Use Weka to train features

Annie Lukas 13.05.2019

Split the data set into training, testing

- First: save xmi of each statement in the corresponding folder. Each document file, e.g “culture21.txt” is related to one folder.
- Second: retrieve all the folder names, randomly shuffle the list
- Third: split the list into training (80%) and testing (20%)

Statistics	All	Train (80%)	Test (20%)
# Files	445	355	90
Sum	16402	13142	3260
% Argu	66%	66%	65%
% Non-argu	34%	34%	35%

Example feature vector token1grams

The mouse run up the clock

The mouse run down

the	1
mouse	2
run	3
up	4
clock	5
down	6

Normalised by length of given text

[1 0.33, 2 0.17, 3 0.17, 4 0.17, 5 0.17]

[1 0.25, 2 0.25, 3 0.25, 6 0.25]

Normalised by maximum occurrence of any string in any text

[1 1, 2 0.5, 3 0.5, 4 0.5, 5 0.5]

[1 0.5, 2 0.5, 3 0.5, 6 0.5]

How does feature generation work?

- Generic Feature File Generator.java
  - use to generate arbitrary feature files for arbitrary text corpora
  - it is configurable via defined properties file
  - file generation can be triggered by executing main method of this class
  - or by creating an object of this class and calling method *generatedFeatureFiles*
- experiment-config.properties:
  - configurations to generate ARFF feature files
- feature-config.properties:
  - parameters of all feature types, which rule what features to consider for a type, how features values are computed and normalized

Configurations of features generation

- Path to collection reader (Annotation File Reader)
- Path to training and testing corpus (Split dataset)
- Path to analysis engine (Dummy)
- Feature Type: Java class used for feature computation together with its parameters (feature configs, normalisation properties)
- Whether to balance training instances by undersample / oversample
- Target class (Argumentative Discourse Unit)
- Output folder (e.g data/arff)

Primitive Feature Type

- Implement Abstract Strings Distribution
  - Implements IFeatureType
  - Serves as the abstract superclass of any feature type class that refers to the distribution of strings in a text, such as token 1-grams, token 2-grams, part-of-speech 1-grams …
  - Implements abstract functions from IFeatureType
  - Some variables :
    - candidate strings (tree map string-int): considered for feature computation together with current count of occurrences in given training set
    - total number of processed texts from the training set
    - considered strings (list): considered for feature computation

Methods of Abstract Strings Distribution

Function Name	Purpose
initialise feature determination	Initializes the creation of the feature type of each text by setting all of the given configuration parameters from the given properties object that are needed for computing the set of candidate features of this type.
update candidate features	Computes the occurrence of each string to be distinguished within the span of the text of the given JCas object with the given indices.
determine features	Determines the set of strings to be distinguished for this type based on the given aggregated feature data of the whole training set using the respective parameters from the given configuration parameters of all feature types. Also, it writes the normalization parameter for the maximum occurrence to the given normalization parameters of all feature types.
initialise feature computation	Initializes the application of this feature type by creating the features composed in this type based on the given set of all features names, and setting all configuration and normalization parameters from the given properties objects that are needed for computing feature vectors.
compute normalized feature values	Computes and returns the relative frequency of each distinguished string in the given text.
count strings in text	Determines the number of occurrences of all strings that occur in the text of the given JCas object within the span defined by given indices. What is supposed to be one string is decided by the method to be implemented in a concrete child class of this class.
count strings	Determines the number of annotations of the string type that occur in the text of the given JCas object within the span defined by given indices.

