

Cinderella – Instrukcja obsługi

Table of Contents

1. Wstęp.....	1
2. Wymagania.....	1
3. Sposób użycia.....	2
4. Dane wejściowe.....	3
5. Fextor.....	4
6. Konfiguracja Cinderelli – sekcja preprocessing.....	13
7. Ważenie.....	14
8. Ekstrakcja cech różniących podkorpusy.....	15
9. Grupowanie.....	16
9.1. Grupowanie Cluto.....	16
9.2. Grupowanie kmeans i kmedoids.....	17
10. Uczenie modelu.....	18
11. Predykcja klas na podstawie istniejącego modelu.....	19
12. Tryb Train-and-predict.....	19
13. Walidacja krzyżowa.....	19

1. Wstęp

Cinderella jest narzędziem służącym do:

1. Klasyfikacji dokumentów tekstowych
2. Grupowania
3. Wyznaczania cech, które odróżniają od siebie podkorpusy tekstów

2. Wymagania

1. Saper – przekształcanie dokumentów tekstowych do formatu CCL
2. Wosodon – ujednaczanie znaczeń słów (opcjonalne)
3. Liner2 – znajdowanie nazw własnych (opcjonalne)
4. Fextor – wydobywanie cech z dokumentów
5. LexCSD branch new_modular – klasyfikacja i selekcja cech narzędziem WEKA
6. Scipy – inne metody selekcji cech – <http://www.scipy.org/>
7. Supermatrix – grupowanie kmeans, ważenie macierzy, funkcje podobieństwa między dokumentami
8. Cluto – grupowanie – <http://glaros.dtc.umn.edu/gkhome/cluto/cluto/overview>
9. WEKA – klasyfikacja i selekcja cech – <http://www.cs.waikato.ac.nz/ml/weka/>
10. Lib-svm – klasyfikacja SVM – <https://www.csie.ntu.edu.tw/~cjlin/libsvm/>

3. Sposób użycia

run.py [-h] -c CONFIG [-m MODEL | -x MMODEL] [-b] [-t | -v | -f | -u | -p PREDICT_MATRIX]
[-a ALGORITHM] [-o OUTPUT] [-j JOBS] file

positional arguments:

file File with multiple file paths, a single ccl file or input LexCSD matrix.

optional arguments:

-h,	--help	Show this help message and exit
-c CONFIG,	--config CONFIG	Path to config file
-m MODEL,	--model MODEL	Path to model you want to use in predict mode, or path to directory where you want to save trained model in train mode
-x MMODEL,	--multi-model MMODEL	Path to file with a list of trained models
-b,	--batch	Batch mode. Use when input is a file with multiple file paths.
-t,	--train-mode	Model training mode.
-v,	--validate	Cross-validation mode.
-f,	--feature-extraction	Extracting features differing document categories
-u,	--cluster	Clustering mode.
-p PREDICT_MATRIX,	--train-and-predict PREDICT_MATRIX	Weights train data together with test data, trains the model and predicts labels for test data.
-a ALGORITHM,	--algorithm ALGORITHM	Classification or clustering algorithm. Options: weka knn svm kmeans cluto
-o OUTPUT,	--output OUTPUT	Output file
-j JOBS	--jobs JOBS	Number of parallel processes.

4. Dane wejściowe

Jest parę możliwości wyboru formatu, w jakim powinny być dane wejściowe. Pierwsza opcja to kolekcja plików premorph, wyglądających w ten sposób:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cesAna SYSTEM "xcesAnaIPI.dtd">
<cesAna xmlns:xlink="http://www.w3.org/1999/xlink" type="pre_morph" version="STUB-1.0">
<chunkList xml:base="text.xml">
<chunk type="p" xlink:href="dv1p1" id="annots#obsz_szt:komp_szczeg">
Irenę Fedorowiczową portretował Witkacy kilkakrotnie (porównaj nr 76). Ten portret, określony
jako „podstawkowo-widmowy”, opatrzony jest też napisem „Nπ9m – Cof”, który we własnym
„witkacowskim” systemie skrótów informuje, że artysta rzekomo od dziewięciu miesięcy nie pił
alkoholu, zażywał natomiast kofeiny pod postacią kawy lub herbaty.
</chunk>
</chunkList>
</cesAna>
```

Kolejnym akceptowalnym formatem jest zbiór plików CCL. Jeżeli dokumenty będą używane do uczenia modelu klasyfikatora, trzeba przypisać klasę każdemu dokumentowi. Dokument może należeć do więcej niż jednej klasy. W pliku premorph klasa powinna być podana w atrybucie “id” elementu “chunk” np.:

