Classification—Practical Exercise

Classification—Practical Exercise

- Materials for this exercise are in the folder called "classification-hands-on"

Classification using Training and Application PRs

Load the corpus

- Create a corpus for testing and one for training (make sure you name them so you can tell which is which!)
- Populate them from classification-hands-on/test-corpus and classification-hands-on/training-corpus
- Open up one of the documents and examine it

Examining the corpus

- The corpus contains an annotation set called “Key”, which has been manually prepared
- Within this annotation set are Document annotations with a “type” feature indicating the medical discipline of the document

What are we going to do with this corpus?

- We are going to train a machine learner to annotate documents with their discipline
- We'll start with separate training and application steps
- Later we can try some of the evaluation techniques we talked about earlier

Instances and Attributes

- This corpus so far contains only the class annotations
- There is not much in this corpus to learn from
- What would our instances be?
- What would our attributes be?
- If we run parts of ANNIE over the corpus, then we can use token features for attributes

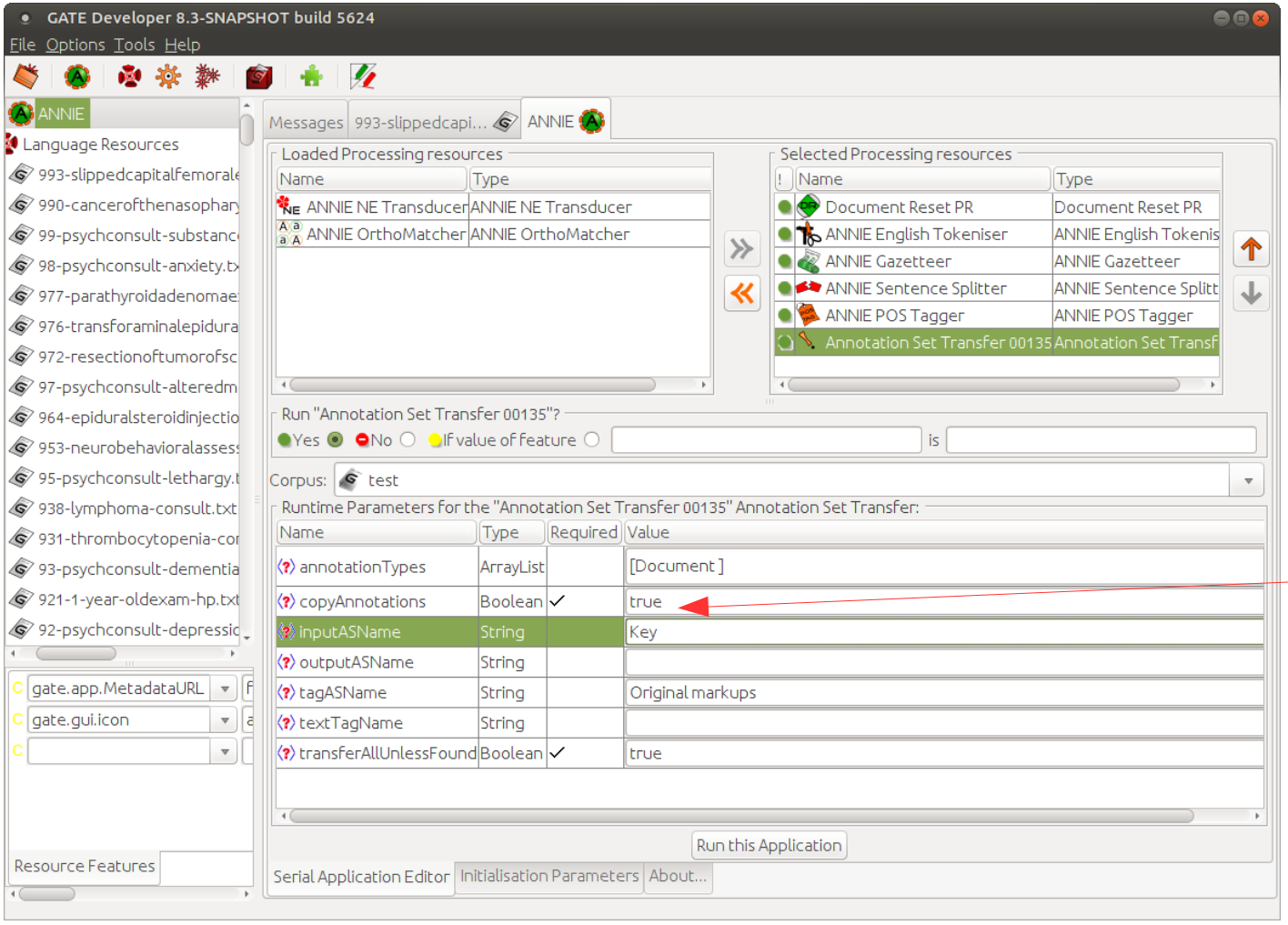
Making the Application

- Load ANNIE with defaults
- We only want tokens and some basic features so remove the last two PRs from the pipeline
  - ANNIE NE Transducer
  - ANNE Orthomatcher
- Check that the document reset PR's setsToKeep parameter includes “Key”!

Annotation Set Transfer

- The Learning Framework expects all class and feature annotations to be in the same set
- ANNIE puts annotations in the default set
- So we need to copy the Document annotations from Key into the default set
  - (We could have ANNIE output to “Key” but it would be a lot more hassle, and “Key” should be reserved for manual annotations really)
- We can use the Annotation Set Transfer PR to do this

Annotation Set Transfer



- Create an Annotation Set Transfer PR (if you can't find it, perhaps you forgot to load the Tools plugin)
