Chunking—Practical Exercise

Chunking for NER

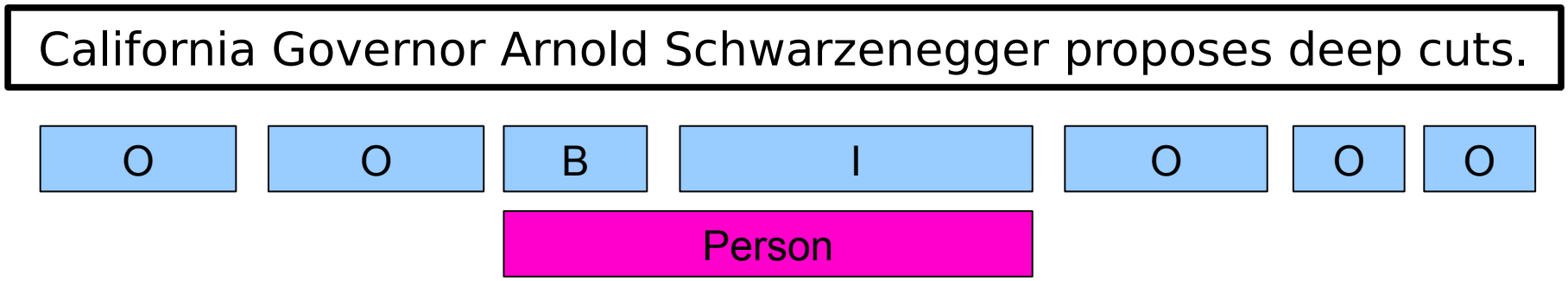
- Chunking, as we saw at the beginning, means finding parts of text
- This task is often called Named Entity Recognition (NER), in the context of finding person and organization names
- The same principle can be applied to any task that involves finding where things are located in text
- For example, finding the noun phrases
- Can you think of any others?

California Governor Arnold Schwarzenegger proposes deep cuts.

Person

Chunking for NER

- It's implemented as a twist on classification (everything is classification under the hood!)
- We achieve this in the Learning Framework by identifying which tokens are the beginning of a mention, which are the insides and which are the outsides (“BIO”)
  - There are other schemes; the old Batch Learning PR used BE (beginnings and ends)
- You don't need to worry about the Bs, Is and Os; the Learning Framework will take care of all that for you! You just need a corpus annotated with entities



Chunking—Practical Exercise

- Materials for this exercise are in the folder called “chunking-hands-on”
- You might want to start by closing any applications and corpora from the previous exercise, so we have a fresh start

Finding Person Mentions using Chunking Training and Application PRs

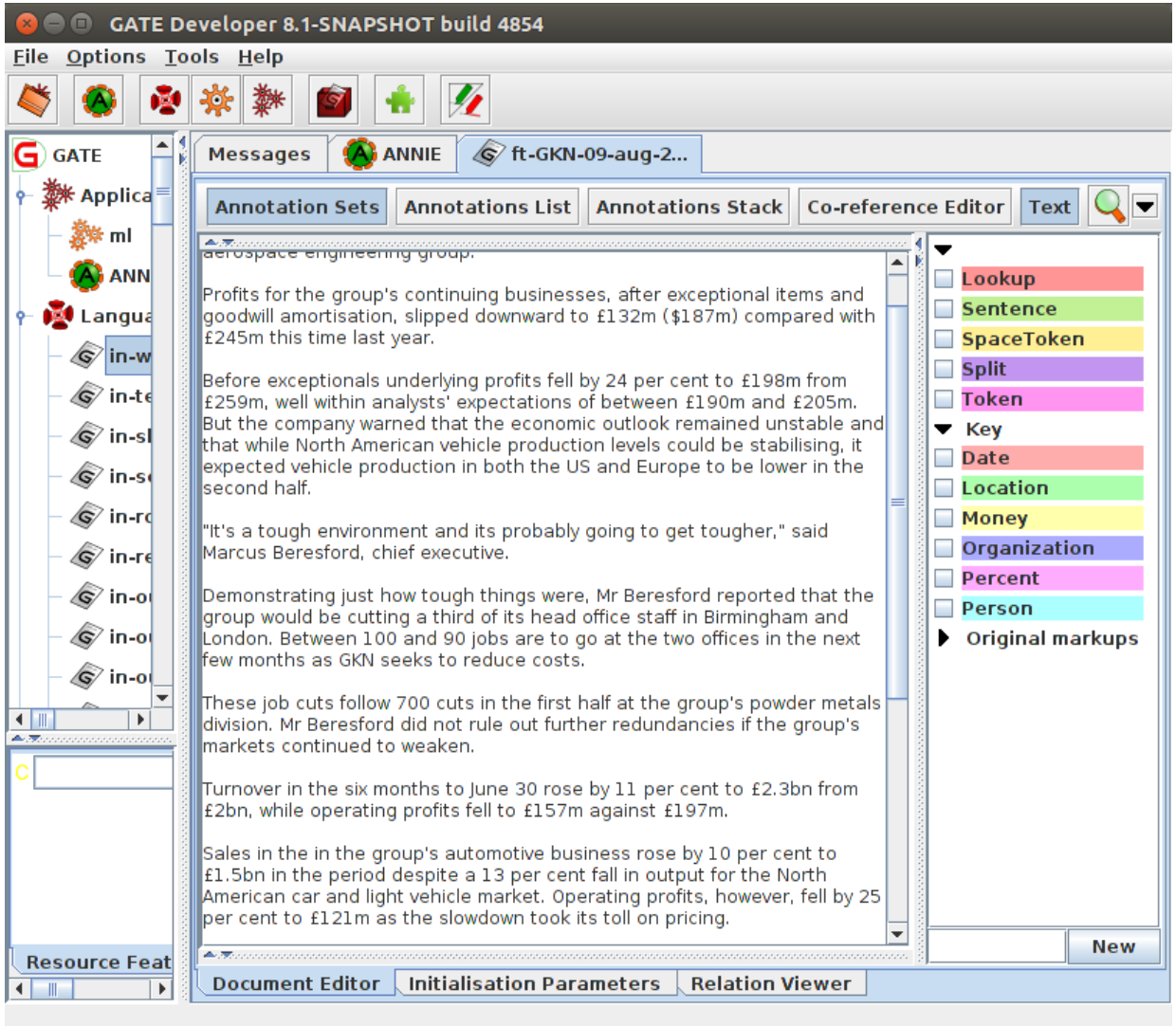
Load the corpus

- Create corpora for training and testing, with sensible names
- Populate them from the training and testing corpora you have in your chunking hands on materials
- Open a document and examine its annotations