```
<chunk type="p" xlink:href="dv1p1" id="annots#obsz_szt:komp_szczeg">
```

W powyższym przykładzie dokument należy do dwóch klas: obsz_szt i komp_szczeg. W pliku CCL można albo dołączyć klasę i potrójne podkreślenie do nazwy pliku (np. “obsz_szt___document_123.ccl”), albo podać ją w atrybucie “id” elementu “chunk”:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE chunkList SYSTEM "ccl.dtd">
<chunkList>
<chunk id="annots#obsz_szt">
<sentence id="s1">
<tok>
<orth>Można</orth>
<lex disamb="1"><base>można</base><ctag>pred</ctag></lex>
</tok>
...
```

Jeżeli używany jest format CCL, dokumenty będą dalej przetwarzane przez Fextora i Fextor2lexcsd – na wyjściu będą w formacie LexCSD SparseMatrix. Macierz LexCSD jest ostatnim dozwolonym formatem wejściowym do Cinderelli. Jeżeli ta sama kolekcja dokumentów ma być używana przy wielu eksperymentach, dobrze jest mieć dane zapisane w tej postaci.

Zapisana macierz LexCSD jest paczką w formacie .tar.bz2 z kilkoma plikami:

boundary.txt – nazwy cech w macierzy i indeks, na którym się kończą, np. base := 123 verb12 := 124 znaczy, że pierwsze 123 kolumny to cecha 'base' (ich nazwa zaczyna się od 'base:'), a 124 kolumna to cecha 'verb12' (liczba czasowników w pierwszej lub drugiej osobie).

class_labels.txt – klasy dokumentów odpowiadających kolejnym rzędom macierzy

column_labels.txt – nazwy kolumn

contexts.txt – teksty kolejnych dokumentów

documents_ids.txt – nazwy kolejnych dokumentów

info.txt – informacja o macierzy

matrix.txt – dane w postaci macierzy rzadkiej. Pierwsza linia składa się z trzech liczb: liczba rzędów, liczba kolumn, liczba niezerowych wartości. Kolejne linie to serie liczb opisujących kolejne rzędy macierzy – numer kolumny z niezerową wartością, wartość w tej kolumnie, następny numer kolumny z niezerową wartością itd.

possible_classes.txt – lista wszystkich klas dokumentów występujących w macierzy. Jeżeli w macierzy występuje dokument z więcej niż jedną przypisaną klasą, t.j. jego class_label to 'obsz_szt:komp_szczeg', tu będzie to traktowane jako osobna klasa.

5. Fextor

Fextor jest narzędziem do ekstrakcji cech z dokumentów w formacie CCL albo Poliqarp. Wyjściem jest plik csv, wyglądający w ten sposób:

base	tok_count	verb12	ppron12	context	doc_name	class
zapraszać:1;do:1; obejrzyć:1;galeria :1	5	1	0	Zapraszam do obejrzenia galerii.	doc_1.xml	true
Ala:1;mieć:2;kot: 2;i:1;ja:1;też:1	10	1	1	Ala ma kota. I ja też mam kota.	doc_2.xml	false

Żeby grupować albo klasyfikować zbiór dokumentów, każdy dokument musi być reprezentowany przez jeden wektor. W tym przypadku plik konfiguracyjny Fextora powinien rozpoczynać się w taki sposób:

[Extractor]

```
iterator = fextor.iterators.WholeDocumentIterator
```

```
features = base interp_signs lex_classes bigrams context doc_name class
```

Po nim następują sekcje odpowiadające poszczególnym cechom. Powinna być jedna sekcja dla każdej nazwy cechy podanej w polu "features". Te nazwy będą również nazwami cech w wyjściowej macierzy LexCSD. Ostatnimi dwoma cechami muszą być nazwa dokumentu (to trafi do pliku documents_ids.txt w macierzy) i klasa dokumentu (class_labels.txt). W pliku konfiguracyjnym Fextora sekcje tych cech wyglądają następująco:

