README.md

WEKA-AWS-Hadoop-EMR-MapReduce-Syntactic-Similarity-Classification

WEKA/AWS/Hadoop Elastic Map Reduce application to measure and classify word pair similarity.

Project GitHub repository link:
https://github.com/itaybou/WEKA-AWS-Hadoop-EMR-MapReduce-Syntactic-Similarity-Classification

- Google English Syntactic Biarcs path: **https://storage.googleapis.com/books/syntactic-ngrams/index.html**
- Assignment Syntactic Biarcs S3 path: **s3://assignment3dsp/biarcs/**

Created by:

Itay Bouganim : 305278384

Sahar Vaya : 205583453

Table of contents

General Info And Stages

This assignment follows the paper from Ljubešić et al.: Comparing Measures of Semantic Similarity with some modifications.
In this assignment we will perform **3** stages in order to determine whether two words are similar:

1. We will generate 4 syntactic relatedness co-occurrence vectors for every lexeme from the given Google English Syntactic Biarcs corpus.
   The features for each lexeme will be pairs of the feature word and the sentence dependency (e.g subject, noun etc.).
   Each vector will contain different ***meausre of assocation*** between the lexeme and the syntactic features related to that lexeme in the syntactic sentence tree so that each vector will be a syntactic representation of the lexeme by using the features it is connected to in the corpus.
2. We will then use a pre-defined golden standard word-pairs set and ***measure the similarity*** of their representing 4 co-occurrence vectors calculated in the previous stage.
   The similarity will be measured by using 6 different methods used to calculate the distance between those vectors and the output will be a vector of size 24 (4 association methods x 6 similarity methods) representing the similarity between the word pair.
3. Then we will use a classical ML Random Forest algorithm (Supervised learning) by using WEKA Java package in order to ***classify*** the similarity vectors produced and compare the results to the pre-classified word-pairs in the golden standard.

We will be using Amazon Elastic Map-Reduce (EMR) in order to compute the first two stages out of the input corpus.
The produced output of the Map-Reduce stages will be the word-pair similaritry vectors.
For the third stage we will use the Java WEKA package in order to train and evaluate our classifier based on the labeld word-pairs given in the golden standard.

Expirements:

We will perform 2 expirements on the classification reults classified on the produced similarty vectors:
Using 1 corpus file and using 15 corpus files from the Google English Syntactic Biarcs corpus. We will show the statistics and classification results for both of the expirements in this summary file.

Memory Assumptions:

- We assume that the Golden Standard word pair classification file and the word-pair data it contains can be stored in the remote Map-Reduce EC2 instances memory.
- We assume that the WEKA ARFF input file that includes the classification input similarity vectors data can be stored in memory.

Stage 1 - Measures Of Association With Context

Measures of association with context are used to compute values that are included in the co-occurrence vectors.
These values are based on the frequencies of lexemes and features extracted from corpus. In this assignment we will use 4 different methods to calculate the measures of association producing 4 different co-occurrence vectors for each lexeme in the corpus with values relevant to each of the association methods.

The 4 association methods used are:

1. **Plain Frequency** - `count(L=l, F=f)` the amount of times that lexeme `l` appeared with feature `f` in the corpus.
2. **Relative Frequency** - `P(F=f | L=l)` the amount of times that lexeme `l` appeared with feature `f` divided by the total amount of appearences of lexeme `l` (So we will get a normalized vector relative to the appearences of the lexeme).
   Meaning that `P(F=f | L=l) = count(L=l, F=f) / count(L=l)` and `count(L=l)` is the amount of time that lexeme `l` appeared in the corpus.
3. **Pointwise Mutual Information (PMI)** - `log_2(P(L=l, F=f) / (P(L=l) * P(F=f)))` where `P(L=l, F=f) = count(L=l, F=f) / count(L)`, `count(L)` is the total number of appearences of different lexemes in the corpus, `P(L=l)` is the frequency of lexeme `l` in the corpus (`count(L=l) / count(L)`) and `P(F=f)` is the frequency of feature `f` in the corpus (`count(F=f) / count(F)`).
   This measure computes how often a lexeme `l` and a feature `f` co-occur, compared with what would be expected if they were independent.
4. **T-Test Measure** - `(P(L=l, F=f) - (P(L=l) * P(F=f))) / sqrt((P(L=l) * P(F=f)))` where `P(L=l, F=f) = count(L=l, F=f) / count(L)`, `P(L=l)` is the frequency of lexeme `l` in the corpus (`count(L=l) / count(L)`) and `P(F=f)` is the frequency of feature `f` in the corpus (`count(F=f) / count(F)`).
   Is a statistic measurment which attempts to capture the same intuition as pointwise mutual information statistic which computes the difference between observed and expected means, normalized by the variance.

Stage 2 - Measures Of Vector Similarity

Measures of vector similarity are used to compare two vectors, ie. lexemes built from association measures for features used to describe the lexemes. In this assignment we will use 6 different methods to calculate the similarity between co-occurrence vectors producing, for each lexeme in the golden standard, a single vector with length 24 (4 measure association x 6 similarity measures).

The 6 similarity measured used are:
`l1` and `l2` are the co-occurrence vectors corrosponding with word 1 and word 2 in the compared word pair.

1. **Manhattan Distance** - Calculates the distance between vectors on all dimensions: `sum(abs(l1[i] - l2[i]))`.
2. **Euclidean Distance** - Measures the geometric distance between the two vectors: `sqrt(sum((l1[i] - l2[i])^2))`.
3. **Cosine Distance** - A measure used in information retrieval is the dot product operator from linear algebra.
   If the vectors are normalized, that measure is equal to the cosine between the two vectors.
   The cosine similarity measure is computed by `(sum(l1[i] * l2[i]) / (sqrt(sum(l1[i]^2)) * sqrt(sum(l2[i]^2))))`.
4. **Jaccard Measure** - A measure that is derived from information retrieval.
   Divides the number of equal features with the number of features in general and extended to vectors with weighted associations as follows: `(sum(min(l1[i], l2[i])) / sum(max(l1[i], l2[i])))`.
5. **Dice Measure** - Very similar to the Jaccard measure and is also introduced from information retrieval.
   The equivalent of the Dice measure for weighted associations is computed as follows: `((2 * sum(min(l1[i], l2[i]))) / sum(l1[i] + l2[i]))`.
6. **Jensen-Shannon Divergence** - From the family of information-theoretic distributional similarity measures.
   The intuition of these methods is that two vectors `l1` and `l2` are similar to the extent that their probability distributions `P(F=f|l1)` and `P(F=f|l2)` are similar.
   We compare the two probability distributions by using the **Kullback-Leibler divergence** that is computed by: `D(l||j) = sum(l[i] *`

`log_10(l[i] / j[i]))`.

To bypass the negative property (Since our co-occurrence vectors are inheritably sparse, meaning they contain a lot of zeros) we use the **Jensen-Shannon** divergence which is computed as follows: `D(l1||(l1 + l2) / 2) + D(l2||(l1 + l2) / 2)`.

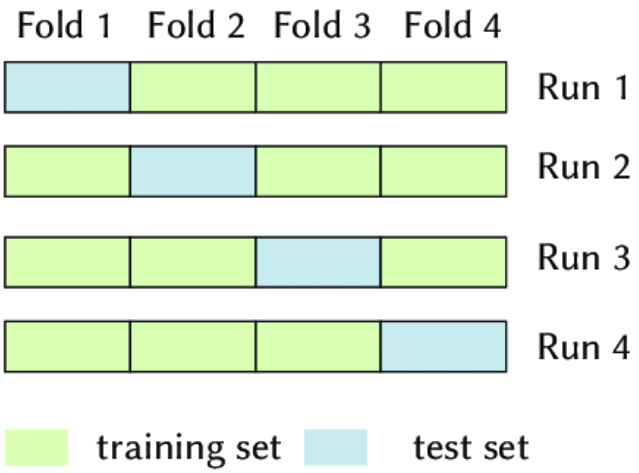
by using both formulas we get that the Jensen-Shannon Divergence of vectors `l1` and `l2` is equal to: `sum(l1[i] * log(l1[i] / ((l1[i] + l2[i]) / 2))) + sum(l2[i] * log(l2[i] / ((l1[i] + l2[i]) / 2)))`

The output vector for each lexeme will have the following shape:

```
1 plain frequency vector - Manhattan distance
2 plain frequency vector - Euclidean distance
3 plain frequency vector - Cosine distance
4 plain frequency vector - Jaccard measure
5 plain frequency vector - Dice measure
6 plain frequency vector - Jensen-Shannon divergence
7 relative frequency vector - Manhattan distance
8 relative frequency vector - Euclidean distance
9 relative frequency vector - Cosine distance
10 relative frequency vector - Jaccard measure
11 relative frequency vector - Dice measure
12 relative frequency vector - Jensen-Shannon divergence
13 pointwise mutual information vector - Manhattan distance
14 pointwise mutual information vector - Euclidean distance
15 pointwise mutual information vector - Cosine distance
16 pointwise mutual information vector - Jaccard measure
17 pointwise mutual information vector - Dice measure
18 pointwise mutual information vector - Jensen-Shannon divergence
19 t-test statistic vector - Manhattan distance
20 t-test statistic vector - Euclidean distance
21 t-test statistic vector - Cosine distance
22 t-test statistic vector - Jaccard measure
23 t-test statistic vector - Dice measure
24 t-test statistic vector- Jensen-Shannon divergence
```

Stage 3 - Similarity Vectors Classification (Similar or Not-Similar)

In this stage we use Random Forest classifier in order te perform a supervised learning task on our pre-labeled word pairs from the golden standard.
The input for our RF classifier is the similarity vectors for the word pairs from the golden standard produced in the previous stage with the labels given in the golden standard.
The output is the **True-Positive, False-Positive, True-Negative and True-Positive rates**. We use **10-fold cross validation** method in order to evaluate our model performance (which trains 10 iterations of the model by using a different portion of the training data as testing data in each iteration).



