Prof. Dr. Chris Biemann
Seid Muhie Yimam

Language Technology Lab
Universität Hamburg

Introduction to ClearTK

CLEARTK

ClearTK = UIMA + Machine Learning

CLEARTK

- The **C**enter for Computational Language and **E**duc**A**tion **R**esearch

Computational Language and EducAtion Research
C L E A R

Colorado
University of Colorado at Boulder

- Tk = **T**ool**k**it

  - Toolkit for statistical natural language processing components
  - Written in Java
  - based on Apache UIMA framework
  - https://cleartk.github.io/cleartk/

MACHINE LEARNING - REMINDER

- There are several kind of machine learning algorithms
  - Supervised
  - Unsupervised
  - And there are others (mostly mixtures of both):
    - Semi Supervised
    - Distant Supervision
    - Reinforcement Learning
    - ….

SUPERVISED ALGORITHMS

- Problem: Sentiment Analysis (detect positive and negative attitude of text)
- Given: Training data

Instance	Class Label
I like hamsters very much.	True
I cannot stand dogs.	False
I love my cat.	True

- Extract Features

like	love	hate	I	Class Label
1	0	0	1	True
0	0	0	1	False
0	1	0	1	True

- Train a model which is able to predict the class label
- Classify data: Apply model to data, where the class label is not known

SUPERVISED ALGORITHMS: SEQUENCE TAGGER

- E.g. Hidden Markov Model, Conditional Random Fields, …
  - Consider instances from previous instance
  - Consider Class label of previous instance
- Training Data (POS Tagging):

Learn Model sentence-wise

Instance	Class Label
Both	DT
were	VBD
recorded	VBN
at	IN
Steve	NP
Rizzo's	NP
studio	NN
in	IN
Rhode	NP
Island	NP
.	SENT
He	PP
….	...

Feature Extraction

Features for **recorded**:

- word: recorded
- lemma: record
- word-1:were
- word+1: at
- lemma-1: are
- lemma+1: at
- word-1+word-2: Both_were

