Decision Trees vs. Linear Models

• Decision Trees are an intuitive way to learn classifiers from data
  • They fit the training data well
  • With heavy pruning, you can control overfitting
• NLP practitioners often use linear models instead
• Most of the models discussed in Sarawagi Chapter 3 are linear models
Decision Trees for NER

• So far we have seen:
  • How to learn rules for NER
  • A basic idea of how to formulate NER as a classification problem
  • Decision trees
    • Including the basic idea of overfitting the training data

• How can we use decision trees for NER?
Rule Sets as Decision Trees

- Decision trees are quite powerful
- It is easy to see that complex rules can be encoded as decision trees
- For instance, let's go back to border detection in CMU seminars...
... the Seminar at `<stime>` 4 pm will ...

<table>
<thead>
<tr>
<th>Position</th>
<th>Condition</th>
<th>Context-independent features</th>
<th>Context Dep.</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Word</td>
<td>Lemma</td>
<td>Capitalization</td>
<td>SemCat</td>
</tr>
<tr>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td><code>at</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Path in the Decision Tree

• The tree will check if the token to the left of the possible start position has "at" as a lemma
• Then check if the token after the possible start position is a Digit
• Then check the second token after the start position is a timeid ("am", "pm", etc)
• If you follow this path at a particular location in the text, then the decision should be to insert a <stime>
Linear Models

• However, in practice decision trees are not used so often in NLP
• Instead, linear models are used
• Let me first present linear models
• Then I will compare linear models and decision trees
Binary Classification

• I'm going to first discuss linear models for binary classification, using binary features
• We'll take the same scenario as before
• Our classifier is trying to decide whether we have a `<stime>` tag or not at the current position (between two words in an email)
• The first thing we will do is encode the context at this position into a feature vector
Feature Vector

• Each feature is true or false, and has a position in the feature vector
• The feature vector is typically sparse, meaning it is mostly zeros (i.e., false)
• The feature vector represents the full feature space. For instance, consider...
... the Seminar at \(<\text{stime}\>) 4 pm will ...

<table>
<thead>
<tr>
<th>Position</th>
<th>Condition</th>
<th>Context-independent features</th>
<th>Context Dep.</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Word</td>
<td>Lemma</td>
<td>Capitalization</td>
<td>SemCat</td>
</tr>
<tr>
<td>-3</td>
<td>the</td>
<td>the</td>
<td>lowercase</td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>Seminar</td>
<td>seminar</td>
<td>uppercase</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>at</td>
<td>at</td>
<td>lowercase</td>
<td>Prep</td>
</tr>
<tr>
<td>+1</td>
<td>4</td>
<td>4</td>
<td>lowercase</td>
<td>Digit</td>
</tr>
<tr>
<td>+2</td>
<td>pm</td>
<td>pm</td>
<td>lowercase</td>
<td>timeid</td>
</tr>
<tr>
<td>+3</td>
<td>will</td>
<td>will</td>
<td>lowercase</td>
<td>Verb</td>
</tr>
</tbody>
</table>

Example modified from Ciravegna 2009
Our features represent this table using binary variables.
For instance, consider the lemma column.
Most features will be false (false = off = 0).
The lemma features that will be on (true = on = 1) are:

-3_lemma_the
-2_lemma_Seminar
-1_lemma_at
+1_lemma_4
+2_lemma_pm
+3_lemma_will
Classification

• To classify we will take the dot product of the feature vector with a learned weight vector
• We will say that the class is true (i.e., we should insert a \(<\text{stime}>\) here) if the dot product is \(> 0\), and false otherwise
• Because we might want to shift the decision boundary, we add a feature that is always true
  • This is called the bias
  • By weighting the bias, we can shift where we make the decision (see next slide)
Feature Vector

- We might use a feature vector like this:
  (this example is simplified – really we'd have all features for all positions)

```
1  Bias term
0  ... (say, -3_lemma_giraffe)
1  -3_lemma_the
0  ...
1  -2_lemma_Seminar
0  ...
0  ...
1  -1_lemma_at
1  +1_lemma_4
0  ...
1  +1_Digit
1  +2_timeid
```
Weight Vector