We also output the **Precision, Recall and F1 measurments** where:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{True\ Positive}{Total\ Predicted\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{True\ Positive}{Total\ Actual\ Positive}$$

$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

F-score or F-measure is a measure of a test's accuracy.
It is calculated from the precision and recall of the test.
Precision - also known as positive predictive value, is the number of correctly identified positive results divided by the number of all positive results, including those not identified correctly.
Recall - also known as sensitivity in diagnostic binary classification, is the number of correctly identified positive results divided by the number of all samples that should have been identified as positive.
The F1 score is the harmonic mean of the precision and recall.

Additonal information

EC2 instances used: Workers -

- Machine types - (64-bit x86) type: M4_LARGE

Setup

1. Install aws cli in your operating system, for more information click here : https://aws.amazon.com/cli/
2. Configure your amazon aws credentials in your .aws directory, alternatively you can set your credentials by using aws cli : write in your cmd - "aws config".

Instructions

1. Inside the project directory compile the project using the command : `mvn package` .
2. Create in the project target directory file named `inputs.txt` .

3. Create input bucket and output bucket (you can use the same bucket and create only one bucket) in AWS S3.
4. Fill the `inputs.txt` file according to the following format:

```
<input-bucket> <input-jar-file-name> <input-golden-standard> <upload-jar-and-golden-standard (true/false)>
<corpus-input-path>
<output-bucket>
<corpus-files-count (0 < x < 100)>
<worker-instance-count (0 < x < 10)>
<calculate-measures (true/false)> <optional-measures-path>
<output-co-occurrence-vectors (true/false)>
<run-classifier (true/false)> <classifier-output-path> <optional-classifier-input-path>
<delete-after-finished (true/false)>
```

- `<input-bucket>` - the bucket that contains the Map-Reduce JAR file and the Golden-Standard file, `<input-jar-file-name>` the name of the Map-Reduce JAR file, `<input-golden-standard>` the name of the golden standard file, `<upload-jar-and-golden-standard>` true/false - wether to upload the JAR file and the golden standard file to the supplied input bucket.
- `<corpus-input-path>` - The input S3 bucket where the corpus files are located.
- `<output-bucket>` - Is the bucket the job will store outputs in, Can be the same as input bucket.
- `<corpus-files-count (0 < x < 100)>` - The amount of files from the Google English Syntactic Biarcs corpus that will be used as input to the Map-Reduce jobs.
- `<worker-instance-count (0 < x < 10)>` - The EC2 instance count that will be used for the map-reduce job (value between 0 excluding and 9 including)
- `<calculate-measures (true/false)>` - Wether the measures of association calculation is needed, if false you will need to supply the `<optional-measures-path>` S3 bucket path in which the measures of association output from a previous run are located.
- `<output-co-occurrence-vectors (true/false)>` - Wether to output the co-occurrence vectors produced after the measures of association job. (**Used for testing purposed only and may create an overhead**).
- `<run-classifier (true/false)>` - Whether to run the WEKA classifier in the end of the Map-Reduce Job, `<classifier-output-path>` is the output directory for the classification summary file, `<optional-classifier-input-path>` a local input path for the similarity vectors **If provided the Map-Reduce Job will not run and only the classifier will run, ignore this parameter to run the whole process**.
- `<delete-after-finished (true/false)>` - Wether to delete the output bucket after the process ends.

5. Make sure your input text file located in the project target directory or in the same directory as the WordPairSimilarityRunner JAR file.
6. The application should be run as follows:
   `java -jar WordPairSimilarityRunner.jar`

IMPORTANT NOTES:

- If the `<upload-jar-and-golden-standard (true/false)>` is set to true, The application automatically uploads the input JAR and the golden standard file provided in the `inputs.txt` file to the input bucket provided in the `inputs.txt` file.
- When the job is finished the output result and log-files will be automatically downloaded to the directory the `java -jar WordPairSimilarityRunner.jar` was ran from.

Communication And Statistics:

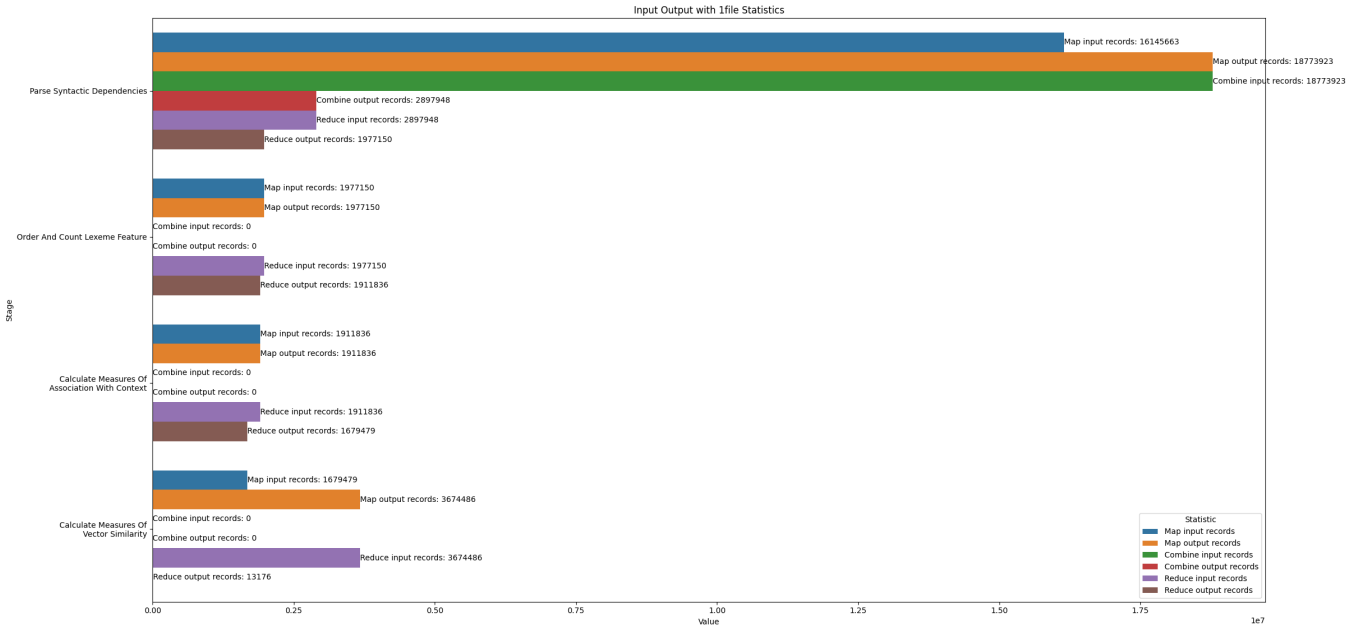
By using the python script present in the `statistics` folder of the project under the name `plot_syslog.py`,
we compared the communication and input-output/bytes statistics from both of the expirementes performed (1 corpus file vs. 15 corpus files) and the following statstics charts and counters were produced from the output log-files:

Using 1 File from the Google English Syntactic Biarcs corpus

Total lexemes read from corpus: 217575117 (`count(L)`)
Total features read from corpus: 227636582 (`count(F)`)