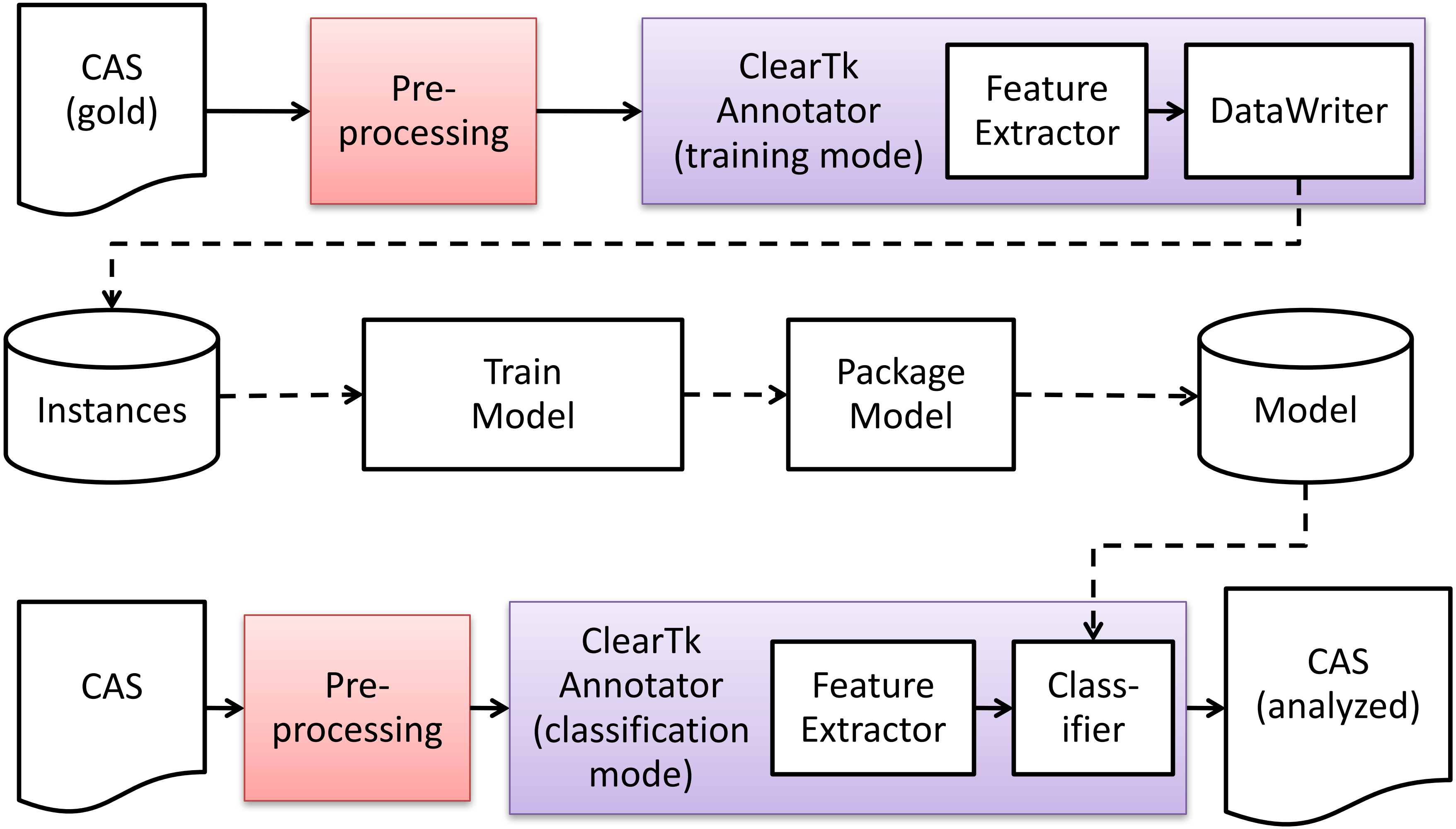
WORKFLOW

- Preprocess data (Feature Extraction)
  - Annotate features
  - Extract features from CAS
  - Generate Instances
  - Run pipeline

- Training model
  - Train classifier
  - Write model to Jar

- Classify/Usage of model
  - Use model to classify data
  - Use model to analyse data

WORKFLOW


CAS (gold)
Pre-processing
ClearTk Annotator (training mode)
Feature Extractor
DataWriter
Instances
Train Model
Package Model
Model
CAS
Pre-processing
ClearTk Annotator (classification mode)
Feature Extractor
Class-ifier
CAS (analyzed)

FEATURE EXTRACTION

- Produce features out of annotations

Stately, plump Buck Mulligan came from the stairhead, bearing a bowl of lather on which a mirror and a razor lay crossed. [1]