• Now we'd like the dot product to be $> 0$ if we should insert a `<stime>` tag
• To encode the rule we looked at before we have three features that we want to have a positive weight
  • `-1_lemma_at`
  • `+1_Digit`
  • `+2_timeid`
• We can give them weights of 1
• Their sum will be three
• To make sure that we only classify if all three weights are on, let's set the weight on the bias term to -2
Dot Product - 1

To compute the dot product first take the product of each row, and then sum these:

<table>
<thead>
<tr>
<th>1</th>
<th>Bias term</th>
<th>-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-3_lemma_the</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-2_lemma_Seminar</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1_lemma_at</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>+1_lemma_4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>+1_Digit</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>+2_timeid</td>
<td>1</td>
</tr>
</tbody>
</table>
## Dot Product - II

<table>
<thead>
<tr>
<th></th>
<th>Bias term</th>
<th>-3_lemma_the</th>
<th>-2_lemma_Seminar</th>
<th>-1_lemma_at</th>
<th>+1_lemma_4</th>
<th>+1_Digit</th>
<th>+2_timeid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>-2</th>
<th>1*+2</th>
<th>1*+2</th>
</tr>
</thead>
</table>
Learning the Weight Vector

• The general learning task is simply to find a good weight vector!
  • This is sometimes also called "training"

• Basic intuition: you can check weight vector candidates to see how well they classify the training data
  • Better weights vectors get more of the training data right

• So we need some way to make (smart) changes to the weight vector
  • The goal is to make better decisions on the training data

• I will talk more about this later
Feature Extraction

• We run **feature extraction** to get the feature vectors for each position in the text
• We typically use a text representation to represent true values (which are sparse)
• Often we define **feature templates** which describe the feature to be extracted and give the name of the feature (i.e., -1_lemma_ XXX)

-3_lemma_the -2_lemma_Seminar -1_lemma_at +1_lemma_4 +1_Digit +2_timeid STIME
-3_lemma_Seminar -2_lemma_at -1_lemma_4 -1_Digit +1_timeid +2_lemma_will NONE

...
Training vs. Testing

• When training the system, we have gold standard labels (see previous slide)
• When testing the system on new data, we have no gold standard
  • We run the same feature extraction first
  • Then we take the dot product with the weight vector to get a classification decision
• Finally, we have to go back to the original text to write the <stime> tags into the correct positions
Summary so far

• So we've seen training and testing
• We have an idea about train error and test error (key concepts!)
• We are aware of the problem of overfitting
  • And we know what overfitting means in terms of train error and test error!

• Now let's compare decision trees and linear models
Linear models are weaker

• Linear models are weaker than decision trees
  • This means they can't express the same richness of decisions as decision trees can (if both have access to the same features)
• It is easy to see this by extending our example
• Recall that we have a weight vector encoding our rule (see next slide)
• Let's take another reasonable rule
... the Seminar at `<stime>` 4 pm will ...

<table>
<thead>
<tr>
<th>Position</th>
<th>Condition</th>
<th>Context-independent features</th>
<th>Context Dep.</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>the</td>
<td>the, lowercase</td>
<td>Art</td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>Seminar</td>
<td>seminar, uppercase</td>
<td>Noun</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>at</td>
<td>at, lowercase</td>
<td>Prep</td>
<td>stime</td>
</tr>
<tr>
<td>+1</td>
<td>4</td>
<td>4, lowercase</td>
<td>Digit</td>
<td></td>
</tr>
<tr>
<td>+2</td>
<td>pm</td>
<td>pm, lowercase</td>
<td>timeid</td>
<td>Other</td>
</tr>
<tr>
<td>+3</td>
<td>will</td>
<td>will, lowercase</td>
<td>Verb</td>
<td></td>
</tr>
</tbody>
</table>

Example modified from Ciravegna 2009
... the Seminar at `<stime>` 4 pm will ...

