely, as:

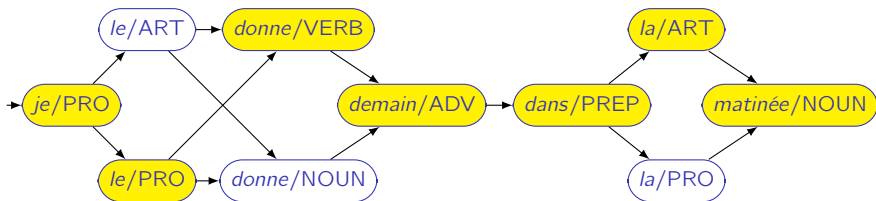
$$\begin{aligned}
 P(\mathbf{y}|\mathbf{x}) &= \frac{P(\mathbf{y}, \mathbf{x})}{\sum_{\mathbf{y}' \in Y} P(\mathbf{y}', \mathbf{x})}, \\
 &= \frac{\prod \exp(w_1 \cdot f(y_t, \mathbf{x}, t)) \cdot \prod \exp(w_2 \cdot f(y_t, y_{t+1}))}{\sum_{\mathbf{y}' \in Y} P(\mathbf{y}', \mathbf{x})},
 \end{aligned}$$

where \mathbf{y}' denotes a sequence, Y , the set of all the possible sequences, and $\sum_{\mathbf{y}' \in Y} P(\mathbf{y}', \mathbf{x})$ is a normalizing factor to have a sum of probabilities of one



Viterbi (Informal)

Je le donne demain dans la matinée 'I give it tomorrow in the morning'



Viterbi (Informal)

The term brought by the word *demain* has still the memory of the ambiguity of *donne*: $P(\text{adv}|\text{verb}) \times P(\text{demain}|\text{adv})$ and $P(\text{adv}|\text{noun}) \times P(\text{demain}|\text{adv})$.

This is no longer the case with *dans*.

According to the noisy channel model and the bigram assumption, the term brought by the word *dans* is $P(\text{dans}|\text{prep}) \times P(\text{prep}|\text{adv})$.

It does not show the ambiguity of *le* and *donne*.

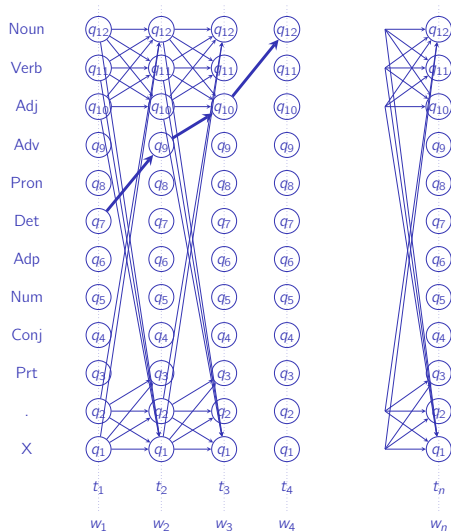
The subsequent terms will ignore it as well.

We can discard the corresponding paths.

The optimal path does not contain nonoptimal subpaths.



Trellis Representation



Filling the Trellis

| $i \backslash \delta$ | δ_1 | δ_2 | δ_3 | δ_4 | δ_5 | δ_6 | δ_7 | δ_8 |
|-----------------------|------------|------------|------------|--------------|---------------|-------------|------------|----------------|
| PREP | 0 | | | | | | | |
| ADV | 0 | | | | | | | |
| PRO | 0 | | | | | | | |
| VERB | 0 | | | | | | | |
| NOUN | 0 | | | | | | | |
| ART | 0 | | | | | | | |
| <s> | 1.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | <s> | <i>Je</i> | <i>le</i> | <i>donne</i> | <i>demain</i> | <i>dans</i> | <i>la</i> | <i>matinée</i> |

To fill the δ_3 column, for each cell j , we compute

$$\max_i P(j|i) \times P(le|j) \times \delta_2(i).$$

The pronoun cell, for instance, is filled with

$$\max_i P(\text{PRO}|i) \times P(le|\text{PRO}) \times \delta_2(i).$$



Supervised Learning: A Summary

Needs a manually annotated corpus called the **Gold Standard**

The Gold Standard may contain errors (*errare humanum est*) that we ignore

A classifier is trained on a part of the corpus, the **training set**, and evaluated on another part, the **test set**, where automatic annotation is compared with the “Gold Standard”

N-fold cross validation is used avoid the influence of a particular division
Some algorithms may require additional optimization on a development set

Classifiers can use statistical or symbolic methods