Examining the corpus

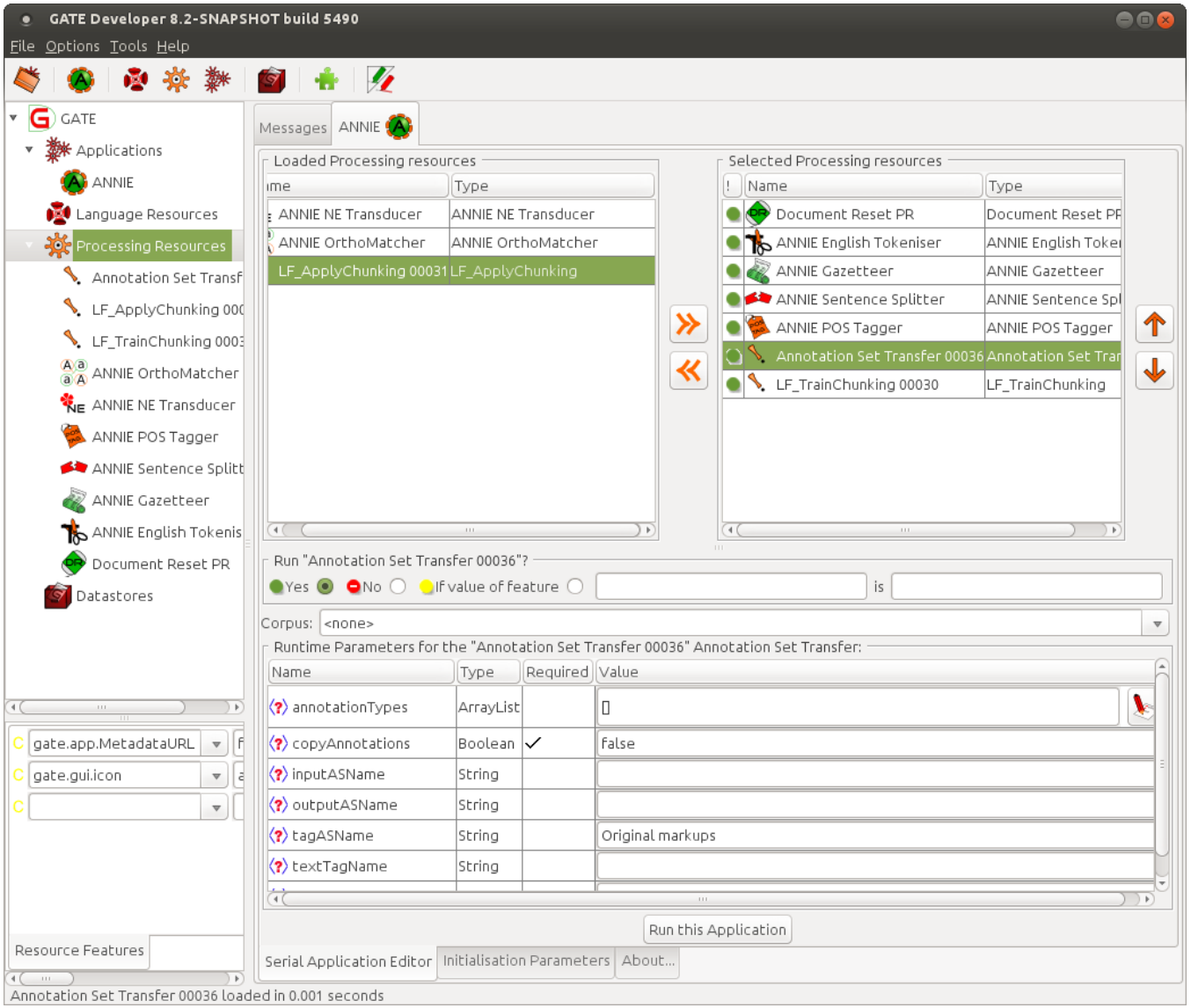
- The corpus contains an annotation set called “Key”, which has been manually prepared
- Within this annotation set are annotations of types “Date”, “Location”, “Money”, “Organization” and so forth

Creating the application



- As previously, if we run ANNIE on the corpus, we have more annotations to work with
- So start by loading ANNIE as the basis for your application
- Again, we don't need the NE transducer or orthomatcher