[doc_name]

```
class = fextor.features.paragraph.DocName
```

```
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
```

```
conversion = none
```

[class]

```
class = fextor.features.paragraph.ParagraphAnnotationFeature
```

```
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
```

```
conversion = none
```

albo jeżeli klasa jest podana w nazwie pliku (class__doc_name.ccl):

[class]

```
class = fextor.features.paragraph.DocNameClassFeature
```

```
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
```

```
conversion = none
```

I jeżeli dokumenty nie mają przypisanych klas (np. będą używane tylko do grupowania), zamiast cechy „class” może zostać użyta cecha „true” – każdemu dokumentowi zostanie przypisana klasa „true”.

[true]

```
class = fextor.features.paragraph.TrueFeature
```

```
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
```

```
conversion = none
```

Trzecią cechą od końca może być cecha „context” (do pliku contexts.txt w macierzy), jeżeli

wartość pola "fextor2lexcsd_context" w sekcji "preprocessing" to True.

```
[context]
class = fextor.features.paragraph.ContextFeature
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
conversion = none
```

Wszystkie inne cechy to pomiary wystąpień czegoś w dokumencie. Typowe cechy do opisu całych dokumentów znajdują się w module `fextor.features.paragraph`:

1. ParagraphBaseDictFeature

Zwraca liczbę wystąpień poszczególnych lematów w dokumencie.

Parametry:

- `base_features` (opcjonalny) – ścieżka do pliku z listą lematów, które mają być liczone

```
[base]
class = fextor.features.paragraph.ParagraphBaseDictFeature
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
base_features = /home/ksenia/repo/cinderella/doc/resources/nkjp500
conversion = dict_to_columns
```

2. ParagraphOrthDictFeature

Zwraca liczbę poszczególnych wyrazów w dokumencie w dokładnie takiej formie, w jakiej wystąpiły.

Parametry:

- `orth_features` (opcjonalny) – ścieżka do pliku z listą orthów, które mają być liczone

```
[orth]
class = fextor.features.paragraph.ParagraphOrthDictFeature
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
orth_features = /home/ksenia/repo/cinderella/doc/resources/blogi_orth_100
conversion = dict_to_columns
```

3. ParagraphPunctuationDictFeature

Zwraca liczbę wystąpień poszczególnych znaków interpunkcyjnych w dokumencie.

Parametry:

- interp_list (opcjonalny) – ścieżka do pliku z listą znaków interpunkcyjnych, które mają być liczone

```
[interp_signs]
class = fextor.features.paragraph.ParagraphPunctuationDictFeature
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
interp_list = /home/ksenia/repo/cinderella/doc/resources/benchmark_interps
conversion = dict_to_columns
```

4. ParagraphLexClassDictFeature

Zwraca liczbę wystąpień słów z poszczególnych klas leksykalnych.

Parametry:

- lex_classes – ścieżka do pliku z listą tagów, które mają być liczone
- tagset (opcjonalny) – domyślnie nkjp

```
[lex_classes]
class = fextor.features.paragraph.ParagraphLexClassDictFeature
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
lex_classes = /home/ksenia/repo/cinderella/doc/resources/taglist.csv
conversion = dict_to_columns
```

5. ParagraphAvgMultiLexClassCountFeature

Zwraca liczbę słów, które zawierają określoną kombinację tagów.

Parametry:

- lex_class – kombinacja tagów. „tag1 tag2 tag3 AND tag4 tag5” znaczy, że dozwolone są słowa, które mają któryś z tagów tag1, tag2, tag3 i któryś z tagów tag4, tag5.

[verb12]

```
class = fextor.features.paragraph.ParagraphAvgMultiLexClassCountFeature
```

```
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
```

```
lex_class = praet fin impt bedzie pred aglt AND pri sec
```

```
conversion = none
```

6. ParagraphProperNameDictFeature

Zwraca liczbę wystąpień słów z poszczególnych kategorii nazw własnych. Wymaga, żeby dokument był otagowany Linerem.