- Add it to your application
- Be sure to “copyAnnotations” !!!!

Training PR

- Make a PR for classification training and add it to the application at the end
- Make one for application too—we'll come to that later. Don't add it yet though

Training PR—Parameters

- algorithmParameters—parameters influencing the algorithm, documented either in the library's own documentation or LF documentation on GitHub
- dataDirectory—where to put the model (it will be saved as a Java object on disk). It should be a directory that already exists.
- featureSpecURL—The xml file containing the description of what attributes to use
- inputASName—Input annotation set containing attributes/class
- instanceType—annotation type to use as instance

Training PR—Parameters

- scaleFeatures—use a feature scaling method for preparation? Some algorithms prefer features of similar magnitude (advanced)
- sequenceSpan—for sequence classifiers only. We'll look at this in the context of chunking
- targetFeature—which feature on the instance annotation indicates the class
- trainingAlgorithm—which algorithm to use

Feature Specification


```
<ML-CONFIG>

<NGRAM>
<NUMBER>1</NUMBER>
<TYPE>Token</TYPE>
<FEATURE>string</FEATURE>
</NGRAM>

</ML-CONFIG>
```

- This file is in your hands-on materials
- Feature specification indicates which attributes we are going to use
- This one just uses the strings of the tokens (“bag of words”, or unigrams)
- What other features might be a good idea for this task?

Feature Scaling

- Feature scaling is an advanced feature that we won't make use of today
- However it can be essential to getting a good result!
- Behind the scenes, all features are converted into numbers, for example one for the presence of a word or zero for its absence
- Other features might be the length of a word, which might range from one to twenty or more, or a frequency figure that might be a very large number
- Many algorithms work better if features have the same approximate magnitude
- Therefore after features have been gathered from the corpus, it can make sense to scale them