Total number of word-pairs classified in the golden standard: 13176

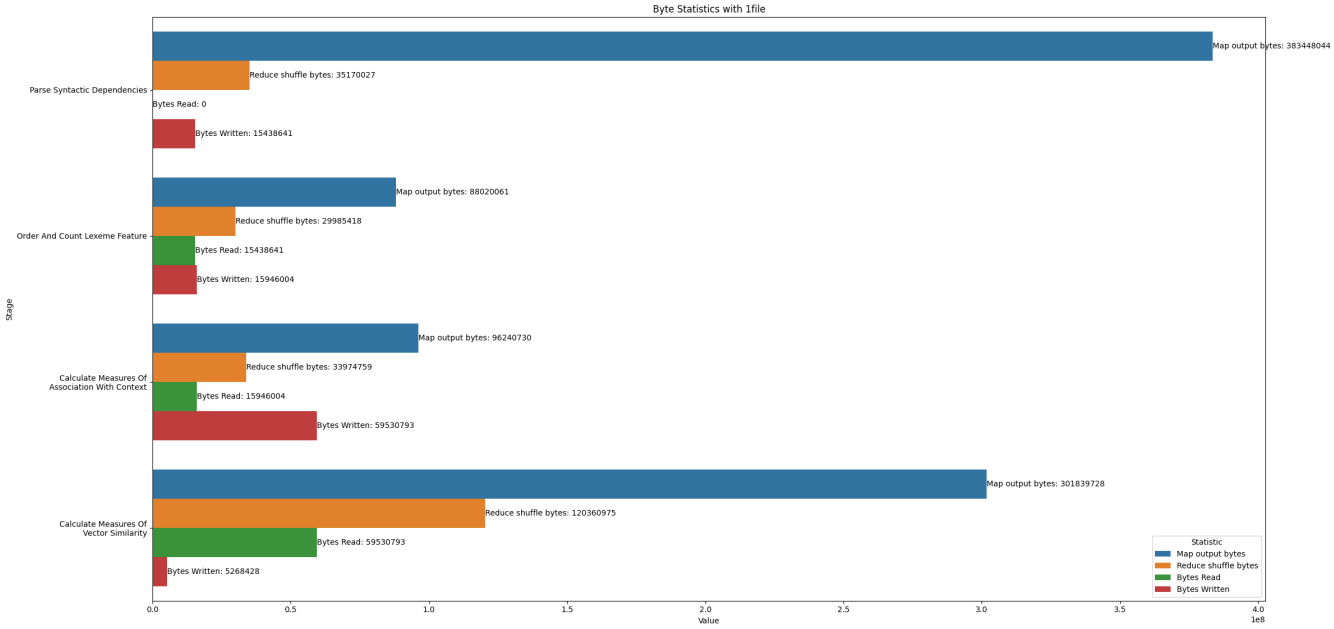
Input Output Records Statistics:



	Status	Statistic	Stage	Value
0	1file	Map input records	Parse Syntactic Dependencies	16145663
1	1file	Map output records	Parse Syntactic Dependencies	18773923
2	1file	Combine input records	Parse Syntactic Dependencies	18773923
3	1file	Combine output records	Parse Syntactic Dependencies	2897948
4	1file	Reduce input records	Parse Syntactic Dependencies	2897948
5	1file	Reduce output records	Parse Syntactic Dependencies	1977150
6	1file	Map input records	Order And Count Lexeme Feature	1977150
7	1file	Map output records	Order And Count Lexeme Feature	1977150
8	1file	Combine input records	Order And Count Lexeme Feature	0
9	1file	Combine output records	Order And Count Lexeme Feature	0
10	1file	Reduce input records	Order And Count Lexeme Feature	1977150
11	1file	Reduce output records	Order And Count Lexeme Feature	1911836
12	1file	Map input records	Calculate Measures Of Association With Context	1911836
13	1file	Map output records	Calculate Measures Of Association With Context	1911836
14	1file	Combine input records	Calculate Measures Of Association With Context	0
15	1file	Combine output records	Calculate Measures Of Association With Context	0
16	1file	Reduce input records	Calculate Measures Of Association With Context	1911836
17	1file	Reduce output records	Calculate Measures Of Association With Context	1679479
18	1file	Map input records	Calculate Measures Of Vector Similarity	1679479
19	1file	Map output records	Calculate Measures Of Vector Similarity	3674486
20	1file	Combine input records	Calculate Measures Of Vector Similarity	0
21	1file	Combine output records	Calculate Measures Of Vector Similarity	0
22	1file	Reduce input records	Calculate Measures Of Vector Similarity	3674486

	Status	Statistic	Stage	Value
23	1file	Reduce output records	Calculate Measures Of Vector Similarity	13176

Bytes Records Statistics:



if (lexemePairs != null) {

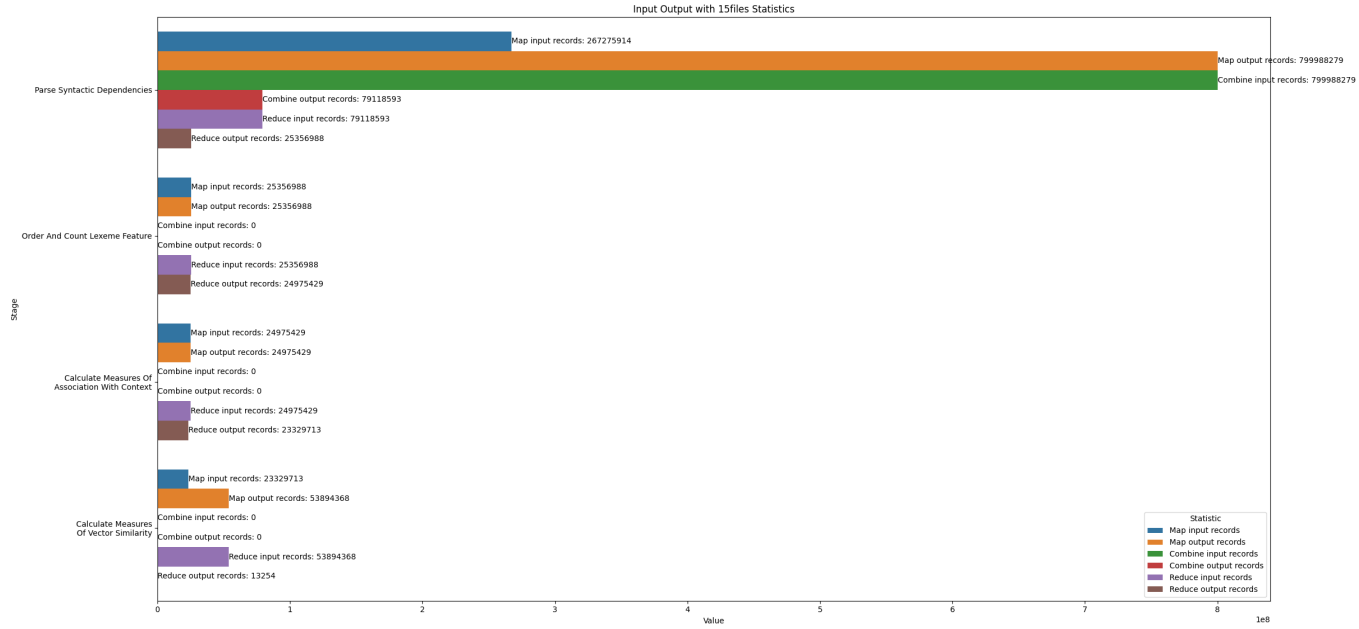
	Status	Statistic	Stage	Value
0	1file	Map output bytes	Parse Syntactic Dependencies	383448044
1	1file	Reduce shuffle bytes	Parse Syntactic Dependencies	35170027
2	1file	Bytes Read	Parse Syntactic Dependencies	0
3	1file	Bytes Written	Parse Syntactic Dependencies	15438641
4	1file	Map output bytes	Order And Count Lexeme Feature	88020061
5	1file	Reduce shuffle bytes	Order And Count Lexeme Feature	29985418
6	1file	Bytes Read	Order And Count Lexeme Feature	15438641
7	1file	Bytes Written	Order And Count Lexeme Feature	15946004
8	1file	Map output bytes	Calculate Measures Of Association With Context	96240730
9	1file	Reduce shuffle bytes	Calculate Measures Of Association With Context	33974759
10	1file	Bytes Read	Calculate Measures Of Association With Context	15946004
11	1file	Bytes Written	Calculate Measures Of Association With Context	59530793
12	1file	Map output bytes	Calculate Measures Of Vector Similarity	301839728
13	1file	Reduce shuffle bytes	Calculate Measures Of Vector Similarity	120360975
14	1file	Bytes Read	Calculate Measures Of Vector Similarity	59530793
15	1file	Bytes Written	Calculate Measures Of Vector Similarity	5268428

Using 15 Files from the Google English Syntactic Biarcs corpus

Total lexemes read from corpus: 8633837354 (`count(L)`)
Total features read from corpus: 10160658623 (`count(F)`)