Configurations of Abstract Strings Distribution

Config parameters	Type	Meaning
Annotation type name	String	The name of the annotation type from which the strings to be counted are inferred
Once per text	Boolean	Whether to count each occurring string only once per text. Used for determining whether a string occurs frequently enough in order to consider it for feature computation
Case sensitive	Boolean	Whether or not to handle strings in a case-sensitive way
Minimum string length	Boolean	The minimum length of a string in order to be considered for feature computations
Special characters	Int	Whether to consider single special characters as strings to be considered
Minimum relative string occurence	Float / Double	The minimum relative occurrence of a string in a text collection in order to consider it for feature computations
Ignore list path	Path	The path of the lexicon that contains all strings to be ignored in the feature computation
Normalisation by length	Boolean	Whether to normalize the frequencies of all considered strings by the length of the given text in strings (true) or by the maximum occurrence of any string in any text from the processed training set (false)
Maximum absolute string occurence	Int	The maximum occurrence of any string in any text from the processed training set

Aggregate Feature Type

- Should implement IFeatureType
- Should contain instances of all sub- feature types composing this feature type
- Should override abstract methods from IFeature Type by sequentially calling the same method of the sub- feature types

Process of generating features files (1)

- Load all properties from experiment, feature properties file
- Initialise Dataset Balancer, Weka Feature File Writer, …
  - Dataset balancer is used to balance the instances of each class of a dataset
  - Weka Feature File Writer is used to write a set of feature vectors and its associated feature set to a dense ARFF
- Create Feature Type based on configuration (could be token1grams, tokenNgrams, posNgrams …)

Process of generating features files (2)

- Loop through all dataset
- Determine features
  - Initialises feature determination
    - create feature type of each text by setting all of the given configuration parameters, e.g annotation type name, count once per text or case sensitive
  - Determine the feature set to be used either by loading it from a file or by computing it using the collection reader with the given path of the training set.

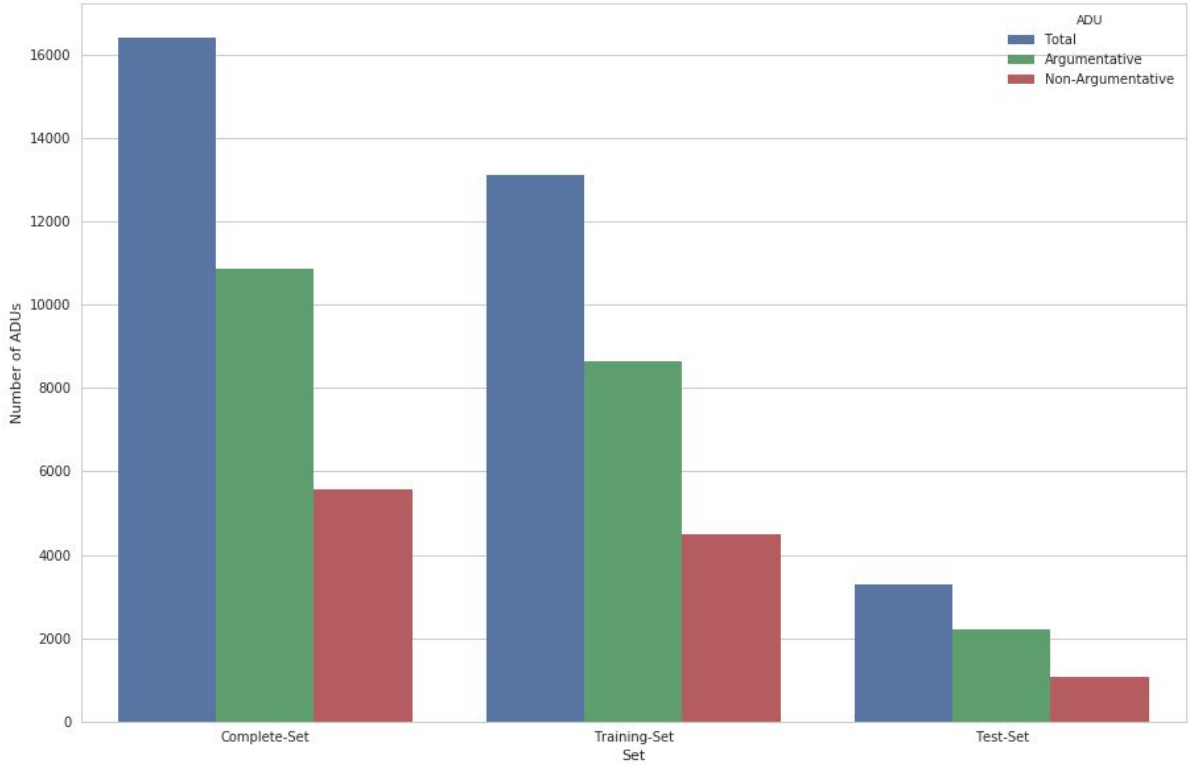
Process of generating features files (3)

- Compute feature vectors:
  - Initialise feature computation:
    - creating the all the composing sub-features based on the given set of all features names and setting all configuration and normalisation parameters from the properties (which are needed to compute feature vectors)
  - Compute and store one feature vector and its associated class values in the given lists for each text of the corpus at the given path, which is processed with the collection reader.