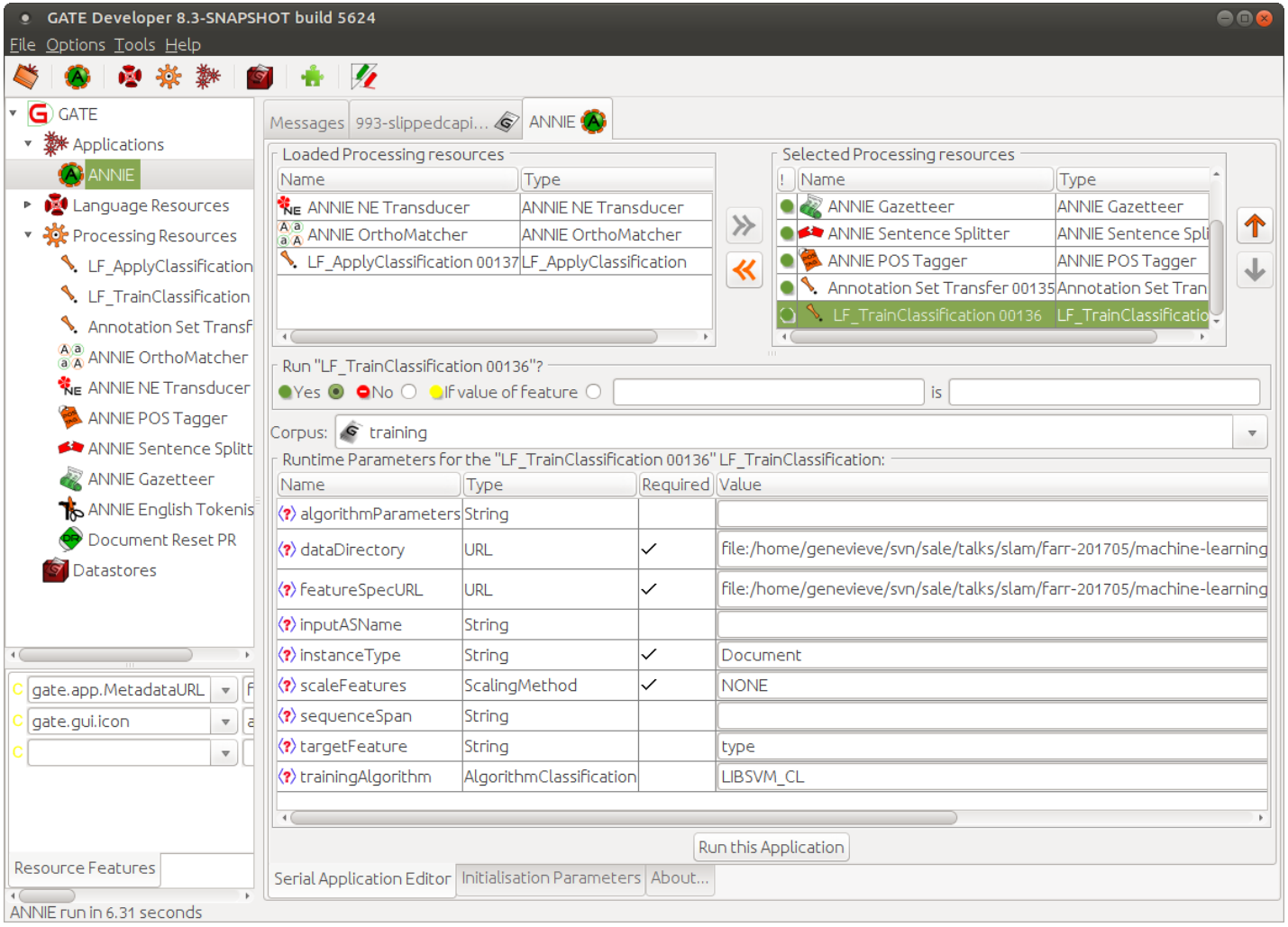
Algorithms

- Three libraries are integrated/available; Mallet and Weka, each providing many algorithms, and LibSVM (support vector machine)
- Weka requires a separate download
- Names begin with the library they are from
- After that, “CL” indicates that it's a classification algorithm and “SEQ” indicates a sequence learner
- Where to start?
  - SVM is good but you must tune it properly
  - Decision trees can be interesting to read
  - (Weka wrapper—Random Forest is good)
  - CRF is good for chunking
  - Try a few later and see for yourself!

Set parameters for training

- Be sure to set the dataDirectory to a place you can store your trained model; perhaps the hands-on folder for this classification exercise?
  - Unlike the evaluation PR, training creates a persistent model on disk that you can reuse later
  - The application PR will use the model it finds there
- You need to set the targetFeature to “type” (why?)
- For algorithm, let's try LibSVM
- Set the feature spec URL to point to the feature XML file “classification-features.xml” in your hands on materials
- instanceType should be Document (why?)

Training Classification



- Be sure to choose the right corpus for training
- Go ahead and train your model!

Training a model

- Switch to the messages pane so you can see the output
- Did it look like it worked? Can you find where it tells you what classes you have and how many features? Does it look right to you?

Classification Application

- Move the training PR out of the application, and put the application one in instead
- You can also take out the Annotation Set Transfer
  - We don't need the right answers at application time!
  - They can stay where they are, in Key, and we'll use them to compare with our new ML annotations later

Classification Application

- Many of the parameters are the same as for the training PR
- outputASName indicates where the final answers will go
  - If you set it blank, the classes will go back onto the instances
  - If you're applying to a test set, this may overwrite your class feature! So be careful! Though in our case, the class is in Key
  - The default of “LearningFramework” is fine
- Set instanceType
  - At training time, we learned from the Document annotations
  - At application time, we need to make Document annotations for the PR to classify—we'll do that next
  - So what do you think instanceType should be?