Total number of word-pairs classified in the golden standard: 13254

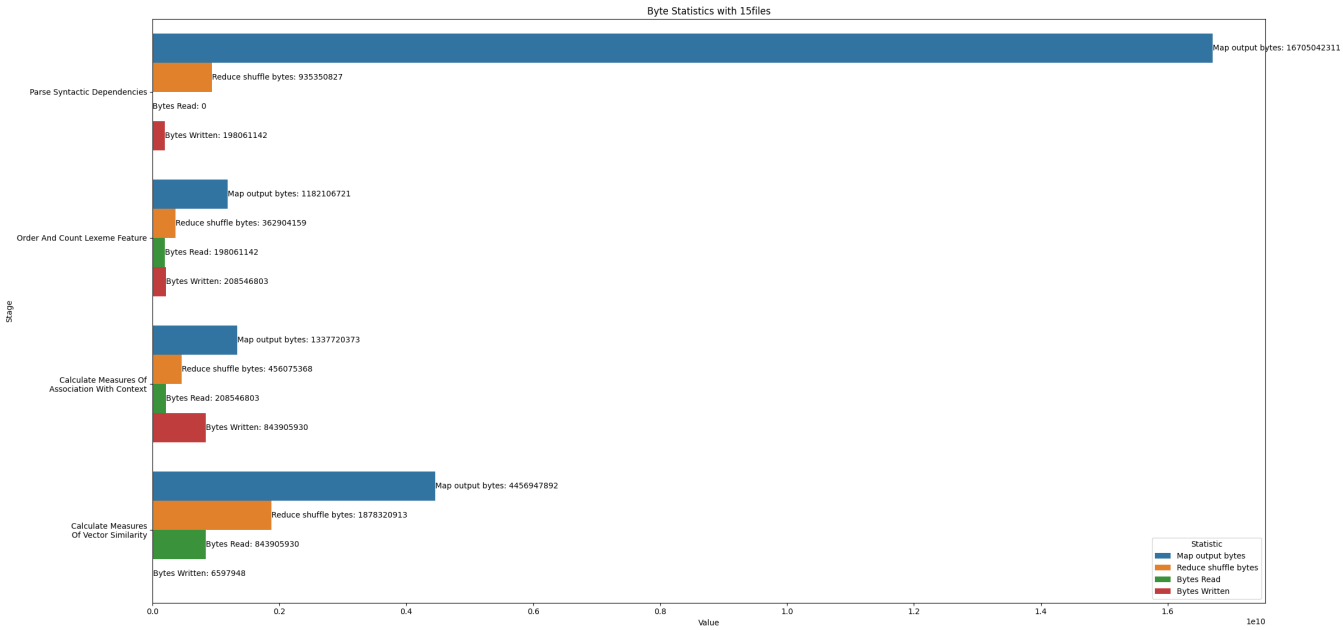
Input Output Records Statistics:



	Status	Statistic	Stage	Value
0	15files	Map input records	Parse Syntactic Dependencies	267275914
1	15files	Map output records	Parse Syntactic Dependencies	799988279
2	15files	Combine input records	Parse Syntactic Dependencies	799988279
3	15files	Combine output records	Parse Syntactic Dependencies	79118593
4	15files	Reduce input records	Parse Syntactic Dependencies	79118593
5	15files	Reduce output records	Parse Syntactic Dependencies	25356988
6	15files	Map input records	Order And Count Lexeme Feature	25356988
7	15files	Map output records	Order And Count Lexeme Feature	25356988
8	15files	Combine input records	Order And Count Lexeme Feature	0
9	15files	Combine output records	Order And Count Lexeme Feature	0
10	15files	Reduce input records	Order And Count Lexeme Feature	25356988
11	15files	Reduce output records	Order And Count Lexeme Feature	24975429
12	15files	Map input records	Calculate Measures Of Association With Context	24975429
13	15files	Map output records	Calculate Measures Of Association With Context	24975429
14	15files	Combine input records	Calculate Measures Of Association With Context	0
15	15files	Combine output records	Calculate Measures Of Association With Context	0
16	15files	Reduce input records	Calculate Measures Of Association With Context	24975429
17	15files	Reduce output records	Calculate Measures Of Association With Context	23329713
18	15files	Map input records	Calculate Measures Of Vector Similarity	23329713

	Status	Statistic	Stage	Value
19	15files	Map output records	Calculate Measures Of Vector Similarity	53894368
20	15files	Combine input records	Calculate Measures Of Vector Similarity	0
21	15files	Combine output records	Calculate Measures Of Vector Similarity	0
22	15files	Reduce input records	Calculate Measures Of Vector Similarity	53894368
23	15files	Reduce output records	Calculate Measures Of Vector Similarity	13254

Bytes Records Statistics:



	Status	Statistic	Stage	Value
0	15files	Map output bytes	Parse Syntactic Dependencies	16705042311
1	15files	Reduce shuffle bytes	Parse Syntactic Dependencies	935350827
2	15files	Bytes Read	Parse Syntactic Dependencies	0
3	15files	Bytes Written	Parse Syntactic Dependencies	198061142
4	15files	Map output bytes	Order And Count Lexeme Feature	1182106721
5	15files	Reduce shuffle bytes	Order And Count Lexeme Feature	362904159
6	15files	Bytes Read	Order And Count Lexeme Feature	198061142
7	15files	Bytes Written	Order And Count Lexeme Feature	208546803
8	15files	Map output bytes	Calculate Measures Of Association With Context	1337720373
9	15files	Reduce shuffle bytes	Calculate Measures Of Association With Context	456075368
10	15files	Bytes Read	Calculate Measures Of Association With Context	208546803
11	15files	Bytes Written	Calculate Measures Of Association With Context	843905930
12	15files	Map output bytes	Calculate Measures Of Vector Similarity	4456947892
13	15files	Reduce shuffle bytes	Calculate Measures Of Vector Similarity	1878320913
14	15files	Bytes Read	Calculate Measures Of Vector Similarity	843905930

	Status	Statistic	Stage	Value
15	15files	Bytes Written	Calculate Measures Of Vector Similarity	6597948

Project workflow and Map-Reduce design

The Map-Reduce design of the project includes 2 steps:

1. **Create Syntactic Measures Of Association (Includes 3 Map-Reduce Jobs)** - Creates the association measurments 4 co-occurrence vectors individual values for each lexeme and associated feature.
   Will be used as input for the following steps which groups those association measurments co-occurrence vector values into an interleaved iterable that contains the vector values for the 2 compared words.
2. **Create Golden Standard Similarity Vectors (Includes 1 Map-Reduce Job)** - Calculates and creates the 24 values similarity vector for the word pairs taken from the golden standard.

The first step is calculated for the entire given corpus input files (defined in the `inputs.txt` file).
You can also define that only the second step will run if the first step has already beeen claculated (also defined in the `inputs.txt` file).

Steps

1. Create Syntactic Measures Of Association -

Step Jobs (Total 3):

1. **Parse Syntactic Dependencies** - This map-reduce job takes the lines from the corpus files as input with the following shape:
   `head_word<TAB>syntactic-ngram<TAB>total_count<TAB>counts_by_year`
   where `syntactic-ngram` is a list of the syntactic sentence with the shape:
   `word/pos-tag/dep-label/head-index`
   The `syntactic-ngram` list is splitted and all the associated lexemes and features and their syntactic connection are emitted to the reducer with the associated `total_count` values.
   This Job also uses the reducer as a local aggregation **combiner**.

   Sentence/Words Pre-processing steps:
   The mapper for this jobs includes a few pre-processing steps that are used to lower the amount of key-value pairs emitted, generalize the lexeme and feature words by using a **Porter-Stemmer** and ignore irellevant data for the task.

   - **English words** - The following regex is used to allow only english words to be taken into considiration as lexemes/features: `[a-z-]+`
   - **Stop words removal** - The english stop words are filtered and not taken into account as lexemes/features. The ignored stop words can be found in: Stop Words Identifier
   - **Porter-Stemmer** - Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form generally a written word form. We use the Open-NLP Java libraries Porter-Stemmer in order to convert all lexemes/features into their stemmed form (e.g. `advantage -> advantag, communicating -> commun, communication -> commun`)

   Mapper
   Input-Output shape:

```
Input shape:
        key: <line-id>
        value: <head_word<TAB>syntactic-ngram<TAB>total_count<TAB>counts_by_year> (syntactic-ngram: [<word/pos-
tag/dep-label/head-index>]),
Output shape:
        key: <lexeme | feature | <lexeme, feature>>
        value: <total_count>
```

   The mapping process will also count the total amount of lexmes (`count(L)`) and the total amount of features (`count(F)`) in the corpus as Map-Reduce Counters.