Process of generating features files (4)

- Balance feature vectors only for training (if necessary)
- Write feature ARFF file
  - ARFF stands for Attribute-Relation File Format, a text file that describes a list of instances sharing a set of attributes.
  - It has 2 distinct sections: Header and Data information
  - Header contains the name of the relation, a list of attributes and their types

Results


ADU
Total
Argumentative
Non-Argumentative
Number of ADUs
0
2000
4000
6000
8000
10000
12000
14000
16000
Complete-Set
Training-Set
Test-Set
Set

Results

Complete Set

	count	mean	std	min	25%	50%	75%	max
UnitsTotal	445.0	36.858427	13.350446	5.0	28.0	37.0	45.0	104.0
Argumentative	445.0	24.373034	10.567253	0.0	18.0	24.0	30.0	72.0
Non-Argumentative	445.0	12.485393	6.716786	0.0	7.0	12.0	16.0	42.0

```
UnitsTotal                16402
Argumentative             10846
Non-Argumentative          5556
```

Training Set

	count	mean	std	min	25%	50%	75%	max
UnitsTotal	356.0	36.867978	13.383908	5.0	28.00	36.5	46.0	104.0
Argumentative	356.0	24.238764	10.461291	0.0	17.75	24.0	30.0	72.0
Non-Argumentative	356.0	12.629213	6.771742	0.0	8.00	12.0	16.0	42.0

```
UnitsTotal                13125
Argumentative              8629
Non-Argumentative          4496
```

Test Set

	count	mean	std	min	25%	50%	75%	max
UnitsTotal	89.0	36.820225	13.290942	5.0	27.0	37.0	44.0	69.0
Argumentative	89.0	24.910112	11.025426	0.0	18.0	23.0	31.0	58.0
Non-Argumentative	89.0	11.910112	6.497623	0.0	7.0	11.0	16.0	33.0

```
UnitsTotal                 3277
Argumentative              2217
Non-Argumentative          1060
```

Results

POS-N-Grams:

854 Features
13125 Training Samples

Random Forest:

bagSizePercent:	100 %
batchSize:	100
maxDepth:	unlimited
numFeatures:	0
numIterations:	100

Correctly Classified:	2396	73.1157 %
Incorrectly Classified:	881	26.8843 %

F-Measure	Class
0.827	Argumentative
0.393	Non-Argumentative
0.687	**Weighted Average**

Argumentative	Non-Argumentative	←Classified as
2111	106	**Argumentative**
775	285	**Non-Argumentative**

Results

Token-N-Grams:

428 Features
13125 Training Samples

Random Forest:

bagSizePercent:	100 %
batchSize:	100
maxDepth:	unlimited
numFeatures:	0
numIterations:	100

Correctly Classified:	2384	72.7495 %
Incorrectly Classified:	893	27.1505 %

F-Measure	Class
0.820	Argumentative
0.439	Non-Argumentative
0.697	**Weighted Average**

Argumentative	Non-Argumentative	←Classified as
2034	183	**Argumentative**
710	350	**Non-Argumentative**

Results

Token + POS-N-Grams:

1281 Features
13125 Training Samples

Random Forest:

bagSizePercent:	100 %
batchSize:	100
maxDepth:	unlimited
numFeatures:	0
numIterations:	100

Correctly Classified:	2444	74.5804 %
Incorrectly Classified:	833	25.4196 %

F-Measure	Class
0.835	Argumentative
0.448	Non-Argumentative
0.710	**Weighted Average**