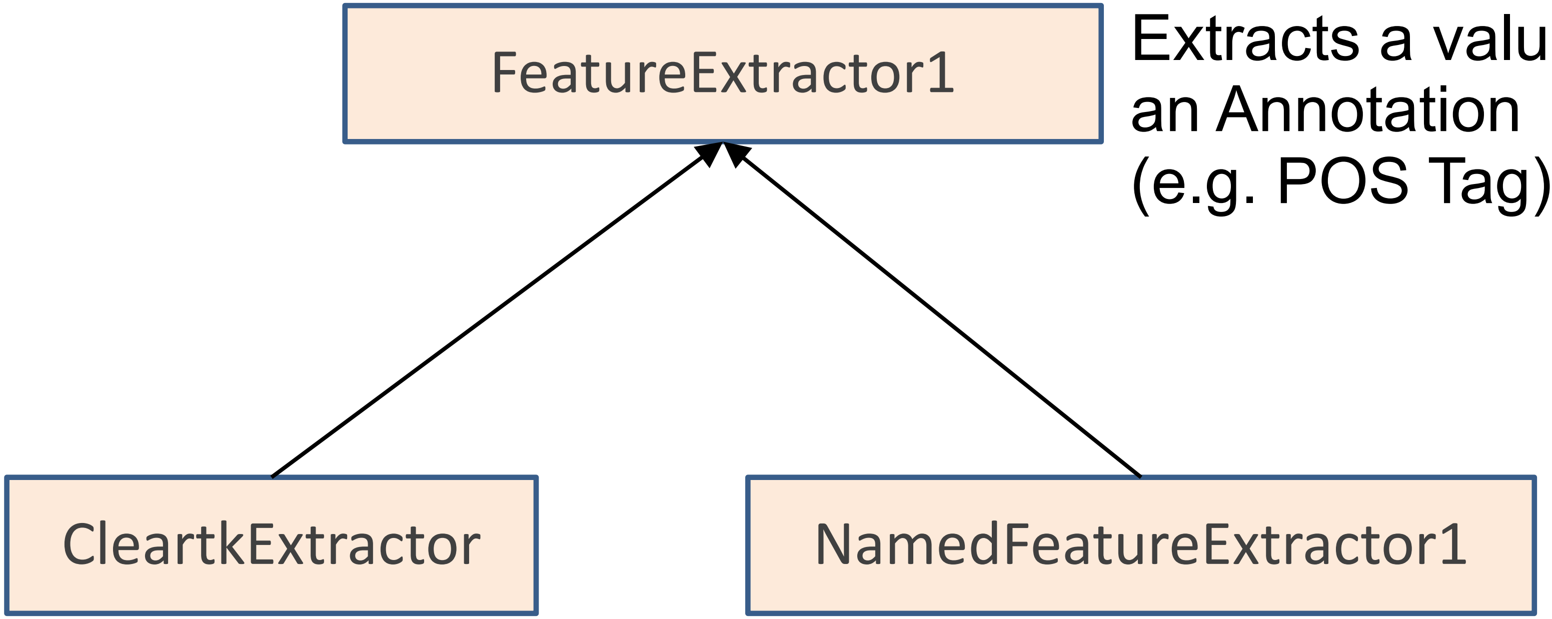
Example:

identifier: name1
tokens: token 4, token 5
character offset: 16-29
Covered text: Buck Mulligan

identifier: token13
character offset: 65-69
Part-of-speech: noun
Covered text: bowl

[1] Ulysses, James Joyce, 1920

THE MAIN FEATURE EXTRACTION INTERFACES



FeatureExtractor1

CleartkExtractor

NamedFeatureExtractor1

Extracts a value from
an Annotation
(e.g. POS Tag)

Extracts context to a
given word
(e.g. the two words
to the left of the
actual word)

Process value of a
FeatureExtractor1
(e.g. use last two
characters of feature)

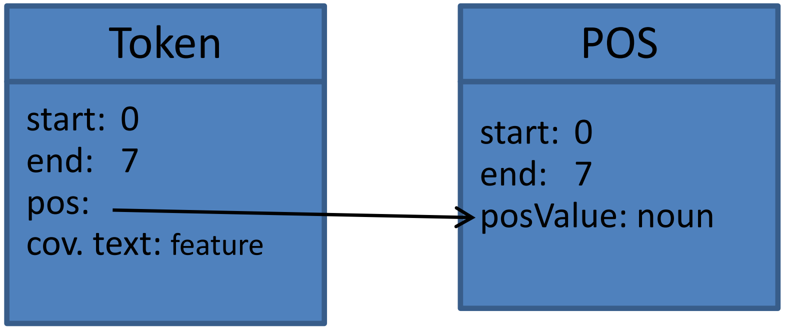
NAMED FEATURE EXTRACTORS

- NFEs extract content from an Annotation
- To create an own NFE you have to implement the following function:
  - **List<Feature> extract(JCas view, Annotation focusAnnotation)**

