Machine learning in NLP
Additional information about Assignment 3

Richard Johansson

October 16, 2014
assignment 3: overview

- implement the perceptron algorithm for structured objects
- use this to build a dependency parser and a named entity tagger
refresher: typical use of scikit-learn classifiers (training)

- get the training data (e.g. read it from a file)
  - we then have a list of **inputs** and another list of corresponding **outputs**
- extract **features** from each input object
  - e.g. a dict with attributes and their values
- convert the features to vectors using one of the **vectorizers**
  - `vec.fit(X)` to create the feature→dimension mapping
  - `Xe = vec.transform(X)` to convert the training set
  - *(fit_transform carries out both steps)*
- **initialize** the classifier object
- **train** the classifier
  - call `classifier.fit(Xe, Y)`
- *(optional: wrap the vectorizer and the classifier into a Pipeline)*
refresher: typical use of scikit-learn classifiers (new data)

- read the data
- **extract features** from each input object
- convert the features to vectors using the previous **vectorizer**
  - again $X_e = \text{transform}(X)$
- for each input, **predict** the output using the classifier
  - $\text{guess} = \text{classifier}.\text{predict}(x)$
- output or evaluate the result
so how does this relate to parsing and tagging?

- when you build the parser or tagger in this assignment, the code will have a quite similar structure
- ...but the scikit-learn components are replaced with specialized components
  - e.g. a ParseVectorizer tailored for sentences/parses, as opposed to the general-purpose DictVectorizer
- we now consider the parser
training the parser

- read the training treebank
  - use the function read_dependency_treebank
  - we then have a list of inputs (sentences) and another list of corresponding outputs (parse trees)
- convert the sentences to vectors using the ParseVectorizer
  - vec.fit(X, Y) to create the feature-dimension mapping
  - Xe, Ye = vec.transform(X, Y) to convert the training set
  - this step includes feature extraction
- initialize the “classifier” (the parse predictor)
  - we give it a problem definition object that contains parsing-specific functionality
- train the parse predictor
  - call parser.fit(Xe, Ye)
after reading the training set

X

<D> Lisa walks home

<D> Put it in the car

Y
the actual Python objects

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ [ (’&lt;TOP&gt;’, ’&lt;TOP&gt;’),</td>
<td>[ [ -1, 2, 0, 2 ],</td>
</tr>
<tr>
<td>(’Lisa’, ’NNP’),</td>
<td>[ -1, 0, 1, 1, 5, 3 ]]</td>
</tr>
<tr>
<td>(’walks’, ’VBZ’),</td>
<td></td>
</tr>
<tr>
<td>(’home’, ’RB’) ]],</td>
<td></td>
</tr>
<tr>
<td>[ (’&lt;TOP&gt;’, ’&lt;TOP&gt;’),</td>
<td></td>
</tr>
<tr>
<td>(’Put’, ’VB’),</td>
<td></td>
</tr>
<tr>
<td>(’it’, ’PRP’),</td>
<td></td>
</tr>
<tr>
<td>(’in’, ’IN’),</td>
<td></td>
</tr>
<tr>
<td>(’the’, ’DT’),</td>
<td></td>
</tr>
<tr>
<td>(’car’, ’NN’) ] ]</td>
<td></td>
</tr>
</tbody>
</table>
what happens in transform in the ParseVectorizer?

<D> Lisa walks home

Lisa walks home

Lisa walks home
pseudo code for training the perceptron

\[
\mathbf{w} = (0, \ldots, 0)
\]

repeat \(N\) times

for \((x, y)\) in \(T\)

\[
g = \arg\max_{y'} \mathbf{w} \cdot f(x, y')
\]

if \(g\) is not equal to \(y\)

\[
\mathbf{w} = \mathbf{w} + f(x, y) - f(x, g)
\]

return \(\mathbf{w}\)

- the two highlighted parts are specific to the problem we’re considering, in this case parsing
- this is why we wrap them into a “problem definition” object
- now, let’s look at the implementation of those two methods
the methods in the problem definition

- **predict**: if we have a weight vector \( w \), find the highest-scoring parse tree for the sentence \( x \)
- **get_features**: find the feature vector \( f(x, y) \) for a sentence \( x \) and a tree \( y \)
predict in MSTParsingDefinition

Lisa walks home <D>

\[
\arg\max_y w \ast f(x, y)
\]
get_features in MSTParsingDefinition

Lisa walks home

\[ f(x, y) \]

\[ x \quad \text{Lisa} \quad \text{walks} \quad \text{home} \]

\[ y \]
a technical note on the feature vectors

\[ f(x, y) \]

- see the code or McDonald’s paper for a description of what features we’re using
- the features are stored in sparse vectors
- for practical reasons, `get_features` returns a sparse matrix, that is a “list” of vectors
  - use the helper function `add_sparse_rows_to_dense` to add all edge vectors to \( w \)
running the parser on new data

- read the testing treebank
- convert the sentences to vectors using your previous ParseVectorizer
  - $X_e, Y_e = \text{vec}.\text{transform}(X, Y)$ to convert the training set
- for each input, predict the output using the parser you trained previously
  - $\text{guess} = \text{parser}.\text{predict}(x)$
- output or evaluate the result
  - in this assignment: compute the attachment accuracy
attachment evaluation: example

gold-standard trees

\[
\begin{array}{l}
\begin{bmatrix}
-1, 2, 0, 2 \\
-1, 0, 1, 1, 5, 3
\end{bmatrix}
\end{array}
\]

predicted trees

\[
\begin{array}{l}
\begin{bmatrix}
-1, 2, 0, 2 \\
-1, 0, 1, 1, 5, 3
\end{bmatrix}
\end{array}
\]

attachment accuracy

\[
\frac{\text{number of correct attachments}}{\text{number of tokens}} = \frac{7}{8}
\]

▶ remember not to count the dummy root token!