<table>
<thead>
<tr>
<th>Position</th>
<th>Condition</th>
<th>Context-independent features</th>
<th>Context Dep.</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Word Lemma Capitalization SemCat POS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td>at</td>
<td></td>
<td>stime</td>
</tr>
<tr>
<td>+1</td>
<td></td>
<td></td>
<td></td>
<td>Digit</td>
</tr>
<tr>
<td>+2</td>
<td></td>
<td></td>
<td></td>
<td>timeid</td>
</tr>
<tr>
<td>+3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• The rule we'd like to learn is that if we have the features:
  -2_lemma_seminar
  -1_lemma_at
  +1_Digit
• We should insert a <stime>
• This is quite a reasonable rule, it lets us correctly cover the new sentence:
  "The Seminar at 3 will be given by ..."
  (there is no timeid like "pm" here!)
• Let's modify the weight vector
Adding the second rule

<table>
<thead>
<tr>
<th></th>
<th>Bias term</th>
<th>-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>-2</td>
</tr>
<tr>
<td>0</td>
<td>-3_lemma_the</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-2_lemma_Seminar</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1_lemma_at</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>+1_lemma_4</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>+1_Digit</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>+2_timeid</td>
<td>1</td>
</tr>
</tbody>
</table>
• Let's first verify that both rules work with this weight vector
• But does anyone see any issues here?
How many rules?

• If we look back at the vector, we see that we have actually encoded quite a number of rules
  • Any combination of three features with ones will be sufficient so that we have a <stime>
  • This might be good (i.e., it might generalize well to other examples). Or it might not.
• But what is definitely true is that it would be easy to create a decision tree that only encodes exactly our two rules!
• This should give you an intuition as to how linear models are weaker than decision trees
• Linear models are used heavily in NLP exactly because they are weaker, since being weaker means they have less problems with overfitting
  • This is particularly important in NLP problems because often NLP researchers like to use a very large number of features (which might lead to really huge decision trees)
How can we get this power in linear models?

• Change the features!
• For instance, we can create combinations of our old features as new features
• For instance, clearly if we have:
  • One feature to encode our first rule
  • Another feature to encode our second rule
  • And we set the bias to 0
• We now get the same as the decision tree
• Sometimes these new compound features would be referred to as trigrams (they each combine three basic features)
Feature Selection

• A task which includes automatically finding such new compound features is called **feature selection**
  • This is built into some machine learning toolkits
  • Or you can implement it yourself by trying out feature combinations and checking the training error
    • Use human intuition to check a small number of combinations
    • Or do it automatically, using a script
Training

Training is **automatically adjusting** the feature vector so as to better fit the training corpus! **Intuition:** make small adjustments to get a better score on the training data (these all fit our example!)

<table>
<thead>
<tr>
<th>-2</th>
<th>-2.01</th>
<th>-1.99</th>
<th>-2.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.04</td>
<td>0.04</td>
<td>0.043</td>
</tr>
<tr>
<td>0</td>
<td>0.0004</td>
<td>0.002</td>
<td>0.0003</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1.1</td>
<td>1.101</td>
<td>1.1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.9001</td>
<td>0.9111</td>
<td>0.9144</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.89</td>
<td>0.892</td>
<td>0.93</td>
</tr>
<tr>
<td>1</td>
<td>0.91</td>
<td>0.91</td>
<td>1.01</td>
</tr>
</tbody>
</table>
Perceptron Update I

- One way to do this is using a so-called perceptron

- Algorithm:
  - Read the training examples one at a time
  - For each training example, decide how to update the weight vector
  - The perceptron update rule says:
    - If a training example is classified correctly:
      - Do nothing (because the current weight vector is fine)
    - If a training example is classified incorrectly:
      - Adjust the weight of every active feature by a small amount towards the desired decision
      - So that the example will score a bit better next time it is observed