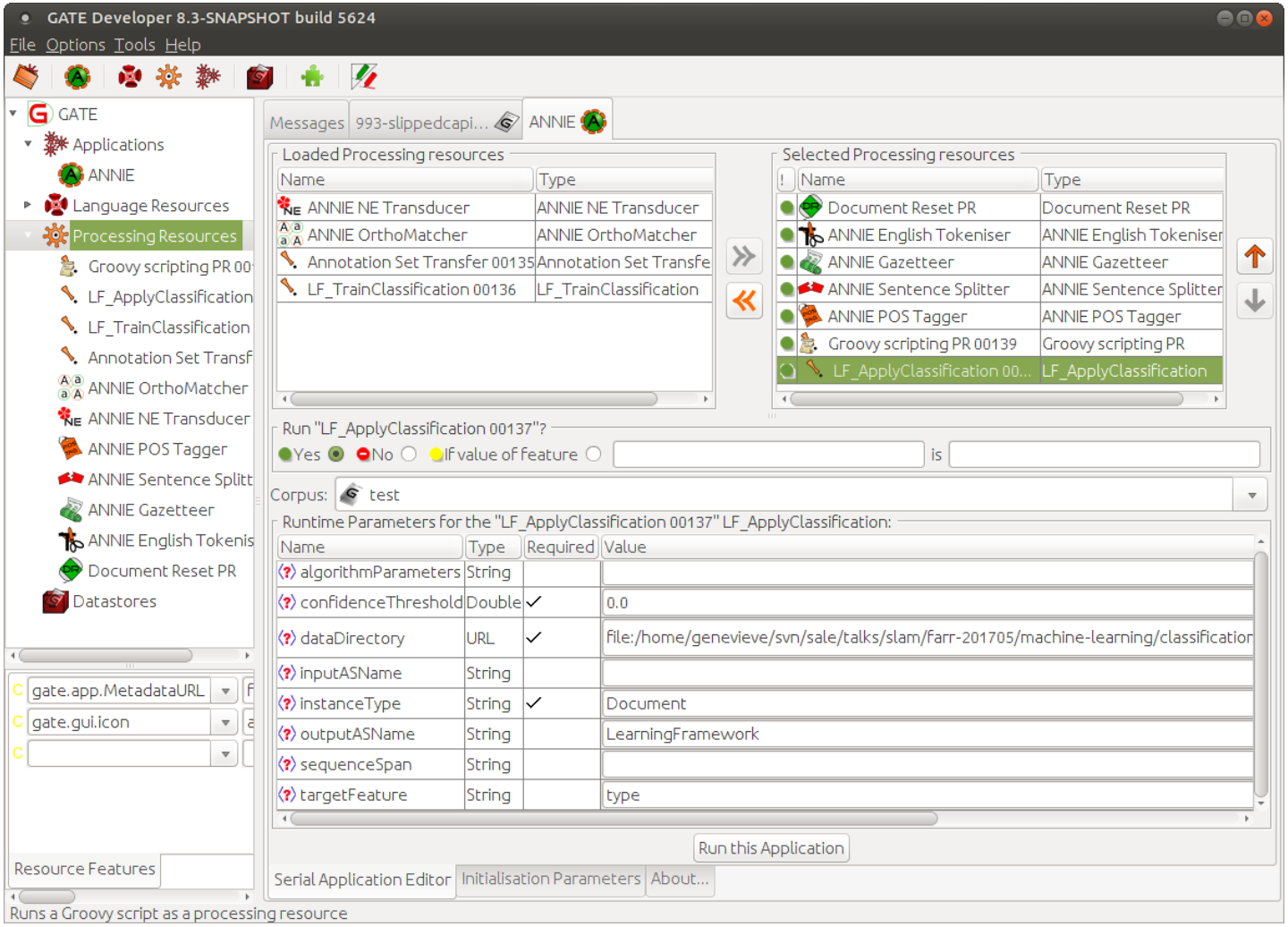
Classification Application

- You can set dataDirectory as previously, so it can find the model you just trained
- targetFeature needs to be the same as the one in the Key set, so that when we evaluate it matches
- confidenceThreshold allows you to set a threshold for how certain the model needs to be to assign a class. For a well tuned model it shouldn't be necessary. It's more relevant for problems such as finding named entities (more on that later). So we'll leave it blank

Classification Application

- Make a Groovy scripting PR
- When you make the PR, you will be asked to set the ScriptURL initialization time parameter—set it to the makeDocumentAnnotation.groovy file that is in the classification-hands-on directory
- Add the Groovy scripting PR to the application directly before the classification application PR