Example:

new TypePathExtractor(Token.class, "pos/posValue"),

NamedFeatureExtractor1	Feature value
CoveredTextExtractor	Covered text of Annotation
WhitespaceExtractor	Returns whether a whitespace is left/right to the Annotation
TypePathExtractor	Returns a value from an Annotation
…	



Token

start: 0
end: 7
pos:
cov. text: feature

POS

start: 0
end: 7
posValue: noun

FEATUREFUNCTIONEXTRACTOR

- Extract additional information from a Feature
  - Constructor:
    FeatureFunctionExtractor(FeatureExtractor1<T> extractor, **FeatureFunction... featureFunctions**)
  - Is useful to extract if a Token/Stem/... contains an upper letter, a digit, prefix, suffix, etc...
  - The **FeatureFunction** interface extends from a **Function** that with the following abstract method:
    List<Feature> apply(Feature f)

FeatureFunctions	Feature value
LowerCaseFeatureFunction	Word lowercased
CapitalTypeFeatureFunction	ALL_UPPERCASE, INITIAL_UPERCASE, ALL_LOWERCASE, MIXED_CASE
NumericTypeFeatureFunction	DIGITS, YEAR_DIGITS, ALPHANUMERIC
CharacterNGramFeatureFunction	(to-from)-character ngram e.g. CharacterNGramProliferator(RIGHT_TO_LEFT, 0,3) from „Cola“ returns „ola“ as feature
...	

FEATUREFUNCTIONEXTRACTOR→ EXAMPLE

- Extract additional information from annotation
  - Example:

```
new FeatureFunctionExtractor(new CoveredTextExtractor(),
                             new LowerCaseFeatureFunction(),
                             new CapitalTypeFeatureFunction ());
```

Text	Stately	,	plump	Buck	Mulligan
Covered Text	Stately	,	plump	Buck	Mulligan
LowerCaseFeatureFunction	stately	,	plump	buck	mulligan
CapitalTypeFeatureFunction	INITIAL_UPPERCASE	ALL_LOWERCASE	ALL_LOWERCASE	INIT_UPPERCASE	INIT_UPPERCASE

features

CLEARTKEXTRACTOR

- Extract features from **context**

```
public CleartkExtractor(Class<? Extends Annotation> annotationClass,
                FeatureExtractor1<T> extractor,
                        Context... contexts)
```

Context	description
Preceding(from [,to])	Extracts Annotation **before** actual Annotation
Following(from [,to])	Extracts Annotation **after** actual Annotation
Covered()	Extracts covered Annotation
…	

CLEARTKEXTRACTOR→ CONTEXT EXTRACTOR

- Extract features from context

```
new CleartkExtractor(Token.class,
                     new CoveredTextExtractor(),
                     new  Preceding(2),              new Following(2),
                     new Bag(new Preceding(2) ),     new Ngram(new Preceding(2)) );
```

Alice	was	beginning	to	get	very	tired	of	sitting	by	her
4	3	2	1	0	actual	0	1	2	3	4

Preceding(2):	Preceding_0_2_1_to, Preceding_0_2_0_get
Following(2):	Following_0_2_0_tired, Following_0_2_1_of
Bag(new Preceding(2))	Bag_Preceding_0_2_1_to, Bag_Preceding_0_2_0_get
Ngram(new Preceding(2))	Preceding_0_2_0_to_get

WRITE YOUR OWN FEATURE EXTRACTOR

- Write an Extractor that matches **regular expressions**

```
public class ContainsRegex implements NamedFeatureExtractor1{
        private String regex;
        public ContainsRegex(String regex) {
                this.regex = regex;
        }
        @Override
        public List<Feature> extract(JCas view, Annotation focusAnnotation)
                        throws CleartkExtractorException {
                boolean match = focusAnnotation.getCoveredText().matches(regex);
                return Collections.singletonList(new Feature("ContainsRegex",Boolean.toString(match)));
        }
}
```

The Feature class should be instanciated with a String value to avoid NullPointer Exceptions

The extract method returns a list of features !