- Intuition: we hope that by making many small changes
  - The weights on important features increase consistently to the desired values which work well on the entire training set
  - The changes to unimportant feature weights will be random (sometimes up, sometimes down), and the weights will tend towards zero (meaning: no effect on the classification)
Perceptron Update II

Say we have \(-2 0 0 0 \ldots 0 0 0 0.5\), and see this training example. Clearly we will get it wrong...

\[
\begin{align*}
1 & \quad \text{Bias term} & -2 & \quad 1 \times -2 & \quad -2 \\
0 & \quad & 0 & \quad & 0 \\
1 & \quad -3 \_lemma\_the & 0 & \quad & 0 \\
0 & \quad & 0 & \quad & 0 \\
1 & \quad -2 \_lemma\_Seminar & 0 & \quad & 0 \\
0 & \quad & 0 & \quad & 0 \\
0 & \quad & 0 & \quad & 0 \\
1 & \quad -1 \_lemma\_at & 0 & \quad & 0 \\
1 & \quad +1 \_lemma\_4 & 0 & \quad & 0 \\
0 & \quad & 0 & \quad & 0 \\
1 & \quad +1 \_Digit & 0.5 & \quad & 0.5 \\
1 & \quad +2 \_timeid & 1 \times 0.5 & \quad & 0.5 \\
\end{align*}
\]
Perceptron Update III

So change the weight vector, by adding 0.1 to all active features. Score is now better (but still still wrong)

\[
\begin{array}{c|c|c|c}
& \text{Bias term} & -1.9 & 1*-1.9 & -1.9 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
+2_{\text{timeid}} \quad 0.6 \\
\]

-0.8
After looking at many other examples, irrelevant features (like "-3_lemma_the") are pushed back towards zero, and important features have stronger weights. We have learned a good weight vector for this example, no further update is needed.

<table>
<thead>
<tr>
<th></th>
<th>Bias term</th>
<th>-3_lemma_the</th>
<th>-2_lemma_Seminar</th>
<th>-1_lemma_at</th>
<th>+1_lemma_4</th>
<th>+1_Digit</th>
<th>+2_timeid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>-2.1</td>
<td>-0.1</td>
<td>0.1</td>
<td>0.7</td>
<td>1.1</td>
<td>1.2</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>1</td>
<td>-3_lemma_the</td>
<td>1*-2.1</td>
<td>1*-0.1</td>
<td>1*0.1</td>
<td>1*0.7</td>
<td>1*1.1</td>
<td>1*1.2</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>-2.1</td>
<td>-0.1</td>
<td>0.1</td>
<td>0.7</td>
<td>1.1</td>
<td>1.2</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>-2.1</td>
<td>-0.1</td>
<td>0.1</td>
<td>0.7</td>
<td>1.1</td>
<td>1.2</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>+1_lemma_4</td>
<td>1*2.1</td>
<td></td>
<td>1*0.1</td>
<td>1*0.7</td>
<td>1*1.1</td>
<td>1*1.2</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>-2.1</td>
<td></td>
<td>0.1</td>
<td>0.7</td>
<td>1.1</td>
<td>1.2</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Word embeddings

- Word embeddings such as the popular word2vec embeddings are a clever way to get better features
  - Word embeddings are learned on huge amounts of text
  - Details in next week’s lecture
- Word-types are represented as positions in a 50-dimensional space
  - For each word-type, we look up its embedding in a table
- Similar words are close to each other in this space, for instance:
  - AM and PM (words for which SemCat=timeid) will have very similar representations
  - Different words with the same lemma will have very similar representations
- So when using word embeddings, we do not need the context-independent features
  - And the embedding space captures many generalizations about word-types that we didn’t actively know would help!
  - These generalizations become available to the learner, which can choose to use them if they are helpful for learning the training data
... the Seminar at <stime> 4 pm will ...