Parametry:

- name_categories – ścieżka do pliku z kategoriami nazw własnych, które mają być liczone

[proper_names]

```
class = fextor.features.paragraph.ParagraphProperNameDictFeature
```

```
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
```

```
name_categories = /home/ksenia/tmp/propernames-top9.csv
```

```
conversion = dict_to_columns
```

7. ParagraphLexNGramDictFeature

Zwraca liczbę wystąpień n-gramów tagów, np. 'adj_subst'. Na początku i końcu zdania dodaje tagi "empty", tak, że wychodzi 'empty_subst' dla zdań zaczynających się od 'subst'.

Dodatkowe parametry:

- n_gram – długość n-gramu

- stoptags – tagi niechciane w n-gramach. Wystąpienia n-gramów z którymkolwiek z tych tagów nie będą zliczane

- starttags – tagi, z których powinien się składać n-gram. N-gramy zawierające jakikolwiek inny tag nie będą zliczane

[bigrams]

```
class = fextor.features.paragraph.ParagraphLexNGramDictFeature
```

```
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
```

```
n_gram = 2
```

```
stoptags = xxx brev
```

```
conversion = dict_to_columns
```


8. ParagraphBigLetterCountFeature

Zwraca liczbę słów w dokumencie, rozpoczynających się wielką literą.

```
[big_letter_count]  
class = fextor.features.paragraph.ParagraphBigLetterCountFeature  
licer = fextor.contexts.slicer.UniversalDocumentSlicer()  
conversion = none
```

9. ParagraphSynsetDictFeature

Zwraca liczbę poszczególnych synsetów, do których należą słowa z dokumentu.

Parametry:

- percent_per_word – ile procent synsetów wyznaczonych przez Woseda zatrzymać (domyślnie 1%). Zaokrągla wyznaczoną liczbę synsetów w górę.
- synonyms – czy uwzględniać synonimy międzyparadygmatyczne (domyślnie True)
- wordnet – dane bazy Słownosieci, potrzebne jeśli parametr synonyms=True
- h_level – jeśli podane, wyznaczone synsety zostaną zrzutowane h poziomów wyżej. Relacje „żeńskość”, „deminutywność”, „typ”, „augmentatywność” i „nacechowanie” liczą się tu jako 1 poziom w górę.

```
[synset_hyp]  
class = fextor.features.paragraph.ParagraphSynsetDictFeature  
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()  
wordnet = root kucyk localhost wordnet_work 3306  
h_level = 2  
conversion = dict_to_columns
```

10. ParagraphAllSynsetsFromLemmaFeature

Zwraca liczbę wystąpień wszystkich synsetów zawierających lemat dla każdego lematu w dokumencie. Dokument nie musi być otagowany przez Woseda.

```
[all_synsets]  
class = fextor.features.paragraph.ParagraphAllSynsetsFromLemmaFeature
```

```
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
wordnet = root kucyk localhost wordnet_work 3306
conversion = dict_to_columns
```

11. ParagraphHypernymsDictFeature

Zwraca liczbę wystąpień hiperonimów do poziomu h włącznie dla najwyższej ocenionego przez Woseda synsetu dla każdego słowa w dokumencie.

```
[hypernyms5]
class = fextor.features.paragraph.ParagraphHypernymsDictFeature
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
wordnet = root kucyk localhost wordnet_work 3306
h_level = 5
conversion = dict_to_columns
```

12. ParagraphHypernymsFromLemmaFeature

Zwraca liczbę wystąpień hiperonimów do poziomu h włącznie dla wszystkich synsetów zawierających dany lemat dla każdego lematu w dokumencie. Dokument nie musi być otagowany Wosedom.

```
[all_hypernyms]
class = fextor.features.paragraph.ParagraphHypernymsFromLemmaFeature
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
wordnet = root kucyk localhost wordnet_work 3306
h_level = 4
conversion = dict_to_columns
```

13. ParagraphDomainDictFeature

Zwraca liczbę wystąpień poszczególnych domen ze Słownosieci dla najwyższej ocenionego przez Woseda synsetu dla każdego słowa w dokumencie.

[domain]

```
class = fextor.features.paragraph.ParagraphDomainDictFeature  
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()  
wordnet = root kucyk localhost wordnet_work 3306  
conversion = dict_to_columns
```

14. ParagraphDomainFromLemmaFeature

Zwraca liczbę wystąpień domen ze Słownosieci dla wszystkich synsetów zawierających dany lemat dla każdego lematu w dokumencie. Dokument nie musi być otagowany przez Woseda.