NER GATE application

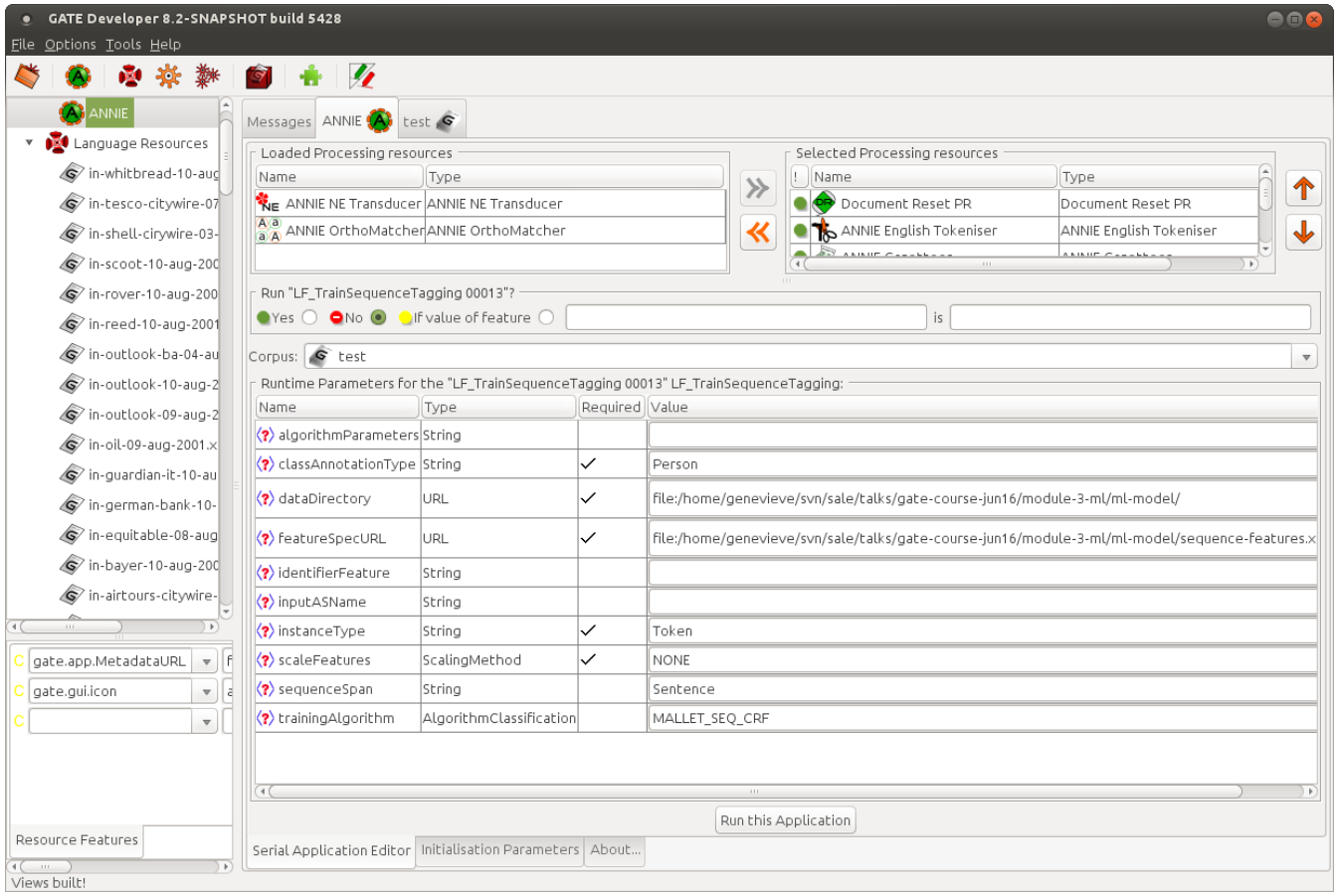


- Again, we need an Annotation Set Transfer, so create and add one
- Then create both training and application chunking PRs
- Start by just adding the training one

Annotation Set Transfer

- We'll use the annotation set transfer to copy the Person annotations up to the default annotation set, where we can learn them
- Go ahead and set up your AST now
- Be sure to copy them, not move them!

Chunking training parameters



- Let's look at the parameters for the training PR
- Instead of targetFeature, we have classAnnotationType

Chunking training parameters

- For classification, the class to learn is in a feature on the instance, is specified to the PR in the targetFeature parameter
- For chunking, the class to learn takes the form of an annotation type. In our case, our corpus is annotated with Person annotations that we are going to learn to locate
- This type to learn is indicated in the classAnnotationType parameter

Chunking training parameters

- Set the classAnnotationType now
- Set the dataDirectory to where you want to save your model, and set the featureSpecURL (there's a feature spec to get you started in the hands on materials)
- Set instanceType. What do you think it should be?

Sequence Spans

- sequenceSpan is only relevant when using sequence learners
- Sequence learners classify each instance in the span by making use of the others
- For example, a noun phrase might be more likely to follow a determiner than a preposition, or a person name might be more likely to follow the word “Mrs”
- The Learning Framework offers the Conditional Random Fields sequence learner
- It might be good for finding Persons, so let's use it!
  - You don't have to use a sequence learner for chunking though
- What do you think would be a good sequence span?

Sequence Spans

- Sequence spans should be spans within which instance classes follow patterns
  - For example, grammatical rules apply to sequences of parts of speech
  - However, sentiment classifications of individual customer reviews don't form a meaningful sequence
- A sequence span shouldn't be longer than necessary
- Sentence would be a good span for our task
- Fortunately, ANNIE creates sentence annotations for us, so those are available to use
- Set sequenceSpan to “Sentence”

Feature Specification

```
<ML-CONFIG>

<ATTRIBUTE>
<TYPE>Token</TYPE>
<FEATURE>category</FEATURE>
<DATATYPE>nominal</DATATYPE>
</ATTRIBUTE>

<ATTRIBUTE>
<TYPE>Token</TYPE>
<FEATURE>kind</FEATURE>
<DATATYPE>nominal</DATATYPE>
</ATTRIBUTE>

<ATTRIBUTE>
<TYPE>Token</TYPE>
<FEATURE>length</FEATURE>
<DATATYPE>numeric</DATATYPE>
</ATTRIBUTE>

<ATTRIBUTE>
<TYPE>Token</TYPE>
<FEATURE>orth</FEATURE>
<DATATYPE>nominal</DATATYPE>
</ATTRIBUTE>

<ATTRIBUTE>
<TYPE>Token</TYPE>
<FEATURE>string</FEATURE>
<DATATYPE>nominal</DATATYPE>
</ATTRIBUTE>

</ML-CONFIG>
```

- For this task, we are using attribute features
- These allow us to take features from the instance annotations or others that are co-located with them
- We specify type, feature and datatype
- Attribute features also can be taken from instances nearby
- That's a bit less useful with a sequence learner though—why?