<table>
<thead>
<tr>
<th>Position</th>
<th>Condition</th>
<th>50-dimen. word-type embeddings (only 3 dimensions shown)</th>
<th>Context Dep.</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Word</td>
<td>Dim 1</td>
<td>Dim 2</td>
<td>Dim 3 ...</td>
</tr>
<tr>
<td>-3</td>
<td>the</td>
<td>-0.234</td>
<td>0.155</td>
<td>0.001</td>
</tr>
<tr>
<td>-2</td>
<td>Seminar</td>
<td>0.555</td>
<td>0.888</td>
<td>0.002</td>
</tr>
<tr>
<td>-1</td>
<td>at</td>
<td>-0.165</td>
<td>-0.122</td>
<td>0.001</td>
</tr>
<tr>
<td>+1</td>
<td>4</td>
<td>0.122</td>
<td>0.095</td>
<td>-0.003</td>
</tr>
<tr>
<td>+2</td>
<td>pm</td>
<td>0.001</td>
<td>0.001</td>
<td>0.999</td>
</tr>
<tr>
<td>+3</td>
<td>will</td>
<td>-0.812</td>
<td>0.201</td>
<td>0.002</td>
</tr>
</tbody>
</table>
Contextualized embeddings

• Contextualized word embeddings allow us to get a different representation of each word token, rather than word-type
  • The entire sentence is used as context
  • Some popular contextualized embeddings are ELMO and BERT
• Contextualized word embeddings capture the same information as word-type embeddings
• But they additionally capture features that are context-dependent
• Makes many more generalizations available to the learner!
  • Part-of-Speech (POS) distinctions will be accessible (as in our example)
  • Polysemy, tokens of a word-type with the same word sense will have similar embeddings
  • Syntactic positions will be captured (e.g., Subject, Verb, Object)
  • Semantic roles will also be captured (e.g., Agent, Patient in a passive sentence)
  • Etc.
• Typically something like 400 dimensional vectors for each word token
  • Input for computing the word-token embeddings is the entire sentence
Two classes

- So far we discussed how to deal with a single label
  - At each position between two words we are asking whether there is a `<stime>` tag
- This is called **binary classification**
- However, we are interested in `<stime>` and `</stime>` tags
- How can we deal with this?
- We can simply train one classifier on the `<stime>` prediction task
  - Here we are treating `</stime>` positions like every other non `<stime>` position
- And train another classifier on the `</stime>` prediction task
  - Likewise, treating `<stime>` positions like every other non `</stime>` position
- If both classifiers predict "true" for a single position, take the one that has the highest dot product
More than two labels

• We can generalize this idea to many possible labels
• This is called **multiclass classification**
  • We are picking one label (class) from a set of classes
• For instance, maybe we are also interested in the `<etime>` and `</etime>` labels
  • These labels indicate seminar end times, which are also often in the announcement emails (see next slide)
Abstract:

This Monday, 4/26, Prof. Makoto Nagao will give a seminar in the CMT red conference room on recent MT research results.
One against all

- We can generalize the way we handled two binary classification decisions to many labels
- Let's add the `<etime>` and `</etime>` labels
- We can train a classifier for each tag
  - Just as before, every position that is not an `<etime>` is a negative example for the `<etime>` classifier, and likewise for `</etime>`
- If multiple classifiers say "true", take the classifier with the highest dot product
- This is called one-against-all
- It is a quite reasonable way to use binary classification to predict one of multiple classes
  - It is not the only option, but it is easy to understand (and to implement too!)
Summary: Multiclass classification

- We discussed **one-against-all**, a framework for combining binary classifiers.
- It is not the only way to do this, but it often works pretty well.
  - There are also techniques involving building classifiers on different subsets of the data and voting for classes.
  - And other techniques can involve, e.g., a sequence of classification decisions (for instance, a tree-like structure of classifications).
As we saw a few lectures ago, we can detect seminar start times by using two binary classifiers:

- One for `<stime>`
- One for `</stime>`

And recall that if they both say "true" to the same position, take the highest dot product
• Then we need to actually annotate the document
• But this is problematic...
Some concerns
A basic approach

• One way to deal with this is to use a greedy algorithm
• Loop:
  • Scan the document until the <stime> classifier says true
  • Then scan the document until the </stime> classifier says true
• If the last tag inserted was <stime> then insert a </stime> at the end of the document
• Naturally, there are smarter algorithms than this that will do a little better
• But the major problem here is more basic.
  • Relying on these two independent classifiers is not optimal!
How can we deal better with sequences?