Applying a model



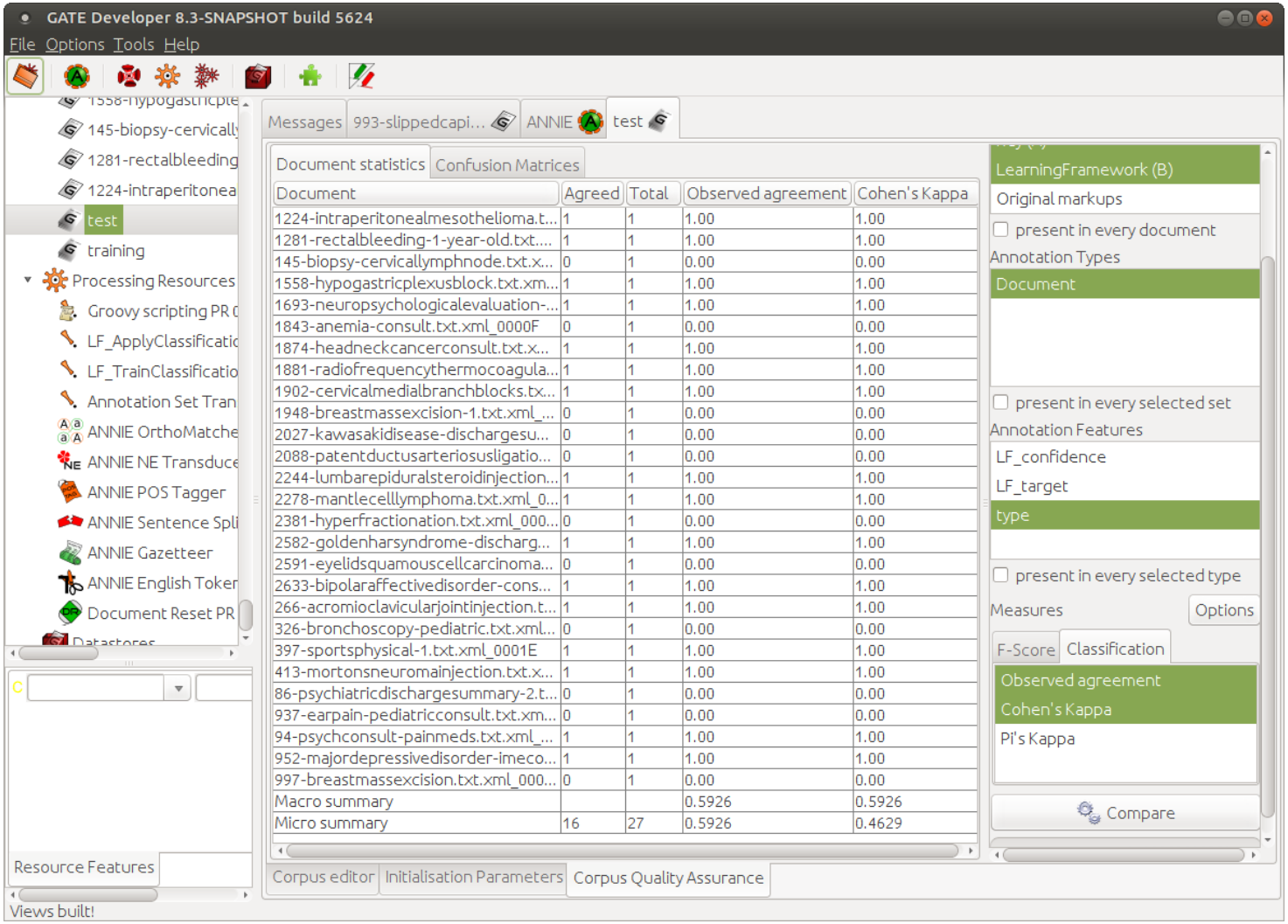
- Make sure you have selected the test corpus
- Go ahead and run the application!