[lemma_domains]

```
class = fextor.features.paragraph.ParagraphDomainFromLemmaFeature  
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()  
wordnet = root kucyk localhost wordnet_work 3306  
conversion = dict_to_columns
```

15. ParagraphSumoDictFeature

Zwraca liczbę wystąpień poszczególnych pojęć z ontologii SUMO w dokumencie.

[sumo]

```
class = fextor.features.paragraph.ParagraphSumoDictFeature  
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()  
conversion = dict_to_columns
```

16. ParagraphRelationFeature

Zwraca liczbę słów w dokumencie, które są połączone z jakimś lematem w Słownosieci określoną relacją.

Parametry:

- wordnet – dane bazy Słownosieci
- relation – nazwa relacji
- tagset – domyślnie nkjp

[diminutive]

class = fextor.features.paragraph.ParagraphRelationFeature

slicer = fextor.contexts.slicer.UniversalDocumentSlicer()

wordnet = root kucyk localhost wordnet_work 3306

relation = deminutywność

tagset = nkjp

conversion = none

17. Dictionary

Zwraca liczbę słów z podanej list, które wystąpiły w dokumencie.

Parametry:

- base_list – ścieżka do pliku ze słownikiem lematów

[noun_emotion_markers]

class = fextor.features.paragraph.ParagraphBaseFromListFeature

base_list = /home/kсения/repo/cinderella/doc/resources/emocje-rzeczowniki.txt

slicer = fextor.contexts.slicer.UniversalDocumentSlicer()

conversion = none

18. ParagraphWordCount

Zwraca liczbę tokenów (słów i znaków interpunkcyjnych) w dokumencie.

[tok_count]

class = fextor.features.paragraph.ParagraphWordCount

slicer = fextor.contexts.slicer.UniversalDocumentSlicer()

conversion = none

19. ParagraphWordCountPerSentence

Zwraca średnią liczbę tokenów na zdanie z dokumencie.

[avg_tok_count]

class = fextor.features.paragraph.ParagraphWordCountPerSentence

```
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
conversion = none
```

20. ParagraphInterpPerSentence

Zwraca średnią liczbę znaków interpunkcyjnych na zdanie w dokumencie.

```
[interp_per_sentence]
class = fextor.features.paragraph.ParagraphInterpPerSentence
slicer = fextor.contexts.slicer.UniversalDocumentSlicer()
conversion = none
```

6. Konfiguracja Cinderelli – sekcja preprocessing

Sekcja „preprocessing” w głównym pliku konfiguracyjnym to miejsce, w którym określa się, w jaki sposób powinny zostać przetworzone dane przed grupowaniem, klasyfikacją bądź ekstrakcją charakterystycznych cech. Przykładowo ta sekcja mogłaby wyglądać tak:

```
[preprocessing]
saper_config = /home/ksenia/repo/cinderella/cfg/saper.ini
saper_input = premorph

fextor_config = /home/ksenia/repo/cinderella/cfg/fextor_all.ini
fextor2lexcsd_context = True

min_tf = 100
min_df = 20
remove_numeric = True
features_to_remove = synset hypernyms5
stoplist = /home/ksenia/samples/lps/stoplist_lps
matrix_save_path = /home/ksenia/samples/lps/matrix_lps_tf100_df20.tar.bz2
arff_save_path = /home/ksenia/samples/lps/matrix_lps_tf100_df20.arff
```

Powyższa konfiguracja miałaby taki skutek:

1. Dokumenty zostałyby przetworzone przez Sopera. Pierwotnie były w formacie premorph. Po tej operacji są w formacie CCL.
2. Dane zostałyby przetworzone przez Fextora. Jako że fextor2lexcsd_context ma wartość True, trzecią cechą od końca w pliku konfiguracyjnym Fextora był context. Po Fextorze i Fextor2lexcsd dane byłyby w formacie macierzy LexCSD.
3. Macierz zostałaby przefiltrowana. Cechy, które wystąpiły w macierzy sumarycznie mniej niż 100 razy zostałyby usunięte. Cechy, które wystąpiły przynajmniej raz w mniej niż 20 dokumentach zostałyby usunięte. Cechy z kategorii 'base', które składają się z samych cyfr (np. base:2003) zostałyby usunięte. Cechy z kategorii synset i hypernoms5 zostałyby usunięte. Cechy z pliku „stoplist_lps” zostałyby usunięte.
4. Po tych operacjach macierz zostałaby zapisana.
5. Macierz zostałaby zapisana też w formacie arff.