   Mapper Output example:

```
1. allig        40  (alligator stemmed taken as lexeme)
2. beauti/nn    40  (beautiful stemmed taken as feature)
3. <allig,beauti/nn>    40 (the syntactic connection of alligator and beautiful)
4. crocodil     15  (crocodile stemmed taken as lexeme)
5. beauti/nn    15  (beautiful stemmed taken as feature)
6. strong/subj  15  (strong stemmed taken as feature)
7. <crocodil,beauti/nn>         15 (the syntactic connection of crocodile and beautiful)
8. <crocodil,strong/subj>       15 (the syntactic connection of crocodile and strong)
...
```

Reducer/Combiner

Input-Output shape:

```
Input shape:
        key: <lexeme | feature | <lexeme, feature>>,
        value: value: [counts]
Output shape:
        key: <lexeme | feature | <lexeme, feature>>
        value: value: <count(L=lexeme) | count(F=feature) | count(F=feature, L=lexeme)>
```

After mapping each lexeme/feature/syntactic connection, the reduce function will sum all the `total_count` values and the output of the reducer will be: Reducer Output example:

```
1. allig        40
2. beauti/nn    55 (Total beauti/nn feature appearence count)
3. <allig,beauti/nn>    40
4. crocodil     15
6. strong/subj  15
7. <crocodil,beauti/nn>         15
8. <crocodil,strong/subj>       15
...
```

2. **Order And Count Lexeme Feature** - This map-reduce job associates the lexemes/features/syntactic dependencies to their role and eventually outputs the **features as keys** with their corresponding counter values.

Mapper

Input-Output shape:

```
Input shape:
        key: <lexeme | feature | <lexeme, feature>>
        value: value: <count(L=lexeme) | count(F=feature) | count(F=feature, L=lexeme)>
Output shape:
        key: <<LEXEME, lexeme> | <FEATURE, feature> | <LEXEME_FEATURE, lexeme>>
        value: <<'L', count(L=lexeme)> | <'F', count(F=feature)> | <'LF', <feature, count(F=feature,
L=lexeme)>>>
```

Mapper Output example:

```
1. LEXEME       allig           L       40
2. FEATURE      beauti/nn       F       55
3. LEXEME_FEATURE       allig           LF      <beauti/nn,40>
4. LEXEME       crocodil        L       15
6. FEATURE      strong/subj     F       15
7. LEXEME_FEATURE       crocodil        LF      <beauti/nn,15>
8. LEXEME_FEATURE       crocodil        LF      <strong/subj,15>
...
```

Reducer

Input-Output shape:

```
Input shape:
        key: <<LEXEME, lexeme> | <FEATURE, feature> | <LEXEME_FEATURE, lexeme>>
        value: <<'L', count(L=lexeme)> | <'F', count(F=feature)> | <'LF', <feature, count(L=lexeme,
F=feature)>>>
```

```
Output shape:
        key: <<FEATURE, feature> | <LEXEME_FEATURE, feature>>
        value: <count(F=feature) | <lexeme, count(F=feature, L=lexeme), count(L=lexeme)>>
```

The mapper output is partitioned by the secondary key values (the lexeme/feature) so that `<LEXEME_FEATURE, lexeme>` and `<LEXEME, lexeme>` will arrive to same reducer.
It is sorted by the lexeme/feature values and then by the LEXEME/FEATURE/LEXEME_FEATURE tags so that `<LEXEME | FEATURE>` tags will arrive before `<LEXEME_FEATURE>` tag for every equal lexeme. And finally grouped by the lexeme/feature value so that the iterable will contain only values associated with the same lexeme/feature values.

Reducer Output example:

```
1. FEATURE      beauti/nn       55
2. LEXEME_FEATURE       beauti/nn       <allig,40,40>
3. FEATURE      strong/subj     15
4. LEXEME_FEATURE       beauti/nn       <crocodil,15,15>
5. LEXEME_FEATURE       strong/subj     <crocodil,15,15>
...
```

3. **Calculate Measures Of Association With Context** - This map-reduce job is responsible for outputing the co-occurence vector values for every syntactic association of lexemes and features.
The mapper emits the tagged input values and the reducer recieves the entire information for calculating the 4 measures of association measures (`count(F=f)` , `count(L=l)` , `count(F=f, L=l)` and the global counters aggregated in the first map-reduce job, `count(L)` and `count(F)`).
The final output of this job is the lexeme value as key associated with the ***feature and the Plain Frequncy, Relative Frequncy, PMI and T-Test measures***.

Mapper
Input-Output shape:

```
Input shape:
        key: <<FEATURE, feature> | <LEXEME_FEATURE, feature>>
        value: <count(F=feature) | <lexeme, count(F=feature, L=lexeme), count(L=lexeme)>>
Output shape:
        key: <<FEATURE, feature> | <LEXEME_FEATURE, feature>>
        value: <<'F', count(F=feature)> | <'LF', <lexeme, count(F=feature, L=lexeme), count(L=lexeme)>>>
```

Mapper Output example:

```
1. FEATURE      beauti/nn       F       55
2. LEXEME_FEATURE       beauti/nn       LF      <allig,40,40>
3. FEATURE      strong/subj     F       15
4. LEXEME_FEATURE       beauti/nn       LF      <crocodil,15,15>
5. LEXEME_FEATURE       strong/subj     LF      <crocodil,15,15>
...
```

Reducer
Input-Output shape:

```
Input shape:
        key: <<FEATURE, feature> | <LEXEME_FEATURE, feature>>
        value: <<'F', count(F=feature)> | <'LF', <lexeme, count(F=feature, L=lexeme), count(L=lexeme)>>>
Output shape:
        key: <lexeme>
        value: <<lexeme, feature, plain-frequency, relative-frequency, pmi, t-test>>
```

The mapper output is partitioned by the secondary key values (the feature) so that `<LEXEME_FEATURE, feature>` and `<FEATURE, feature>` will arrive to same reducer.
It is sorted by the feature values and then by the FEATURE/LEXEME_FEATURE tags so that `<FEATURE>` tags will arrive before `<LEXEME_FEATURE>` tag for every equal feature. And finally grouped by the feature value so that the iterable will contain only values associated with the same feature value.

Reducer Output example:

```
1. allig        <allig,beauti/nn,40.0,1.0,0.3479,0.208>
2. crocodil     <crocodil,beauti/nn,15.0,1.0,0.349,0.125>
3. crocodil     <crocodil,strong/subj,15.0,1.0,2.22,0.886>
...
```

NOTE: We do not create 4 seperate vectors in this job but we create the measurments assocation values for the same positions in all of the 4 vectors so that each value in the output indicated the same index in all of the 4 vectors and the relevant values in this index of the vector.

2. Create Golden Standard Similarity Vectors -

Step Jobs (Total 1):

1. **Calculate Measures Of Vector Similarity** - This map-reduce job maps the measures of association co-occurrence vector values for each pair of words in the golden standard to an interleaved co-occurrence vectors for the pair of words and the reducer then calculates the 24 similarity vector measure values by iterating the interleaved vectors.
   NOTE:
   - the co-occurrence vectors are in size of the entire feature set taken from the corpus but it is sparse so that it conatains a lot of zero values that are not taken into account while calculating similarity.
   - The golden standard is loaded into memory (**Only memory assumption**) so that we can check if word pairs are present in it (and their pre-defined true/false classification).

   The interleaved vectors are sorted so that the compared lexemes are sorted alphabitcally and the feature values are also sorted alphabitcally so that each co-occurrence vector value is calculated with its counterpart from the other lexeme vector value.
   If only one of the co-occurrence vectors contains a certain feature, the value of that feature is calculated as zero for the other vector.

   Mapper
   Input-Output shape:

```
Input shape:
        key: <lexeme>
        value: <<lexeme, feature, plain-frequency, relative-frequency, pmi, t-test>>
Output shape:
        key: <<lexeme1, lexeme2>, lexeme, feature> (<lexeme1, lexeme2> from golden standard)
        value: <<lexeme, feature, plain-frequency, relative-frequency, pmi, t-test>>
```

   Mapper Output example (assuming the the pair <alligator,crocodile> is present in the golden standard):

```
1. <alligator,crocodile>        allig           beauti/nn       <allig,beauti/nn,40.0,1.0,0.3479,0.208>
2. <alligator,crocodile>        crocodil        beauti/nn       <crocodil,beauti/nn,15.0,1.0,0.349,0.125>
3. <alligator,crocodile>        crocodil        strong/subj     <crocodil,strong/subj,15.0,1.0,2.22,0.886>
...
```

   Reducer
   Input-Output shape:

```
Input shape:
        key: <<lexeme1, lexeme2>, lexeme, feature> (<lexeme1, lexeme2> from golden standard)
        value: <<lexeme, feature, plain-frequency, relative-frequency, pmi, t-test>>
Output shape:
        key: <<lexeme1, lexeme2>>
        value: Similarity vector of size 24 between lexeme1 and lexeme2
```

   The mapper output is partitioned by the sorted word pair text (e.g. <alligator,crocodile>, sorted to avoid multiplication) so that each pair in the comparison will end up in the same reducer. It is sorted by the word pair text values and then by the feature values(to make sure the co-occurrence vectors are structured simillarly) and then by the compared lexeme (e.g. allig/crocodil, breaks ties with the alphabetically smaller lexeme to maintain order of calculation in the vectors). And finally grouped by the word pair text values so that the iterable will contain only values associated with the same compared word pair.
   NOTE: The calculation of the similarity measures is done by iterating the interleaved co-occurrence vectors using two pointers to determine if the evaluated features and lexemes are equal so that the appropriate calculation will be done for the specific feature (counted as zero in the other vector or not).