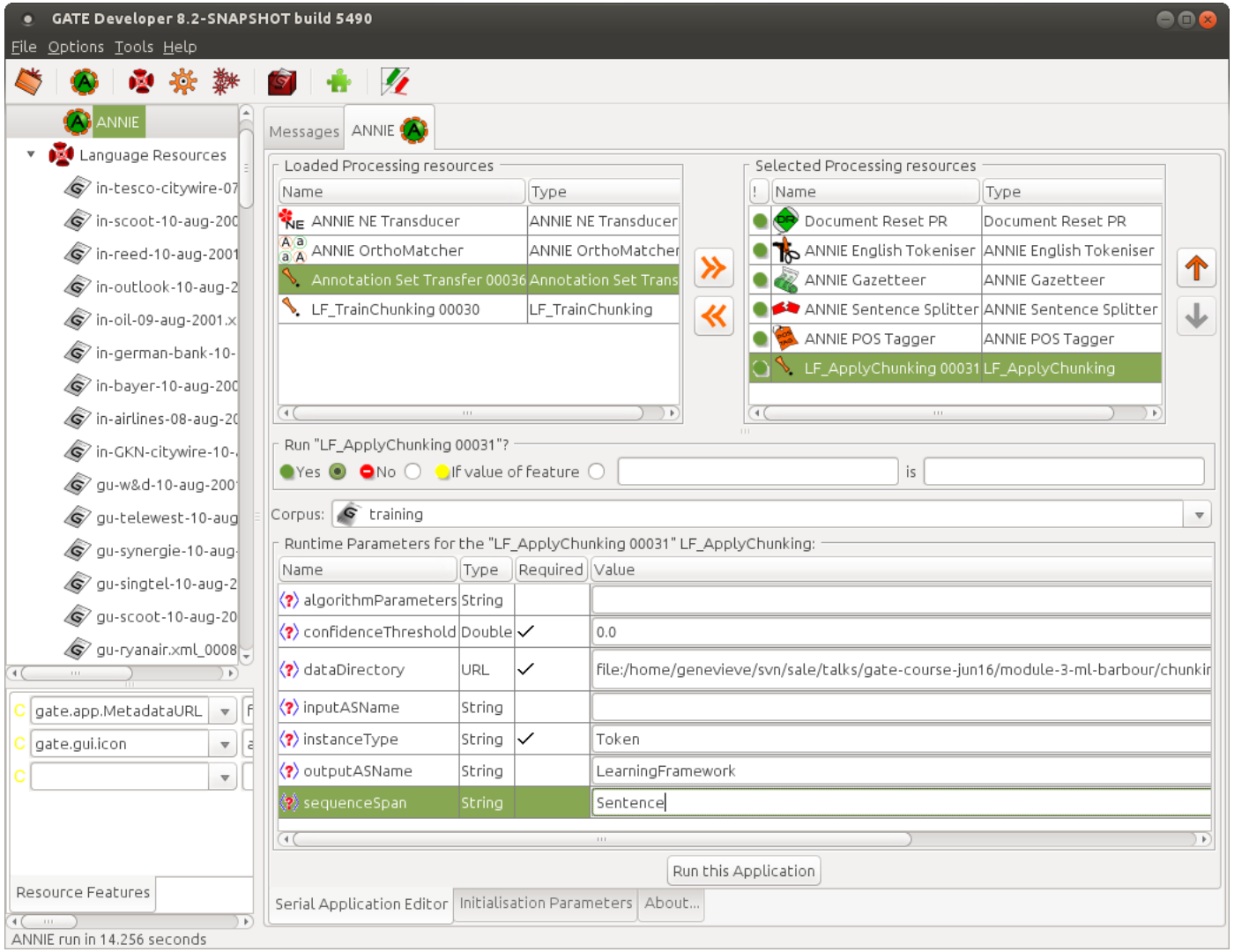
Training

- Make sure you have selected the training corpus
- Run the application!

Chunking application parameters

- Now move the training PR out of the application and add the application PR
- You can take the annotation set transfer out too
- It doesn't have a targetFeature parameter like the classification application PR did
- You don't need to tell it what type to create because the model knows it from training!
- Set dataDirectory to the location where you told the training PR to put the model
- Set the sequence span

Applying



- Now run this on the test corpus

Chunking—Evaluation using Corpus QA

Chunking Evaluation

- For classification, each response is simply right or wrong
- For NER, there are more ways to be wrong
  - Fewer or more mentions than there really are, or you can overlap
- So we need different metrics

What are precision, recall and F1?

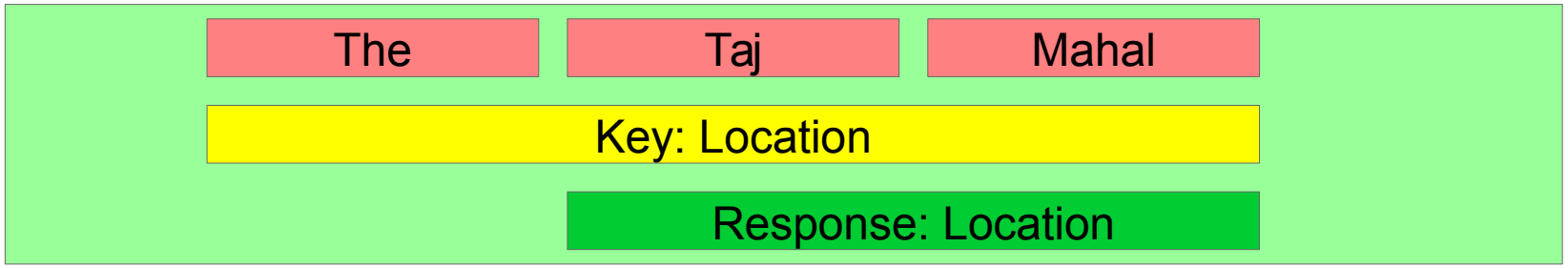
- Precision: what proportion of our automatic annotations were correct?
- Recall: what proportion of the correct annotations did our automatic tool create?
- P = correct / (correct + spurious) = tp / (tp + fp)
- R = correct / (correct + missing) = tp / (tp + fn)
- where tp = true positives, fp = false positives, fn = false negatives

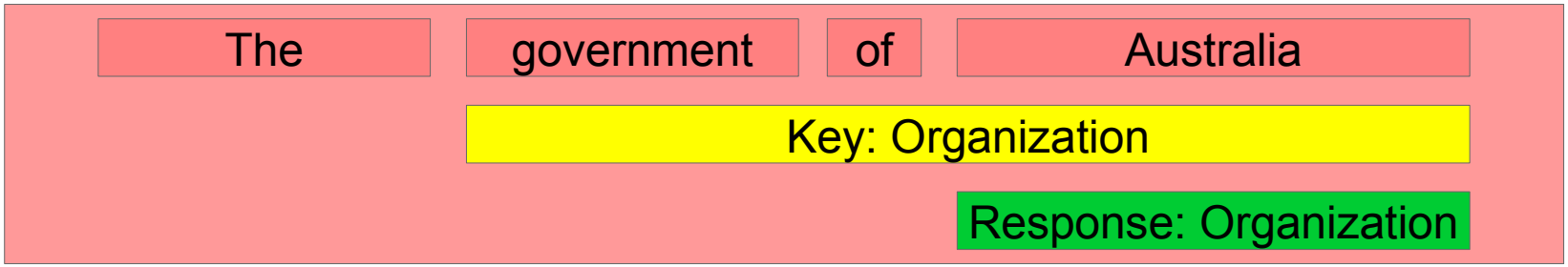
What are precision, recall and F1?