Examining classification results using Corpus QA

Evaluating Classification

- Accuracy is a simple statistic that describes how many of the instances were correctly classified
- But what constitutes a good figure? 95%
- What if 99% of your instances are the majority class? You could get an accuracy of 99% whilst completely failing to separate the classes and identify any of the minority class instances at all!
- Kappa metrics provide a measure of the statistical independence of your result from the actual right answers
- Accuracy is a useful metric for parameter tuning but tells you little about how well your system is performing at its task

Corpus QA for classification



In the Corpus QA tab, select annotation sets to compare, instance type and class feature and choose both agreement and a kappa statistic

Click on “Compare”

Classification metrics

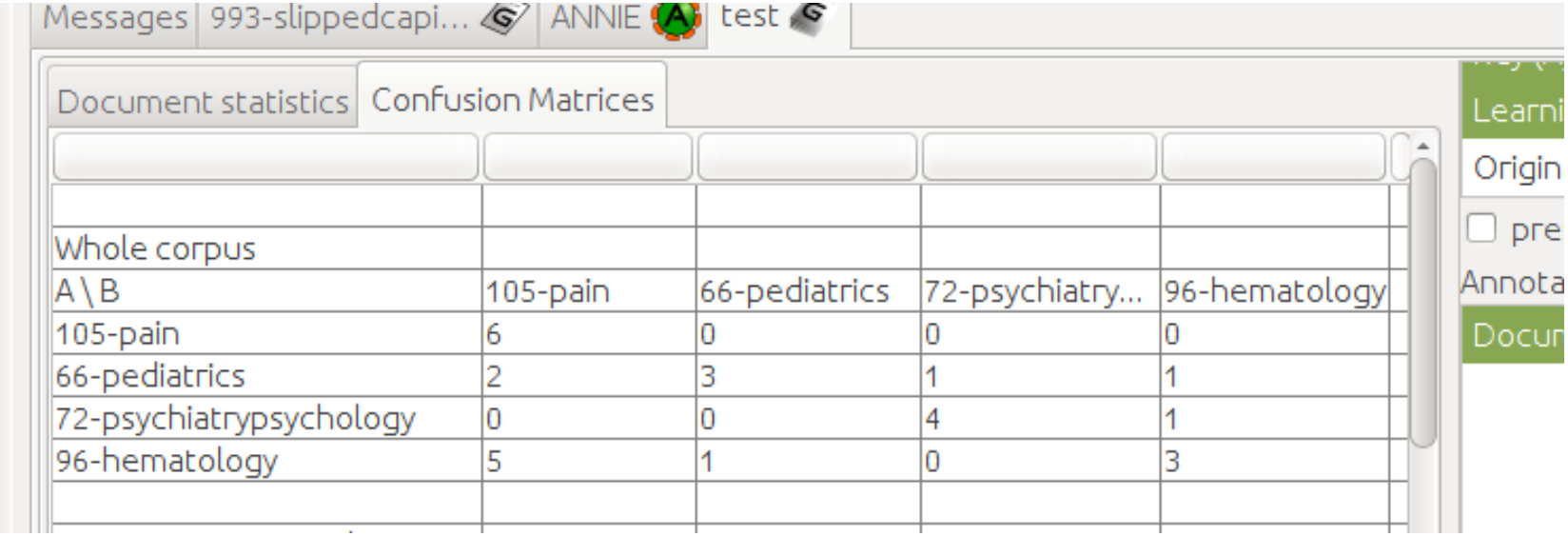
- What do you think about this result? Not bad?
- What do you think of this kappa statistic? (A kappa of over 0.5 is considered good, and over 0.8 excellent.)

Confusion matrices

- Often you can learn a lot about what might be improved by looking at the kind of mistakes your classifier is making
- A confusion matrix shows you which types tend to get confused with which other types

Confusion Matrices

- Confusion matrices are available on the next tab (at the top of the screen)
- What do you think about the misclassifications?



Whole corpus				
A \ B	105-pain	66-pediatrics	72-psychiatry...	96-hematology
105-pain	6	0	0	0
66-pediatrics	2	3	1	1
72-psychiatrypsychology	0	0	4	1
96-hematology	5	1	0	3