• We can make our classification decisions dependent on previous classification decisions

• For instance, think of the Hidden Markov Model as used in POS-tagging

• The probability of a verb increases after a noun
Basic Sequence Classification

• We will do the following
  • We will add a feature template into each classification decision representing the previous classification decision
  • And we will change the labels we are predicting, so that in the span between a start and end boundary we are predicting a different label than outside
The basic idea is that we want to use the previous classification decision.
We add a special feature template -1_label_XXX.
For instance, between 4 and pm, we have:
-1_label_<stime>

Suppose we have learned reasonable classifiers.
How often should we get a <stime> classification here? (Think about the training data in this sort of position)
-1_label_<stime>

- This should be an extremely strong indicator not to annotate a <stime>

- What else should it indicate?
  - It should indicate that there must be either a in-stime or a </stime> here!
Changing the problem slightly

- We'll now change the problem to a problem of annotating tokens (rather than annotating boundaries)
- This is traditional in IE, and you'll see that it is slightly more powerful than the boundary style of annotation
- We also make less decisions (see next slide)
• This is called IOB markup (or BIO = begin-in-out)
• This is a standardly used markup when modeling IE problems as sequence classification problems

• We can use a variety of models to solve this problem
• One popular model is the Hidden Markov Model, which you have seen in Statistical Methods
  • There, the label is the state
• However, in this course we will (mostly) stay more general and talk about binary classifiers and one-against-all
(Greedy) classification with IOB

Seminar  at  4  pm  will  be  on  ...
O    O  B-stime  I-stime  O  O  O

- To perform greedy classification, first run your classifier on "Seminar"
- You can use a label feature here like -1_Label_StartOfSentence
- Suppose you correctly choose "O"
- Then when classifying "at", use the feature: -1_Label_O
- Suppose you correctly choose "O"
- Then when classifying "4", use the feature: -1_Label_O
- Suppose you correctly choose "B-stime"
- Then when classifying "pm", use the feature: -1_Label_B-stime
- Etc...
Training

• How to create the training data (do feature extraction) should be obvious
  • We can just use the gold standard label of the previous position as our feature
BIEWO Markup

• A popular alternative to IOB markup is BIEWO markup
• E stands for "end"
• W stands for "whole", meaning we have a one-word entity (i.e., this position is both the begin and end)

Seminar at 4 pm will be on ...
O O B-stime E-stime O O O

Seminar at 4 will be on ...
O O W-stime O O O
BIEWO vs IOB

- BIEWO fragments the training data
  - Recall that we are learning a binary classifier for each label
  - In our two examples on the previous slide, this means we are not using the same classifiers!
- Use BIEWO when single-word mentions require different features to be active than the first word of a multi-word mention
Conclusion

• I've taught you the basics of:
  • Binary classification using features
    • I also briefly presented word-type embeddings (word2vec) and contextualized word-token embeddings (e.g., BERT, ELMO)
  • Multiclass classification (using one-against-all)
  • Sequence classification (using a feature that uses the previous decision)
    • And IOB or BIEWO labels
• I've skipped a lot of details
  • I haven't talked about non-greedy ways to do sequence classification
  • And I didn't talk about probabilities, which are used directly, or at least approximated, in many kinds of commonly used linear models!
• Hopefully what I did tell you is fairly intuitive and helps you understand classification, that is the goal
• Further reading:
• More advanced, highly recommended:
  • Hal Daumé III. A Course in Machine Learning. 2017 (beta version 0.99, free, or 1.0, not free)
• Word embeddings (including word2vec, ELMO, BERT):
• Thank you for your attention!