7. Ważenie

Przy grupowaniu, walidacji krzyżowej, trybie Train-and-predict oraz ekstrakcji charakterystycznych cech można zważyć macierz. Żeby to zrobić, należy dopisać sekcję „weighting” do głównego pliku konfiguracyjnego:

```
[weighting]
scheme = all:tf
```

Schematy ważenia mogą być bardziej skomplikowane, np.:

```
scheme = base interp_signs:sm-mi;lex_classes bigrams:sm-mi svd-200
```

Powyższy schemat spowodowałby ważenie cech “base” i “interp_signs” funkcją MI z SuperMatrix, a cech “lex_classes” i “bigrams” najpierw funkcją MI, a potem SVD. Pozostałe cechy zostałyby bez zmian.

Dostępne metody ważenia:

- normalize
- tf
- tf_idf
- sm-function
- svd-x, gdzie x to oczekiwana liczba wymiarów

Ważenie SuperMatrix wymaga pliku konfiguracyjnego sm.ini. Są w nim podane dostępne funkcje ważące.

8. Ekstrakcja cech różniących podkorpusy

Żeby ustalić, którymi cechami najbardziej różnią się dokumenty jakiejś klasy od wszystkich innych dokumentów, można użyć Cinderelli w następujący sposób:

```
./run.py -c doc/examples/extract_features/example_config_features.ini -f -j 6 -o  
doc/examples/extract_features/output doc/examples/extract_features/blogi_tf100.tar.bz2
```

Wynik znajduje się w „doc/examples/extract_features/output/features.csv”. Jest to lista cech, które są najbardziej charakterystyczne dla każdej kategorii, począwszy od najlepszych cech. Selekcja jest wykonywana narzędziem WEKA albo Scipy.

W tym trybie trzeba zamieścić sekcję “feature_selection” w głównym pliku konfiguracyjnym:

```
[feature_selection]
```

```
attr_eval = InfoGainAttributeEval  
search_method = Ranker  
no_features_to_select = 1000
```

Możliwe opcje dla parametru attr_eval:

- KruskalWallis
- KolmogorovSmirnov
- MannWhitney
- WilcoxonRankSum
- Ttest
- InfoGainAttributeEval (*)
- CfsSubsetEval
- PrincipalComponents
- ConsistencySubsetEval
- GainRatioAttributeEval (*)
- ChiSquaredAttributeEval (*)

Metody oznaczone (*) wymagają też parametru 'no_of_features_to_select', tj. liczby najlepszych cech, które należy wyznaczyć. Metody Subset z WEKA same wyznaczają tę liczbę, a metody Scipy zwracają cechy z p-value większym niż 0.05. Parametr 'search_method' jest opcjonalny i używany tylko w WEKA: jego wartość domyślna to GreedyStepwise dla metod Subset i Ranker dla pozostałych metod.

9. Grupowanie

Znajduje grupy podobnych do siebie dokumentów w danym zbiorze. Grupować można albo programem Cluto, albo SuperMatrix (algorytmy kmeans i kmedoids)

9.1. Grupowanie Cluto

```
./run.py -c doc/examples/clustering/example_config_cluster.ini -u -a cluto -j 6 -o  
doc/examples/clustering/output doc/examples/clustering/blogi_tf100.tar.bz2
```

Cluto ma dwa podprogramy – scluster i vcluster. Wejściem do vcluster są wektory cech w formie macierzy rzadkiej (szczegółowy opis w dokumentacji Cluto). Podobieństwo między dokumentami jest wtedy liczone wbudowaną w Cluto funkcją cosinus. Scluster potrzebuje kwadratowej macierzy podobieństwa – można taką uzyskać programem parallelSimilarity z SuperMatrix (uwaga: parametr '-minimum-similarity' ma ustawioną domyślną wartość '0.001', a parametr '-k-best-results' - '20'). Można użyć dowolnej miary podobieństwa, zaimplementowanej w Supermatrix. (m.in. cosine, dice, ratio, pentp, shd, lin, jaccard, bhattacharyya).