Classification Evaluation

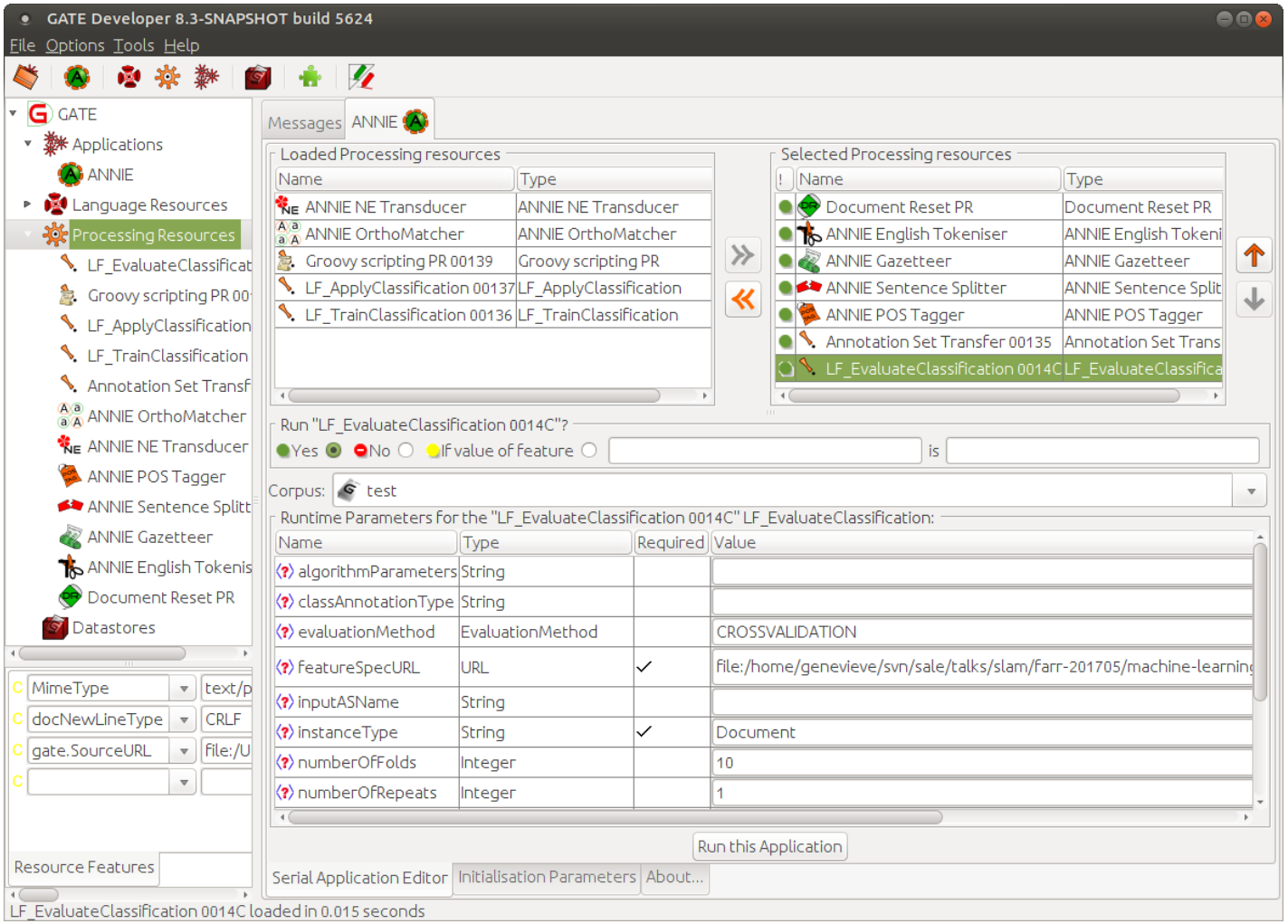
- The result is okay but hematology documents are more likely to be classified as pain documents than hematology
- Maybe we can improve this
- It would be easier to try different things using holdout or cross validation approaches, which would automate the process of splitting, training and testing

Classification using the Evaluation PR

Classification Evaluation PR

- This implements holdout and n-fold cross validation evaluation
- It will split, train and test, and give you an accuracy figure
- It does not create persistent annotations on the corpus that can be examined
- It does not provide a kappa statistic
- However it is a fast way to tune parameters
- We can later return to separate training and application, once we have improved our parameters

Making the Application



- Create and add a classification evaluation PR
- We'll need the annotation set transfer too!

Evaluation PR—Parameters

- We have already introduced some of the parameters, but this PR has several new ones
- classAnnotationType—the annotation type to use as target for chunking*. **Leave blank to indicate classification**
- evaluationMethod—Cross-validation or hold-out
- featureSpecURL—As previously, the xml file containing the feature specification
- inputASName—Input annotation set containing attributes/class (we have everything in the default annotation set)
- instanceType—annotation type to use as instance

*Why would you evaluate chunking using the classification evaluation PR? I'll tell you later!

Evaluation PR—Parameters

- numberOfFolds—number of folds for cross-validation
- numberOfRepeats—number of repeats for hold-out
- targetFeature—for classification only, which feature on the instance annotation (not classAnnotationType!) indicates the class? **Leave blank to indicate chunking**
- trainingAlgorithm—which algorithm to use
- trainingFraction—for hold-out evaluation, what fraction to train on?