- F-score is an amalgam of the two measures
  - $F_\beta = (1+\beta^2)PR / (\beta^2 P + R)$
  - The equally balanced F1 ($\beta = 1$) is the most common F-measure
  - $F1 = 2PR / (P + R)$

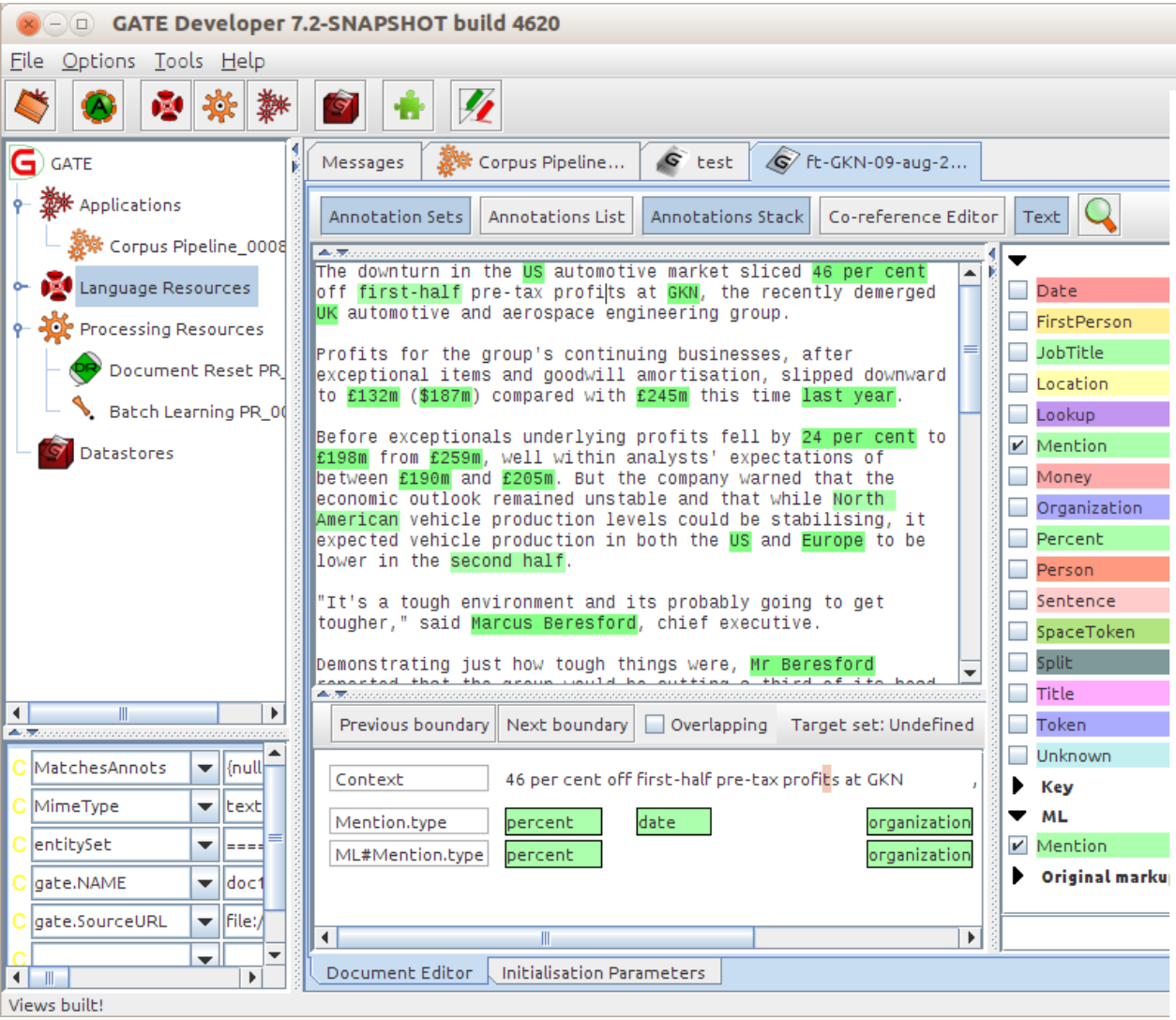
Strict and Lenient

- “Strict” means we count an annotation as correct only if it has the same span as the gold standard annotation
- Lenient means we allow an annotation that overlaps to be correct, even if it isn't a perfect span match
- Which do you think is the right way to do it?