W głównym pliku konfiguracyjnym trzeba dopisać sekcję 'clustering':

```
[clustering]
```

```
# ścieżka do vcluster albo scluster
```

```
cluto = /home/ksenia/tmp/cluto-2.1.1/Linux/scluster
```

```
#konwerter formatu z macierzy rzadkiej SuperMatrix do macierzy rzadkiej Cluto
```

```
clutoconv = /home/ksenia/repo/supermatrix/bin/tools/ClutoConv/ClutoConv
```

```
# Ścieżka do parallelSimilarity i funkcja podobieństwa (tylko dla scluster)
```

```
similarity = /home/ksenia/repo/supermatrix/bin/parallel/similarity/parallelSimilarity
```

```
simfunction = ratio
```

```
# Liczba grup (default: 10)
```

```
no_clusters = 20
```


Wynik:

- stats.txt – plik ze statystykami z Cluto
- contents.csv – nazwy dokumentów w każdej grupie
- features.csv – charakterystyczne cechy dla poszczególnych grup (powstaje, jeżeli plik konfiguracyjny zawierał sekcję „feature_selection” - opis w rozdziale 8)
- tree.ps – drzewo podobieństwa z Cluto

9.2. Grupowanie kmeans i kmedoids

Grupowanie za pomocą SuperMatrix algorytmami kmeans i kmedoids. Wywołanie:

```
./run.py -c doc/examples/clustering_kmeans/example_config_kmeans.ini -u -a kmeans -j 6 -o  
doc/examples/clustering_kmeans/output doc/examples/clustering_kmeans/blogi_tf100.tar.bz2
```

Sekcja „clustering” w tym wypadku wygląda następująco:

```
[clustering]
```

```
#Bin-paths - jedna z dwóch poniższych ścieżek
```

```
kmeans = /home/ksenia/repo/supermatrix/bin/clustering/kmeans/kmeans2cluto
```

```
#kmedoids = /home/ksenia/repo/supermatrix/bin/clustering/kmedoids/kmedoids2cluto
```

```
# Funkcja podobieństwa dla SuperMatrix (default: lin_cos())
```

```
sim_function_cluster = lin_cos()
```

```
# Liczba grup (default: 10)
```

```
no_clusters = 20
```

Wynik:

- clustering_output – plik wyjściowy z SuperMatrix
- contents.csv – nazwy dokumentów w każdej grupie
- features.csv – charakterystyczne cechy dla poszczególnych grup (powstaje, jeżeli plik konfiguracyjny zawierał sekcję „feature_selection” - opis w rozdziale 8)

10. *Uczenie modelu*

Uczy jeden model na wszystkich danych wejściowych. W tym trybie nie ma opcji ważenia – żeby zważyć macierz, należy użyć trybu Train-and-predict.

```
./run.py -c doc/examples/train/example_config_train.ini -t -b -a weka -m doc/examples/train/model  
doc/examples/train/input_data.txt
```

Do pliku konfiguracyjnego należy dodać sekcję „train”:

```
[train]
```

```
classifier_config = cfg/classification.ini
```

```
#kernel = linear
```

```
class_splitter = MatrixSplitter
```

```
class_delimiter = :
```

```
do_feature_selection = True
```

```
scaling = standarize
```

Parametry:

- classifier_config – plik konfiguracyjny, potrzebny jeżeli używana jest WEKA
- kernel – potrzebny, jeżeli używany jest LibSVM
- class_splitter i class_delimiter – potrzebne, jeżeli macierz zawiera wiele klas i chcemy mieć model dla każdej klasy z osobna. Po podziale jest tyle macierzy, ile było klas w macierzy wejściowej. Każda z nich ma nazwę klasy w head_word i możliwe klasy „true” i „false”.
- do_feature_selection – czy robić selekcję cech (jeżeli True, potrzebna jest sekcja „feature_selection”)
- scaling – sposób skalowania danych. Opcje: standarize – każda kolumna będzie miała średnią wartość 0 i wariancję 1, rescale – każda kolumna będzie miała minimalną wartość 0 i maksymalną 1. Bez tego parametru nie ma skalowania.