Instances can have different number of features

ANNOTATOR FOR CLASSIFICATION AND FEATURE EXTRACTION

- Extend class from:
  - CleartkAnnotator<OUTCOME_TYPE>
    Used for non-sequential algorithms like SVM
  - **CleartkSequenceAnnotator<OUTCOME_TYPE>**
    e.g. a pos-tagger classifies tokens corresponding to one sentence
  - **OUTCOME_TYPE**
    - String
    - Boolean
    - Integer

EXAMPLE FOR POS-TAGGING WITH CRF

ANNOTATOR FOR CLASSIFICATION AND FEATURE EXTRACTION

```
public class PosTaggerAnnotator extends CleartkSequenceAnnotator<String> {
      //lists for features
       private FeatureExtractor1<Token> tokenFeatureExtractor;
    private CleartkExtractor<Token, Token> contextFeatureExtractor;
    private TypePathExtractor<Token> stemExtractor;

   public void initialize(UimaContext context)
        throws ResourceInitializationException {
          super.initialize(context);
          //initialize feature extractors (see following slides)
      }

      public void process(JCas jCas) throws AnalysisEngineProcessException {
          //extract features (see following slides)
      }
```

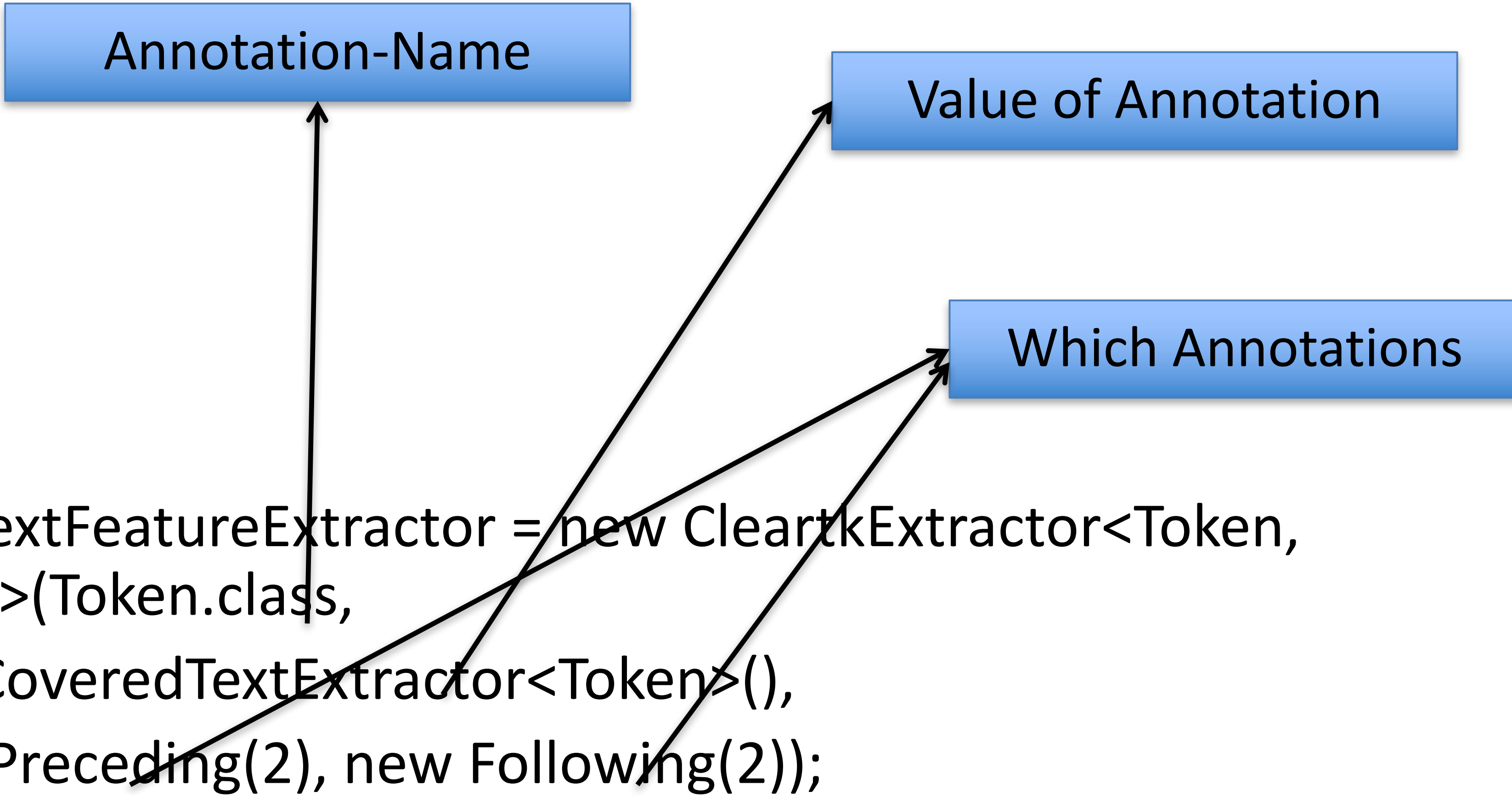
Same functions as in a „normal“ JCasAnnotator