Reducer Output example:

```
1. <alligator,crocodile>        [44606.0, 3703.660081594962, 0.42162927787038723, 0.07379568106312293,
0.13744827319487954, 27658.996564902867, 1.7285574188242916, 0.13773672867460315, 0.42162927787038734,
0.15252876095224707, 0.26468538767956523, 1.0377248242623596, 5815.402758593203, 253.2698158672323,
0.20689536224570676, 0.09511245893155527, 0.1737035464364117, 3961.7051470371716, 4.155266637310252,
0.38677456515249947, 0.1200093534609998, 0.06646645654403041, 0.1246480020748524, 2.701911515411438]    false
```

Classification Results

Comparing the results from both of the expirementes performed (1 corpus file vs. 15 corpus files):

Using 1 file from the Google English Syntactic Biarcs corpus:

```
WEKA Random Forest Classifier Results
=============================================

Correctly Classified Instances       12120               91.9854 %
Incorrectly Classified Instances      1056                8.0146 %
Kappa statistic                          0.3682
K&B Relative Info Score                 -5.067  %
K&B Information Score                 -314.7848 bits     -0.0239 bits/instance
Class complexity | order 0            6212.39   bits      0.4715 bits/instance
Class complexity | scheme            77568.2153 bits      5.8871 bits/instance
Complexity improvement     (Sf)     -71355.8253 bits     -5.4156 bits/instance
Mean absolute error                      0.1411
Root mean squared error                  0.2639
Relative absolute error                 77.8398 %
Root relative squared error             87.6636 %
Total Number of Instances            13176

F1 Measure: 0.39931740614334477
Precision: 0.8162790697674419
Recall: 0.26430722891566266

True-Positive rate: 0.26430722891566266
False-Positive rate: 0.006667792032410533
True-Negative rate: 0.9933322079675895
False-Negative rate: 0.7356927710843374

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                0.993    0.736    0.923      0.993    0.957      0.437    0.777     0.957     FALSE
                0.264    0.007    0.816      0.264    0.399      0.437    0.777     0.474     TRUE
Weighted Avg.   0.920    0.662    0.913      0.920    0.901      0.437    0.777     0.908

=== Confusion Matrix ===

     a     b   <-- classified as
 11769    79 |     a = FALSE
   977   351 |     b = TRUE

=============================================
```

Using 15 files from the Google English Syntactic Biarcs corpus:

```
WEKA Random Forest Classifier Results
=============================================

Correctly Classified Instances       12386               93.451  %
Incorrectly Classified Instances       868                6.549  %
Kappa statistic                          0.5123
K&B Relative Info Score                 15.6321 %
K&B Information Score                  976.9473 bits      0.0737 bits/instance
Class complexity | order 0            6249.6044 bits      0.4715 bits/instance
Class complexity | scheme            20789.1311 bits      1.5685 bits/instance
Complexity improvement     (Sf)     -14539.5267 bits     -1.097  bits/instance
```

```
Mean absolute error                      0.1198
Root mean squared error                  0.2318
Relative absolute error                 66.092  %
Root relative squared error             77.0099 %
Total Number of Instances            13254

F1 Measure: 0.5412262156448203
Precision: 0.920863309352518
Recall: 0.38323353293413176

True-Positive rate: 0.38323353293413176
False-Positive rate: 0.0036918946131901326
True-Negative rate: 0.9963081053868099
False-Negative rate: 0.6167664670658682

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.996    0.617    0.935      0.996    0.965      0.570    0.901     0.984     FALSE
                 0.383    0.004    0.921      0.383    0.541      0.570    0.901     0.679     TRUE
Weighted Avg.    0.935    0.555    0.934      0.935    0.922      0.570    0.901     0.953

=== Confusion Matrix ===

     a     b   <-- classified as
 11874    44 |     a = FALSE
   824   512 |     b = TRUE

================================================
```

True/False-Positive/Negative explained:

True-Positive - is an outcome where the model correctly predicts the positive class.
True-Negative - is an outcome where the model correctly predicts the negative class.
False-Positive - is an outcome where the model incorrectly predicts the positive class.
False-Negative - is an outcome where the model incorrectly predicts the negative class.

Conclusions:

We can see that using more corpus files managed to improve the Correctly Classified Instances from 91.9854% to 93.451% when using 15 corpus files instead of the 1 file used in the first expirement (and to also decrease the Incorrectly Classified Instances from 8.0146% to 6.549%).
We can conclude that using more information produced a more accurate and distinguishable similarity vectors that the classifier classified more accurately.
We can also see that the precision, recall and F1 measure improved between the two expirements.

When running on 1 corpus file we got:

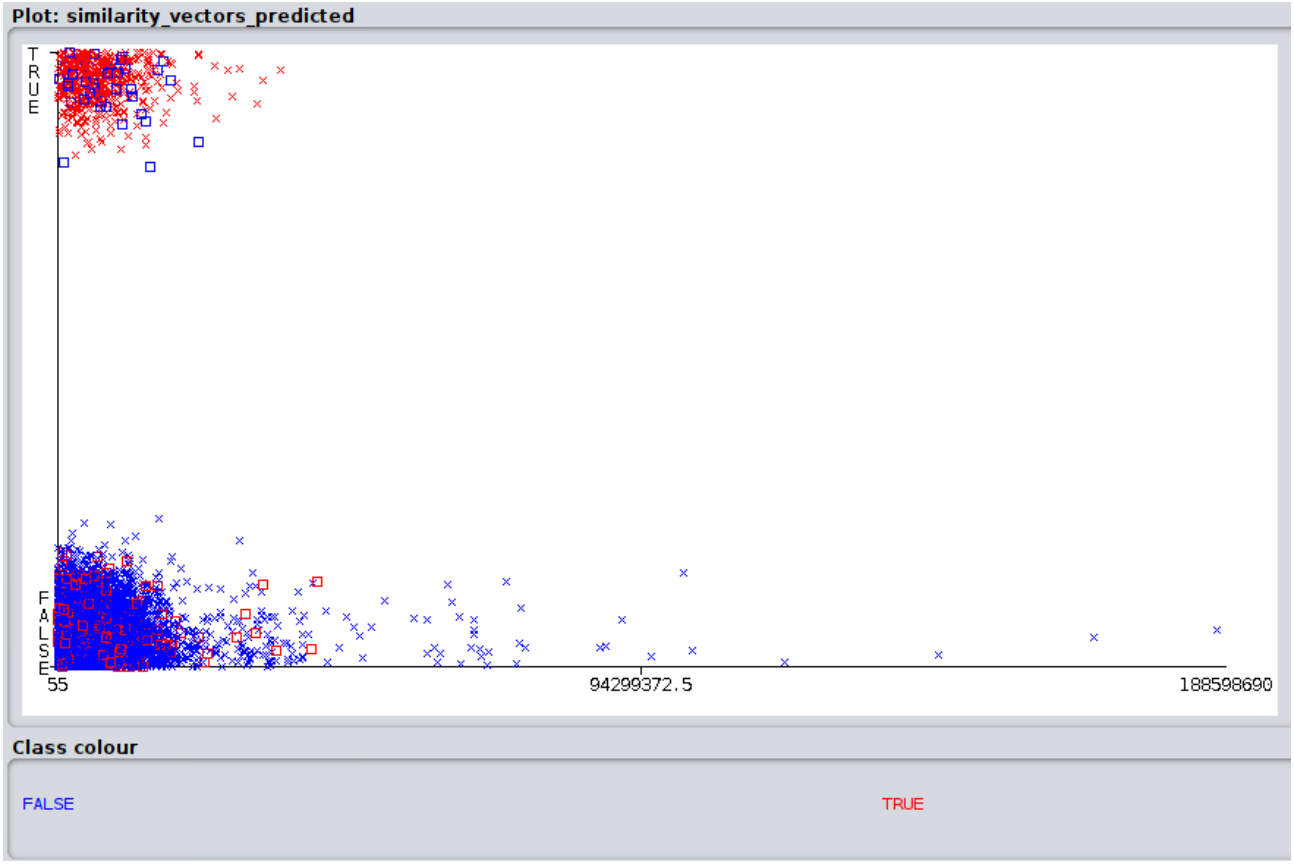
```
F1 Measure: 0.39931740614334477
Precision: 0.8162790697674419
Recall: 0.26430722891566266
```

While when running on 15 corpus files we got:

```
F1 Measure: 0.5412262156448203
Precision: 0.920863309352518
Recall: 0.38323353293413176
```

15 Corpus Files Classification Results Distribution Visualization:

(Using WEKA GUI, link in the Examples And Resources section)



Results Analysis

By using the python script present in the `analysis` folder of the project under the name `word_pair_vector_analysis.py`, and by using the WEKA GUI predictions visualizer (link in the Examples And Resources section) we chose 3 noun-pairs under each type of classification (False-Positive, False-Negative, True-Positive, True-Negative) and analyzed the features composing their co-occurrence vector.
By doing so we can gain a better understanding for the False predictions.
The results analysis was performed on the results of running the classifier on the **15 corpus files** input.