11. Predykcja klas na podstawie istniejącego modelu

Przypisuje klasy dokumentom na podstawie nauczonego wcześniej modelu.

```
.run.py -c doc/examples/predict/example_config_predict.ini -a weka -b -x  
doc/examples/train/model/models_paths.txt -o doc/examples/predict/output.csv  
doc/examples/predict/input_data.txt
```

Jeżeli przy uczeniu modelu dane były skalowane, należy dodać do pliku konfiguracyjnego sekcję „predict” z taką samą wartością parametru „scaling”, co przy uczeniu:

```
[predict]  
scaling = standarize
```

12. Tryb Train-and-predict

Łączy dane treningowe i dane testowe w jedną macierz, waży ją, potem uczy model na danych treningowych i przypisuje klasy danym testowym.

```
./run.py -c doc/examples/train_and_predict/example_config_t-p.ini -a svm -m  
doc/examples/train_and_predict/model -p doc/examples/train_and_predict/matrix_test.tar.bz2 -o  
doc/examples/train_and_predict/output doc/examples/train_and_predict/matrix_train.tar.bz2
```

13. Walidacja krzyżowa

Walidacja krzyżowa n-fold.

```
./run.py -c doc/examples/cross-validation/example_config_c-v.ini -v -b -a svm -m  
doc/examples/cross-validation/model -j 6 -o doc/examples/cross-validation/output  
doc/examples/cross-validation/input_data.txt
```

Najpierw cała macierz jest ważona, jeżeli w pliku konfiguracyjnym występuje sekcja „weighting”. Następnie dane są dzielone na trzy zbiory: treningowy, tuningowy i ewentualnie testowy. Na zbiorze tuningowym wykonywana jest selekcja cech dla każdej klasy z osobna. Zbiór treningowy jest używany do walidacji krzyżowej – dzielony jest na n części, z czego za każdym razem jedna jest odłożona do testowania, a na pozostałych uczony jest model. Jeżeli ma być też użyty osobny zbiór testowy, to po walidacji krzyżowej uczony jest model na wszystkich danych treningowych, a potem jest on testowany na zbiorze testowym.

W pliku konfiguracyjnym należy umieścić sekcję „validation”:

[validation]

```
#sets_file = doc/examples/cross-validation/folds_5
```

```
folds = 5
```

```
save_sets = doc/examples/cross-validation/folds_5
```

```
train_set_fraction = 0.7
```

```
test_set_fraction = 0.1
```

```
use_test_set = True
```

```
categories = Szachy Narciarstwo
```

Parametry:

- sets_file – ścieżka do pliku z podziałem na zbiory treningowy, tuningowy i testowy.
- folds – liczba foldów
- save_sets – gdzie zapisać podział na zbiory, jeżeli taki jeszcze nie istnieje
- train_set_fraction – ile danych powinno trafić do zbioru treningowego
- test_set_fraction – ile do testowego. Pozostałe trafiają do tuningowego.
- use_test_set – czy użyć zbioru testowego
- categories – lista kategorii, których należy użyć w walidacji. Jeżeli nie ma tego parametru, użyte będą wszystkie kategorie

Należy także dodać sekcję „tune”:

[tune]

```
class_splitter = MatrixSplitter
```

```
class_delimiter = :
```

```
subsampling_ratio = 2
```

```
do_feature_selection = True
```

- class_splitter i class delimiter - potrzebne w przypadku, gdy macierz zawiera wiele klas.
- subsampling_ratio - stosunek liczby danych negatywnych do liczby danych pozytywnych w danych uczących. Jeżeli nie ma tego parametru, dane uczące nie zostaną podpróbkowane.
- do_feature_selection – czy przeprowadzić selekcję cech na zbiorze tuningowym

Należy też dodać sekcje „train” i „predict”. W sekcji „train” nie należy umieszczać parametru `matrix_splitter`, a parametr `do_feature_selection` należy ustawić na `False`.

Wynik:

- `fold_stats.csv` – podsumowanie wyników dla każdej klasy i dla każdego foldu
- `classname_results.csv` (jeden dla każdej klasy)
- `classname_features.csv` (jeden dla każdej klasy)