CREATE FEATURE EXTRACTORS

```
stemExtractor = new TypePathExtractor<Token>(Token.class,
"stem/value");


this.tokenFeatureExtractor =
      new FeatureFunctionExtractor<Token>(
      new CoveredTextExtractor<Token>(),
       new LowerCaseFeatureFunction(), new
      CapitalTypeFeatureFunction(),
       new NumericTypeFeatureFunction(),
      new CharacterNgramFeatureFunction(fromRight, 0, 2));
```

CREATE FEATURE EXTRACTORS FOR CONTEXT



Annotation-Name

Value of Annotation

Which Annotations

```
this.contextFeatureExtractor = new CleartkExtractor<Token,
     Token>(Token.class,
     new CoveredTextExtractor<Token>(),
      new Preceding(2), new Following(2));
```

FEATURE EXTRACTION

THE PROCESS METHOD

```
for (Sentence sentence : select(jCas, Sentence.class)) {
        List<Instance<String>> instances = new ArrayList<Instance<String>>();
        List<Token> tokens = selectCovered(jCas, Token.class, sentence);
        for (Token token : tokens) {
           Instance<String> instance = new Instance<String>();
           instance.addAll(stemExtractor.extract(jCas, token));
           instance.addAll(tokenFeatureExtractor.extract(jCas, token));
           instance.addAll(contextFeatureExtractor.extractWithin(jCas, token, sentence));
           instance.setOutcome(token.getPos().getPosValue());
           // add the instance to the list !!!
           instances.add(instance);
        }
    // differentiate between training and classifying
    if (this.isTraining()) {
       this.dataWriter.write(instances);
    }
    else {
       List<String> posTags = this.classify(instances);
        }
}
```

TRAINING & CLASSIFICATION

```
for (Sentence sentence : select(jCas, Sentence.class)) {
    List<Instance<String>> instances = new ArrayList<Instance<String>>();
    List<Token> tokens = selectCovered(jCas, Token.class, sentence);
    for (Token token : tokens) {
      Instance<String> instance = new Instance<String>();
      instance.addAll(stemExtractor.extract(jCas, token));
      instance.addAll(tokenFeatureExtractor.extract(jCas, token));
      instance.addAll(contextFeatureExtractor.extractWithin(jCas, token, sentence));
      instance.setOutcome(token.getPos().getPosValue());
      // add the instance to the list !!!
      instances.add(instance);
    }

      if (this.isTraining()) {
        this.dataWriter.write(instances);
      }
      else {
        List<String> posTags = this.classify(instances);
       }
}
```