NOTE: True meaning is that word pair classified as similar while False meaning is that word pair classified non-similar.

True-Postiive predicted noun-pairs: (Actual: True, Predicted: True):

```
knife, tool
gun, rifle
beet, food
```

True-Negative predicted noun-pairs: (Actual: False, Predicted: False):

```
cocktail, van
chemical, pear
freezer, investigation
```

False-Positive predicted noun-pairs: (Actual: False, Predicted: True):

```
Word pair: barrel, revolver
Common feature count (intersection): 189
Uncommon feature count (symetric difference): 4096

Common features: 'fit/conj', 'hand/dep', 'fast/advmod', 'around/prep', 'bought/rcmod', 'take/rcmod', 'unit/conj',
'distanc/dep', 'grip/rcmod', 'stuck/partmod', 'ax/conj', 'sword/nn', 'stick/partmod', 'ly/partmod', 'mani/amod',
'therefor/advmod', 'rusti/amod', 'littl/amod', 'extra/amod', 'barrel/dep', 'probabl/advmod', 'long/amod',
```

```
'huge/amod', 'good/amod', 'five/dep', 'shoot/rcmod', 'big/amod', 'carefulli/advmod', 'blade/conj', 'first/amod',
'full/amod', 'given/partmod', 'leav/rcmod', 'bomb/conj', 'suppli/conj', 'deadli/amod', 'look/dep', 'rifl/conj',
'dai/dobj', 'plate/dep', 'steel/dep', 'revolv/nn', 'clear/conj', 'fallen/partmod', 'convers/nsubj', 'fill/ccomp',
'without/prep', 'long/advmod', 'toward/prep', 'rest/partmod', 'held/partmod', 'either/advmod', 'unload/amod',
'larg/amod', 'contain/rcmod', 'articl/conj', 'near/prep', 'sixteen/num', 'found/partmod', 'shot/nn', 'stick/conj',
'onli/advmod', 'beneath/prep', 'modern/amod', 'blue/amod', 'caus/rcmod', 'load/amod', 'shot/dep', 'heavi/amod',
'given/rcmod', 'thei/nsubj', 'torch/conj', 'outsid/prep', 'marri/ccomp', 'numer/amod', 'readi/amod', 'half/dep',
'rest/conj', 'longer/advmod', 'size/amod', 'three/num', 'upon/prep', 'beyond/prep', 'speed/dobj', 'wai/dobj',
'ammunit/conj', 'splendid/amod', 'wa/ccomp', 'close/amod', 'two/num', 'round/prep', 'box/conj', 'drum/dobj',
'lai/rcmod', 'new/amod', 'third/conj', 'gleam/amod', 'beauti/amod', 'soon/advmod', 'taken/rcmod', 'steel/nn',
'wa/conj', 'readi/dep', 'cost/partmod', 'make/conj', 'along/prep', 'enough/advmod', 'sword/conj', 'like/prep',
'german/amod', 'togeth/advmod', 'done/rcmod', 'american/amod', 'befor/prep', 'rifl/nn', 'stout/amod', 'excel/amod',
'n/prep', 'armi/nn', 'barrel/dobj', 'protrud/partmod', 'apiec/advmod', 'pistol/conj', 'round/conj', 'hung/rcmod',
'still/advmod', 'roll/xcomp', 'dure/prep', 'empti/amod', 'abov/prep', 'first/advmod', 'sever/amod', 'point/amod',
'awai/advmod', 'peep/partmod', 'next/advmod', 'forward/advmod', 'wa/rcmod', 'pretend/advcl', 'ha/rcmod',
'advanc/conj', 'door/dobj', 'daili/advmod', 'annual/advmod', 'saw/conj', 'carbin/nn', 'arm/conj', 'gun/dep',
'gun/conj', 'show/rcmod', 'bullet/conj', 'point/partmod', 'mill/nsubj', 'within/prep', 'govern/nn', 'veri/amod',
'avoid/infmod', 'new/dep', 'hand/conj', 'call/partmod', 'carri/rcmod', 'calib/nn', 'around/advmod', 'old/amod',
'prepar/rcmod', 'also/advmod', 'rather/cc', 'accord/prep', 'english/amod', 'knive/conj', 'cartridg/conj',
'instrument/conj', 'nose/dep', 'great/amod', 'revolv/conj', 'back/advmod', 'among/prep', 'year/dep', 'mere/amod',
'hold/rcmod', 'dozen/num', 'calib/dep', 'small/amod', 'crown/conj', 'bait/dep', 'hammer/conj', 'enorm/amod',
'level/amod', 'eight/num'
```

We can see that the two words 4% of their unified features as common features.
Semantically a barrel is a part of the revolver, therefore the classification we got in this case could have been classified as True as part of the golden standard.

```
Word pair: food, stove
Common feature count (intersection): 202
Uncommon feature count (symetric difference): 5076

Common features: 'also/advmod', 'resembl/amod', 'bed/conj', 'hot/amod', 'requir/rcmod', 'veri/amod',
'prevent/infmod', 'seen/rcmod', 'heavi/amod', 'oil/nn', 'special/amod', 'cool/infmod', 'pan/conj', 'russian/amod',
'sourc/conj', 'style/dep', 'avail/amod', 'form/rcmod', 'show/rcmod', 'condens/amod', 'applianc/conj', 'water/conj',
'serv/rcmod', 'ordinari/amod', 'go/infmod', 'fire/conj', 'fuel/dep', 'dish/conj', 'larg/amod', 'keep/infmod',
'us/rcmod', 'even/advmod', 'maintain/rcmod', 'persist/rcmod', 'green/amod', 'product/conj', 'nine/num',
'pressur/amod', 'stove/conj', 'came/rcmod', 'regular/amod', 'anim/nn', 'take/infmod', 'soft/nn', 'iron/nn',
'kitchen/nn', 'de/nn', 'dry/nn', 'two/num', 'thei/nsubj', 'ga/nn', 'either/preconj', 'call/partmod', 'coal/nn',
'air/conj', 'effici/amod', 'old/amod', 'nearli/advmod', 'three/num', 'plate/conj', 'bought/rcmod', 'fork/conj',
'miser/amod', 'thing/conj', 'stolen/amod', 'complet/advmod', 'great/amod', 'oil/dep', 'stand/rcmod', 'sever/amod',
'emploi/partmod', 'sort/conj', 'pot/conj', 'wonder/amod', 'heat/rcmod', 'tabl/conj', 'us/partmod', 'portabl/amod',
'heat/nn', 'low/amod', 'iron/conj', 'cook/nn', 'littl/amod', 'cook/dep', 'fuel/nn', 'modern/amod', 'alcohol/nn',
'save/rcmod', 'done/rcmod', 'fit/rcmod', 'common/amod', 'onion/conj', 'close/amod', 'unus/amod', 'stew/conj',
'readi/amod', 'onli/advmod', 'tropic/amod', 'find/rcmod', 'befor/prep', 'call/dep', 'compani/conj', 'car/nn',
'cheap/amod', 'reduc/rcmod', 'heat/amod', 'brick/conj', 'french/amod', 'four/num', 'thing/appos', 'cold/amod',
'seem/rcmod', 'camp/nn', 'satisfactori/amod', 'shoe/conj', 'cook/rcmod', 'true/amod', 'german/amod', 'whose/rcmod',
'fine/amod', 'cloak/conj', 'knive/conj', 'tradit/amod', 'excel/amod', 'place/rcmod', 'beauti/amod', 'pellet/amod',
'color/amod', 'ar/rcmod', 'behind/prep', 'stir/partmod', 'provid/partmod', 'set/partmod', 'place/partmod',
'convent/amod', 'patent/nn', 'mani/amod', 'york/appos', 'black/amod', 'dress/partmod', 'second/amod', 'good/amod',
'white/amod', 'like/prep', 'firewood/conj', 'beneath/prep', 'first/amod', 'water/nn', 'utensil/conj', 'cook/conj',
'long/amod', 'equip/conj', 'candl/conj', 'time/dep', 'jar/conj', 'japanes/amod', 'anyth/conj', 'facil/conj',
'made/partmod', 'dirti/amod', 'liquid/amod', 'smell/dep', 'keep/rcmod', 'glow/amod', 'condit/conj', 'ga/conj',
'greas/conj', 'food/appos', 'type/dep', 'design/partmod', 'finest/amod', 'cast/conj', 'small/amod', 'allow/rcmod',
'bit/appos', 'famili/nn', 'plenti/conj', 'dry/amod', 'red/amod', 'burn/rcmod', 'wa/rcmod', 'home/amod',
'third/amod', 'place/conj', 'blanket/conj', 'new/amod', 'room/conj', 'help/rcmod', 'soon/advmod', 'real/amod',
'gasolin/nn', 'boil/rcmod', 'cat/appos', 'domest/amod', 'warm/rcmod', 'household/nn', 'oven/conj', 'attain/rcmod',
'warm/amod', 'cook/infmod', 'five/num', 'arrang/conj'
```