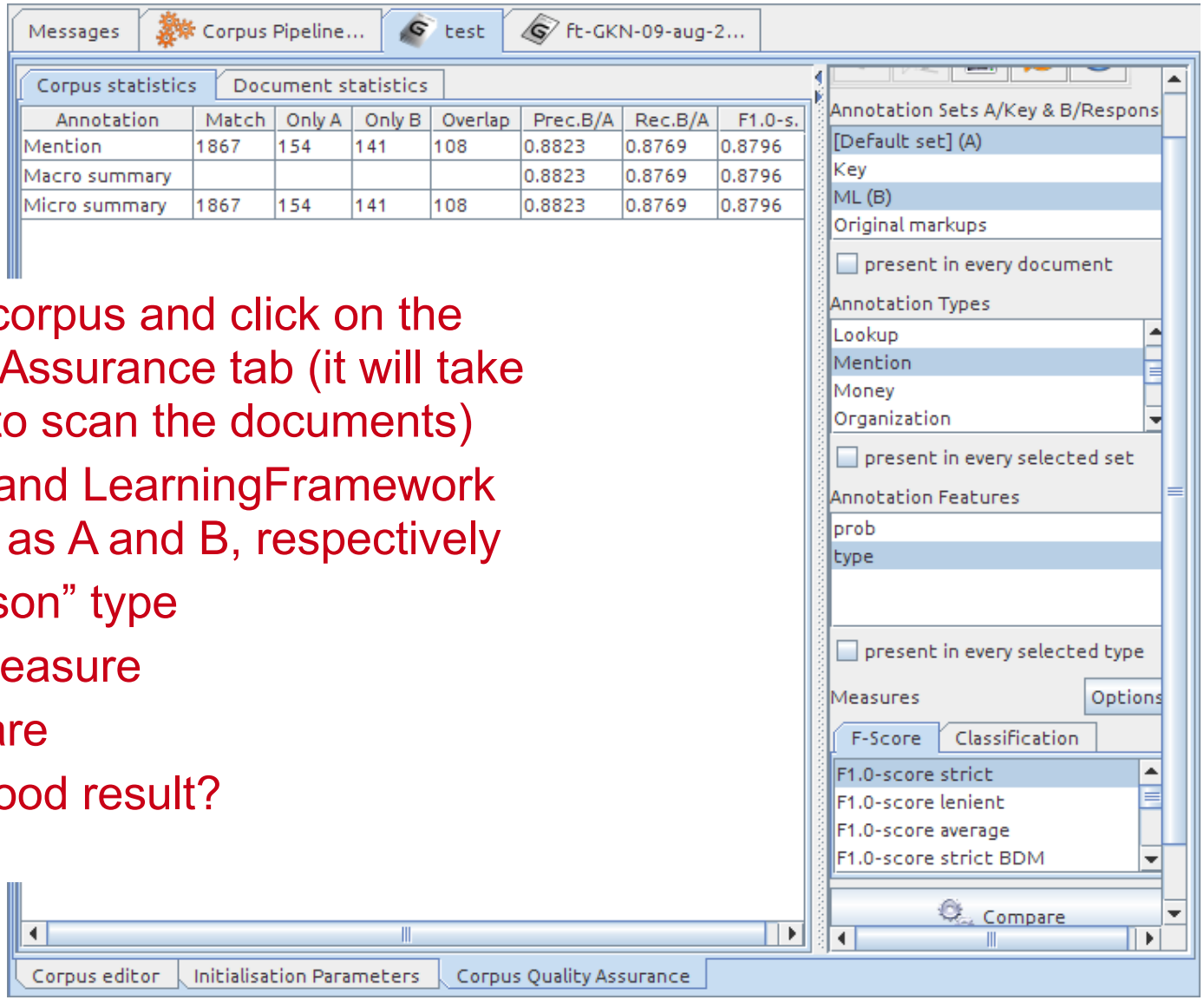


Examining the results of application



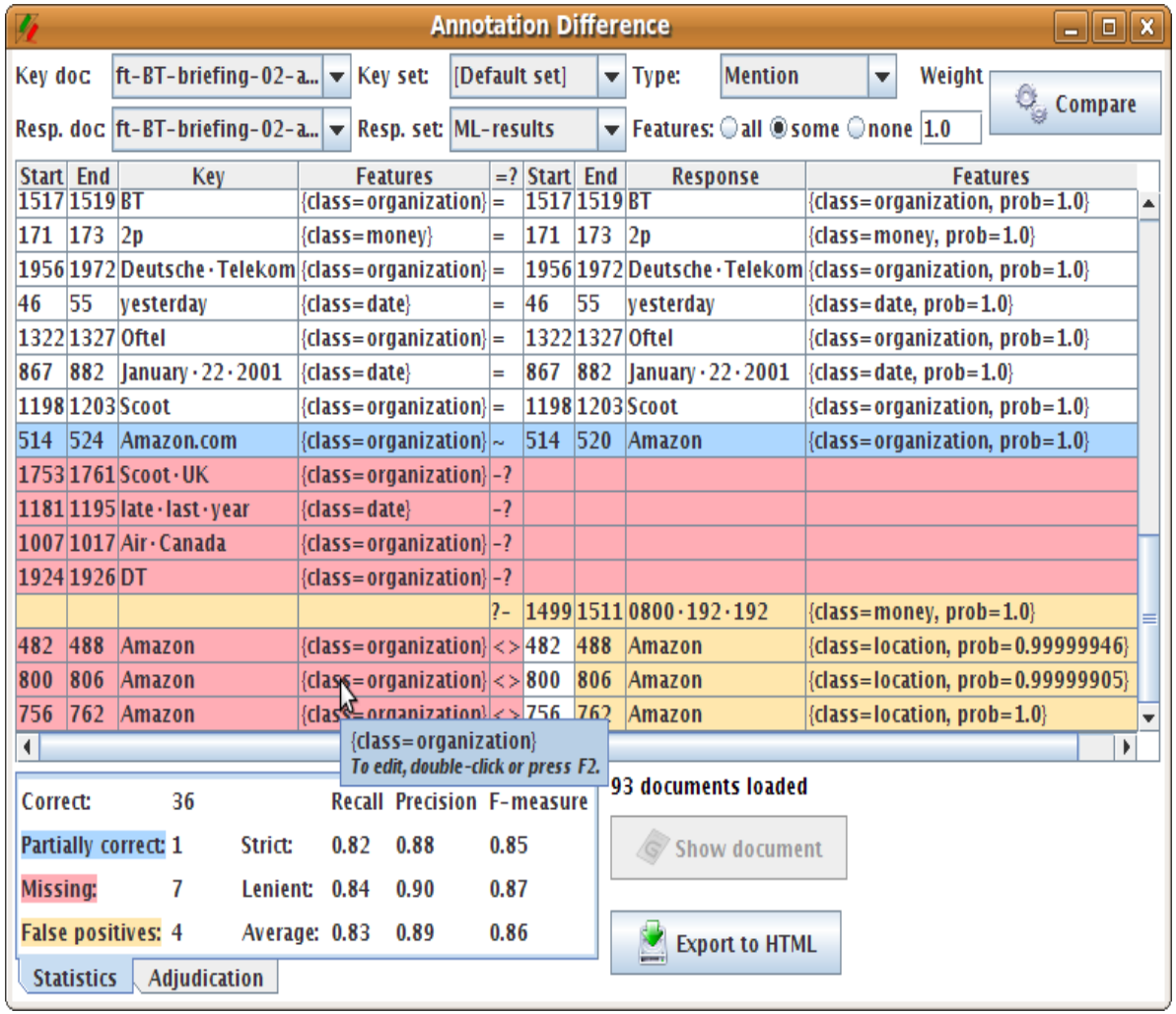
- **Examine a document from the test corpus**
- You should have a new “LearningFramework” AS with Person annotations
- The original Person annotations (in the Key AS) are similar but not always identical!
- The Annotations Stack is good for comparing them
- How similar do they appear to be? Do you think you will get a good result?

Comparing the Sets with Corpus QA



- Select the test corpus and click on the Corpus Quality Assurance tab (it will take a few seconds to scan the documents)
- Select the Key and LearningFramework annotation sets as A and B, respectively
- Select the “Person” type
- Choose an F-measure
- Click on Compare
- Did you get a good result?

Using Annotation Diff to examine performance



- **Switch to the "Document statistics" tab**
- **Choose a document**
- **Click on the Annotation Diff icon**
- What kind of mistakes did your application make?

Using Annotation Diff...

- “Correct”: the response annotation has the right feature and span
- “Partially correct”: response has the right feature and overlapping but not exactly matched span; this counts as correct in the “lenient” scoring
- “Missing”: key annotation+feature is missing from the response (a.k.a. “false negative”)
- “False positive”: response annotation+feature shouldn't be there (a.k.a. “spurious”)

Classification Evaluation PR for Chunking?