Argumentative	Non-Argumentative	←Classified as
2106	111	**Argumentative**
722	338	**Non-Argumentative**

Results

Token + POS-N-Grams with Undersampling:

1238 Features
8992 Training Samples

Random Forest:

bagSizePercent:	100 %
batchSize:	100
maxDepth:	unlimited
numFeatures:	0
numIterations:	100

Correctly Classified:	2343	71.4983 %
Incorrectly Classified:	934	28.5017 %

F-Measure	Class
0.785	Argumentative
0.575	Non-Argumentative
0.718	**Weighted Average**

Argumentative	Non-Argumentative	←Classified as
1710	507	**Argumentative**
427	633	**Non-Argumentative**

Results

Token + POS-N-Grams with Oversampling:

1238 Features
17258 Training Samples

Random Forest:

bagSizePercent:	100 %
batchSize:	100
maxDepth:	unlimited
numFeatures:	0
numIterations:	100

Correctly Classified:	2428	74.0922 %
Incorrectly Classified:	849	25.9078 %

F-Measure	Class
0.823	Argumentative
0.520	Non-Argumentative
0.725	**Weighted Average**

Argumentative	Non-Argumentative	←Classified as
1969	248	**Argumentative**
601	459	**Non-Argumentative**

Results

Best Intermediate Results:

Token + POS-N-Grams

Accuracy: **74.5804** % F-Measure: **0.710**

Token + POS-N-Grams with Oversampling

Accuracy: **74.0922** % F-Measure: **0.725**

Results

Token + POS-N-Grams:

1281 Features
13125 Training Samples

Random Forest:

bagSizePercent:	100 %
batchSize:	100
maxDepth:	unlimited
numFeatures:	**35**
numIterations:	**200**

Correctly Classified:	2480	75.679 %
Incorrectly Classified:	797	24.321 %

F-Measure	Class
0.840	Argumentative
0.497	Non-Argumentative
0.729	**Weighted Average**

Argumentative	Non-Argumentative	←Classified as
2086	131	**Argumentative**
666	394	**Non-Argumentative**

Results

Token + POS-N-Grams with Oversampling:

1238 Features
17258 Training Samples

Random Forest:

bagSizePercent:	100 %
batchSize:	100
maxDepth:	unlimited
numFeatures:	**35**
numIterations:	**200**

Correctly Classified:	2486	75.8621 %
Incorrectly Classified:	791	24.1379 %

F-Measure	Class
0.832	Argumentative
0.574	Non-Argumentative
0.748	**Weighted Average**

Argumentative	Non-Argumentative	←Classified as
1954	263	**Argumentative**
528	532	**Non-Argumentative**

Features	Sampling	Parameters	Accuracy	F-Measure
POS-N-Grams	No Sampling	numIterations=100, numFeatures=0	73.1157 %	0.687
POS-N-Grams	No Sampling	numIterations=200, numFeatures=35	73.5429 %	0.702
Token-N-Grams	No Sampling	numIterations=100, numFeatures=0	72.7495 %	0.697
Token-N-Grams	No-Sampling	numIterations=200, numFeatures=35	72.5359 %	0.703
POS + Token-N-Grams	No-Sampling	numIterations=100, numFeatures=0	74.5804 %	0.710
POS + Token-N-Grams	No-Sampling	numIterations=200, numFeatures=35	75.679 %	0.729
POS + Token-N-Grams	Undersampling	numIterations=100, numFeatures=0	71.4983 %	0.718
POS + Token-N-Grams	Undersampling	numIterations=200, numFeatures=35	71.8645 %	0.722
POS + Token-N-Grams	Oversampling	numIterations=100, numFeatures=0	74.0922 %	0.725
POS + Token-N-Grams	Oversampling	numIterations=200, numFeatures=35	**75.8621** %	**0.748**
POS + Token-N-Grams*	Oversampling	numIterations=100, numFeatures=0	73.4513 %	0.717
POS + Token-N-Grams*	Oversampling	numIterations=200, numFeatures=35	75.5264 %	0.744

* minoccurence for POS and Token-3-Grams set to 0.005