More operations—Evaluation

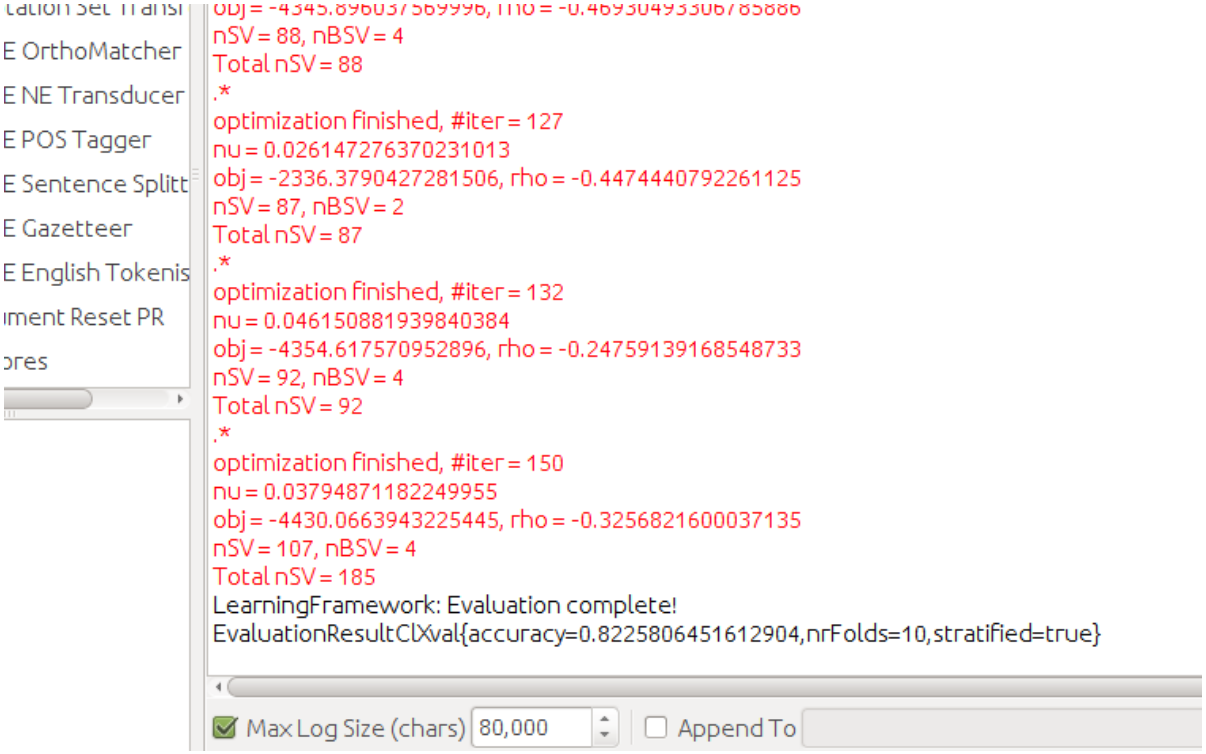
- Two evaluation modes are provided; CROSSVALIDATION and HOLDOUT
- These wrap the evaluation implementation provided by the machine learning library for that algorithm

Setting the parameters

- Now set the parameters of the evaluation PR
- classAnnotationType MUST be left blank, to indicate that we are running a classification problem
- featureSpecURL should point to the feature file
- instanceType is the annotation type we created when we copied our training sentences over from the Key set
- The more folds you use, the better your result will be, because your training portion is larger, but it will take longer to run—10 is common
- targetFeature is the feature containing the class we want to learn —what will that be?
- Let's try the LibSVM algorithm!

Running the application

- Now run the PR
- If you switch to the messages pane, before running the application by right clicking on the application in the resources pane, you can see the output as it appears



Classification Exercises

- Now see if you can improve your result
- Ideas:
  - Try different algorithms
  - For SVM, it's important to tune cost. Cost is the penalty attached to misclassification. A high cost could result in an overfitted model (it just memorised the training data and may be unable to generalize) but a low cost might mean that it didn't really try to learn! In “algorithmParameters” you can set a different cost for example like this: “-c 2”. The default cost is 1.
  - Add new features
- Where to get help: https://github.com/GateNLP/gateplugin-LearningFramework/wiki
  - E.g. the Algorithm Parameters page

Classification Exercises

- Once you have a result you're happy with, transfer what you learned back into your training application
- Then you can train a new model and run it on the test corpus
- How does your confusion matrix look now?