- We didn't use a Learning Framework evaluation PR for this chunking task
- What do you think would happen if you used the Classification Evaluation PR to do a chunking problem?
- It would work! It would evaluate the accuracy of the system in correctly identifying beginnings, insides and outsides
- However, it wouldn't tell you much about how well you did finding named entities
  - There are so many outsides that you can get a high score just by saying everything is an outside!
- You could use it to tune parameters if you wanted, though

Exercise—Improving the result

- Again, see if you can improve your result
- Try different features and algorithms

Exercise 2

- Try to learn different entity types

Exporting Feature Data

Exporting feature data

- A GATE ML PR serves a number of functions
  - Scraping features off the documents and formulating them as ML training sets
  - Sending the training sets to ML libraries to train a model
  - Creating instances (without class) at apply time to send to a trained model to be classified and writing the resulting class back onto the application instance
- We have integrated quite a few algorithms and some ML facilitation technology, so many ML tasks can be accomplished entirely in GATE

Exporting feature data

- However, GATE isn't an ML tool—its forte and contribution is complex linguistic features. There is a limit to what we will include in the way of ML innovations.
- For example, the Learning Framework;
  - doesn't include feature selection technologies
  - includes only limited feature scaling
  - doesn't integrate all algorithm variants

Exporting feature data

- For more advanced needs, there are other ways to work
- You can export your training set and use it to train a model outside of GATE
  - The Learning Framework will allow you to use a model trained outside of GATE to create an application
- Exporting data and working in e.g. Weka can also provide a faster way to tune parameters
  - When you change parameters in the LF it starts over again scraping the features off the documents, which takes a long time on a big corpus
- You could use e.g. Weka's feature selection technology and bring what you learned back into GATE by editing your feature spec
- It can also be a good sanity check to see your data in export format

Export the data as ARFF

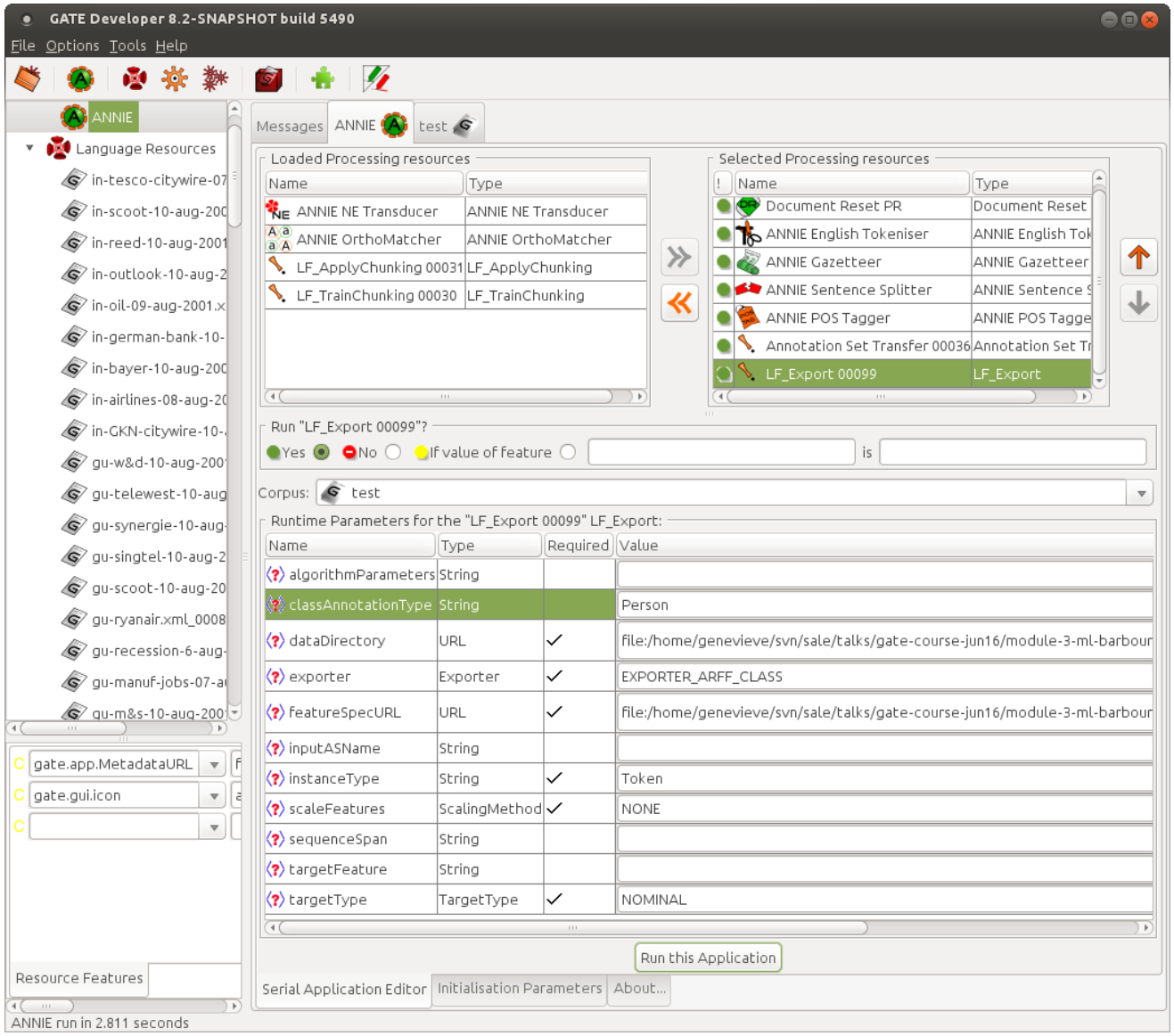
- Create an Export PR and add it to the application
- You can remove the other Learning Framework PRs
- Annotation Set Transfer needs to stay though

Export Parameters

- classAnnotationType is as for training, and its presence indicates that we are exporting a CHUNKING dataset—set it to Person
- dataDirectory, featureSpecURL, inputASName and instanceType you are familiar with by now—set them
- For exporter, choose EXPORTER_ARFF_CLASS*
- Don't set target feature! This would indicate that we want to export a classification dataset!
- Don't set sequenceSpan—this would indicate that we want to export data in a format suitable for training a sequence learner. This isn't supported yet.