Difference between feature extraction and classification

RUN PIPELINE FOR PREPROCESSING & FEATURE EXTRACTION

```
runPipeline(filereader, stemmer,
    createEngine(
     PosTaggerAnnotator.class,
     CleartkSequenceAnnotator.PARAM_IS_TRAINING,true,
     DirectoryDataWriterFactory.PARAM_OUTPUT_DIRECTORY, dir,
     DefaultSequenceDataWriterFactory.PARAM_DATA_WRITER_CLASS_NAME,MalletCrfStringOutcomeDataWriter.class)));
```

- The DataWriterFactory specifies which ClassifierBuilder (training algorithm) is used

RUN PIPELINE FOR PREPROCESSING & FEATURE EXTRACTION

- **Output:**
  - **training-data.malletcrf**
    containing the training data
  - **MANIFEST.MF**
    storing the name of the classifier class
  - **encoders.ser**
    storing informations about the encoding classes for the data writer

RUN TRAINING

org.cleartk.ml.jar.Train.*main(dir);*

- Parameters can be passed to the main method
  - Depends on the algorithm
  - e.g. for MalletCRF --threads, --iterations, …
- The model is packaged into a convenient jar including the MANIFEST.MF and encoders.ser

RUN CLASSIFICATION

```
runPipeline(
reader,
stemmer,
createEngine(PosTaggerAnnotator.class,
GenericJarClassifierFactory.PARAM_CLASSIFIER_JAR_PATH, dir+"model.jar"));
```

- PosTaggerAnnotator could be extended to store the POS information

GET THE POS-TAGGER STARTED AT YOUR COMPUTER

RUN EXAMPLE

- Download Maven project from Moodle

- Import Project:
  tut5Pos

- Run main method in class:
  tut5Pos.*postagger.ExecutePosTagger*

SMALL TASK [WITHOUT CREDITS] TO GET FAMILIAR WITH CLEARTK (THE REAL TASK IS DESCRIBED SEPARATELY)

- Get familiar with the example source code
- Add additional feature extractors to improve accuracy
- Use a larger POS Tagged Dataset
  - src/main/resources/wsj_pos.train_10000
- Change parameters of the algorithm itself
- Change the Mallet CRF to the **CrfSuite**[1] implementation:
- The starter project contains the dependency as follows:
  - Package Name: **cleartk-ml-crfsuite**
  - Version: **2.0.0**
  - Data Writer Factory: **DefaultDataWriterFactory** with **CrfSuiteStringOutcomeDataWriter**

→ **What are the differences ?**

[1] http://www.chokkan.org/software/crfsuite/

INSTALLING THE CRFSUITE

–**Linux:** Should work out of the box

–**Windows**:

- Install *Microsoft Visual C++ 2010 Redistributable Package (x86)* http://www.microsoft.com/download/en/details.aspx?id=5555
- Otherwise install CrfSuite manually and add it to the PATH Environment variable

- other unixoide OS: Should work out of the box, Otherwise install the liblbfgs and the CrfSuite [1] :

```
liblbfgs-1.10:
  ./configure --enable-sse2
  make
  sudo make install

crfsuite-0.12:
  ./configure
  make
  sudo make install
```

[1]http://www.chokkan.org/software/crfsuite/manual.html#id457263

INSTALLING THE CRFSUITE

–**Linux (Only if there are any problems):**

- export LD_LIBRARY_PATH=/usr/local/lib/
- Install the CrfSuite as described on the previous slide (for macs)
- [Restart your computer]

[1]http://www.chokkan.org/software/crfsuite/manual.html#id457263

REFERENCES

- ClearTK code/documentation
  http://cleartk.github.io/cleartk/
- ClearTK POS Tagger Example
  https://cleartk.github.io/cleartk/docs/tutorial/sequence_classifier.html
- Philip V. Ogren, Philipp G. Wetzler, Steven Bethard: ClearTK: A UIMA toolkit for statistical natural language processing (2008)
  (**Attention**: some parts are deprecated !!!)