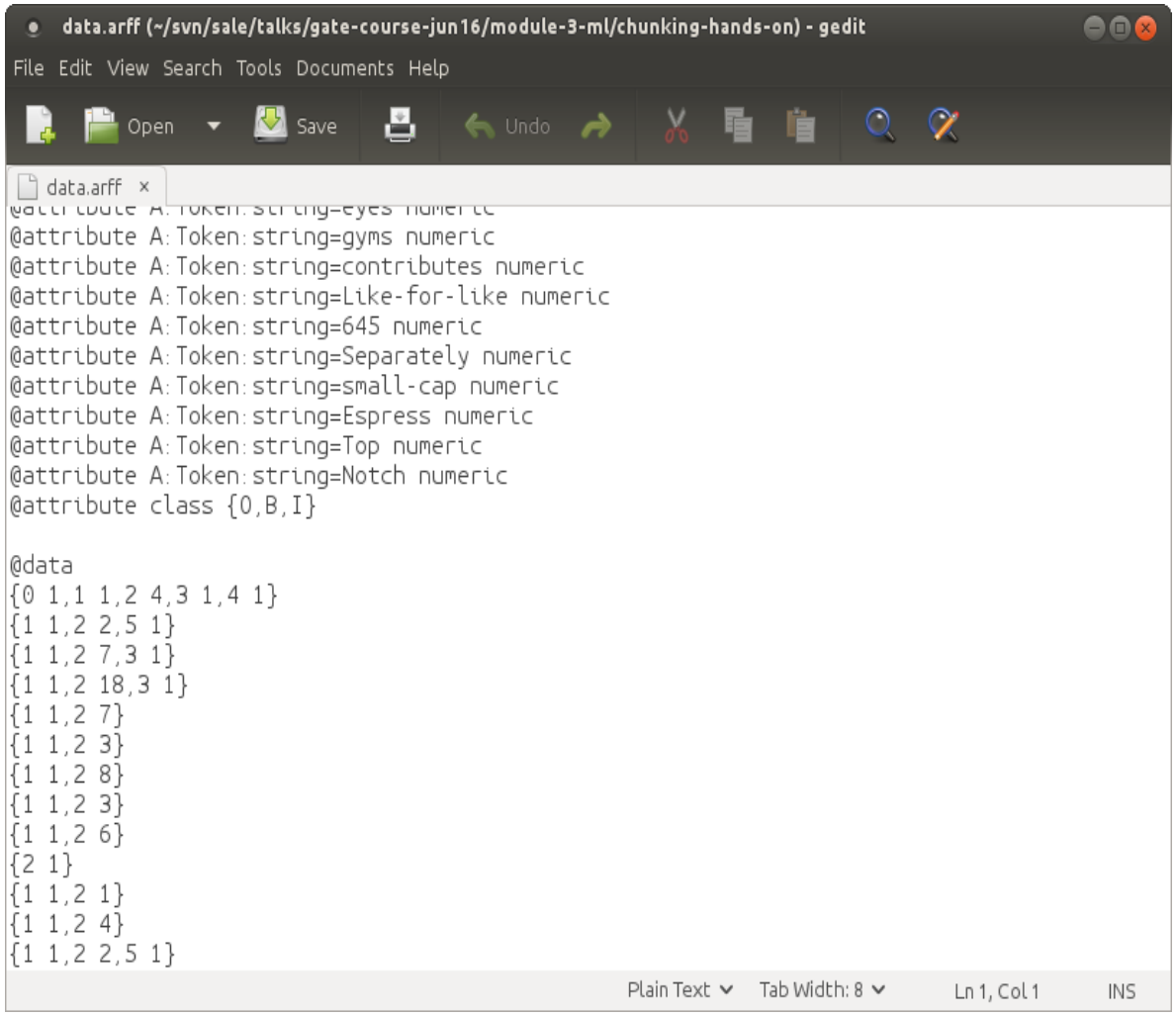
* ”CLASS” means classification—why are we exporting a classification dataset for a chunking problem? Because they're all classification behind the scenes. GATE turns the chunking problem into a classification problem for training and then turns it back again!

Exporting



- Set targetType to nominal, because beginnings, insides and outsides are nominal classes
- Go ahead and export the data!

Examining the ARFF



- You'll find your exported ARFF in your dataDirectory, called data.arff
- Examine it now
- At the top are a list of attributes. Are they as expected?
- The last attribute is the class attribute. Do you see it?
- After that come feature vectors in sparse format. How can you tell that they are in sparse format? What would this file look like if they were written out in full?

Working with Weka

Why would I want to use Weka?

- As noted previously, Weka can be faster and better for playing around with parameters to get the best result
  - Now that you have exported your data, you can try loading it into Weka in your own time, and see what you can do there
- But then you need to bring that result back into GATE! So you need to run the Weka algorithm in GATE
- Weka has some good algorithms that might be better for your task
  - Though note that Mallet's CRF is often the best for chunking, and LibSVM is often the best for most things, and you don't need Weka for those
- However, due to licensing incompatibility, we can't integrate Weka into GATE as seamlessly as we integrated LibSVM and Mallet

What you need

- Weka integration comes as a separate project, but it's easy to do!
- You need to get the Weka wrapper from here (downloading the zip is easiest):

  https://github.com/GateNLP/weka-wrapper/
- You need to tell your application where to find the Weka wrapper
  - Use the environment variable WEKA_WRAPPER_HOME
  - Or use the java property gate.plugin.learningframework.wekawrapper.home
  - Or the setting wekawrapper.home in a file weka.yaml in the data directory used

Using Weka in the GATE GUI

- Then you can go ahead and use Weka for classification and chunking by:
  - Creating a training PR
  - Selecting WEKA_CL_WRAPPER for trainingAlgorithm
  - Giving the full class name of the Weka algorithm as the first algorithmParameters argument
    - For example “weka.classifiers.trees.RandomForest”
  - A model will be created in the specified directory as before
  - At apply time, you simply indicate this model as usual
- (Weka in the evaluation PR isn't supported—try using Weka to evaluate!)

Where to find documentation about ...

- Getting the Weka wrapper and using it to train models outside of GATE:
  - https://github.com/GateNLP/weka-wrapper
- Using Weka inside of GATE:
  - https://github.com/GateNLP/gateplugin-LearningFramework/wiki/UsingWeka
- What Weka algorithms' full class names are:
  - Weka's Javadoc, e.g. http://weka.sourceforge.net/doc.dev/weka/classifiers/Classifier.html
- Note that the Weka wrapper is very new code! Let us know if you find any problems with it!