We can see that the two words 3.8% of their unified features as common features.
Food is cooked on stove, therfore, we can make sense of their appearence with common features in the corpus.
The calssification here is wrong because food is not similar to stove but because of the similarities and world of interest we can see why the classifier classified those two words as similar by taking into account their sentence syntactic meaninings.

```
Word pair: hospital, school
Common feature count (intersection): 1642
Uncommon feature count (symetric difference): 8702

Common features: 'edit/dep', 'chelsea/nn', 'polit/amod', 'entir/amod', 'within/prep', 'd/nn', 'time/dobj',
'colour/amod', 'chiefli/advmod', 'christian/amod', 'cincinnati/appos', 'servic/dep', 'joint/nn', 'start/rcmod',
'camp/conj', 'brooklyn/nn', 'agricultur/nn', 'previous/advmod', 'store/conj', 'middlesex/nn', 'full/amod',
```

'friendli/dep', 'accommod/conj', 'typic/amod', 'road/appos', 'conduct/dep', 'methodist/nn', 'equip/dep',
'total/amod', 'indian/amod', 'north/nn', 'nearli/advmod', 'orient/dep', 'contain/partmod', 'equip/amod',
'librari/conj', 'philadelphia/appos', 'swedish/nn', 'white/nn', 'anti/dep', 'thirti/num', 'children/conj',
'feder/dep', 'unit/conj', 'intellig/conj', 'said/amod', 'jackson/nn', 'richmond/appos', 'southern/amod',
'attend/infmod', 'across/prep', 'univers/conj', 'final/advmod', 'four/num', 'washington/nn', 'fe/nn', 'lead/amod',
'work/partmod', 'parish/conj', 'flower/nn', 'great/nn', 'true/amod', 'nottingham/nn', 'miscellan/amod',
'work/rcmod', 'grown/rcmod', 'estat/nn', 'agre/rcmod', 'engag/partmod', 'neither/preconj', 'boston/nn',
'counti/conj', 'seven/num', 'workshop/conj', 'previou/amod', 'specialti/nn', 'grove/nn', 'particular/amod',
'support/dep', 'matern/amod', 'fairfax/nn', 'oxford/nn', 'utica/nn', 'lakeland/nn', 'staf/partmod', 'florida/nn',
'novemb/appos', 'john/nn', 'review/dep', 'voluntari/amod', 'rest/conj', 'ucla/nn', 'vast/amod', 'post/conj',
'expens/amod', 'eventu/advmod', 'whole/amod', 'befor/advmod', 'street/nn', 'cambridg/nn', 'report/partmod',
'room/conj', 'hotel/conj', 'todai/advmod', 'german/nn', 'locat/partmod', 'viii/nn', 'british/nn', 'anderson/nn',
'school/dep', 'known/amod', 'vallei/nn', 'health/nn', 'assist/nn', 'characterist/conj', 'red/nn', 'paso/nn',
'louisvil/nn', 'involv/rcmod', 'cincinnati/nn', 'villag/conj', 'corrupt/amod', 'us/rcmod', 'chief/amod',
'equip/rcmod', 'manor/nn', 'usa/appos', 'hancock/nn', 'nation/amod', 'least/amod', 'veterinari/amod',
'chang/infmod', 'bath/nn', 'prevail/rcmod', 'ago/advmod', 'necessari/amod', 'survei/partmod', 'manag/nn',
'eastern/nn', 'willard/nn', 'elabor/amod', 'usual/advmod', 'booth/nn', 'canton/nn', 'special/rcmod',
'suffici/advmod', 'feder/amod', 'reveal/rcmod', 'servic/conj', 'decemb/appos', 'edinburgh/nn', 'serv/partmod',
'project/conj', 'cottag/nn', 'monasteri/nn', 'begun/rcmod', 'call/dep', 'wisconsin/nn', 'leagu/conj', 'french/amod',
'administ/rcmod', 'american/nn', 'us/conj', 'ibadan/appos', 'member/nn', 'park/nn', 'natur/conj', 'control/partmod',
'onli/preconj', 'sinai/nn', 'hebrew/nn', 'georg/nn', 'onli/num', 'sai/rcmod', 'central/nn', 'avenu/appos',
'formerli/advmod', 'work/conj', 'indiana/nn', 'presbyterian/nn', 'support/amod', 'victoria/nn', 'design/amod',
'happi/conj', 'model/nn', 'johnston/nn', 'left/partmod', 'almshous/conj', 'hous/conj', 'sydnei/nn', 'ancient/amod',
'colleg/nn', 'height/nn', 'hamburg/nn', 'respond/amod', 'western/nn', 'run/dep', 'list/partmod', 'love/conj',
'certain/amod', 'asylum/conj', 'miami/nn', 'build/nn', 'give/partmod', 'pre/dep', 'sixteen/num', 'servic/nn',
'manitoba/nn', 'consid/infmod', 'told/rcmod', 'recommend/appos', 'locat/rcmod', 'negro/nn', 'smaller/amod',
'wood/nn', 'largest/amod', 'urban/amod', 'london/nn', 'emploi/rcmod', 'leisur/conj', 'saint/nn', 'wai/conj',
'counti/appos', 'avenu/nn', 'memori/nn', 'perth/nn', 'trivial/amod', 'oversea/advmod', 'genuin/amod',
'nonprofit/amod', 'hous/appos', 'later/advmod', 'posit/appos', 'grace/amod', 'francisco/nn', 'inner/amod',
'board/appos', 'segreg/amod', 'lake/nn', 'cook/nn', 'lane/nn', 'cleveland/appos', 'rd/appos', 'roman/amod',
'alfr/nn', 'c/conj', 'critic/amod', 'augusta/nn', 'work/nn', 'parish/nn', 'babi/nn', 'lima/nn', 'equival/conj',
'medic/amod', 'attitud/conj', 'therapi/nn', 'sure/amod', 'provid/rcmod', 'particularli/advmod', 'theater/conj',
'bryan/nn', 'mention/partmod', 'divis/conj', 'postgradu/amod', 'ten/num', 'forth/conj', 'simplic/conj',
'seek/partmod', 'make/partmod', 'peter/nn', 'northwestern/nn', 'mohammedan/amod', 'toward/prep', 'workhous/conj',
'appropri/amod', 'govern/nn', 'bombai/nn', 'open/amod', 'mari/nn', 'gave/rcmod', 'carmel/nn', 'dental/amod',
'beyond/prep', 'todai/nn', 'marion/nn', 'support/partmod', 'watertown/nn', 'larger/amod', 'dai/dep', 'church/conj',
'factori/nn', 'independ/amod', 'associ/conj', 'point/nn', 'anyth/conj', 'univers/dep', 'citi/nn', 'chiropract/amod',
'missionari/amod', 'hous/nn', 'allen/nn', 'oper/partmod', 'seat/conj', 'new/amod', 'son/appos', 'stabl/conj',
'alabama/nn', 'promin/amod', 'member/amod', 'littl/measure', 'build/conj', 'outsid/amod', 'trust/nn',
'come/partmod', 'compulsori/amod', 'enjoi/rcmod', 'set/conj', 'time/nn', 'requir/rcmod', 'road/conj', 'pari/appos',
'fame/amod', 'partial/amod', 'york/nn', 'dear/amod', 'b/nn', 'manag/conj', 'urban/nn', 'infirmari/conj',
'royal/amod', 'nine/num', 'jewish/nn', 'modern/nn', 'public/dep', 'poorhous/conj', 'health/conj', 'enorm/amod',
'jame/nn', 'town/nn', 'specif/amod', 'duke/nn', 'children/nsubjpass', 'relat/dep', 'earlier/advmod', 'second/nn',
'laboratori/conj', 'p/dep', 'admit/infmod', 're/dep', 'armi/nn', 'center/conj', 'charit/nn', 'social/amod',
'institut/conj', 'improv/rcmod', 'much/advmod', 'east/nn', 'import/amod', 'oper/rcmod', 'tampa/nn', 'fit/rcmod',
'manila/appos', 'find/rcmod', 'examin/rcmod', 'jungl/nn', 'chapter/nn', 'charl/nn', 'pacif/nn', 'sai/partmod',
'entertain/conj', 'want/rcmod', 'tradit/amod', 'gradi/nn', 'milwauke/nn', 'human/conj', 'abov/amod',
'allopath/amod', 'base/prep', 'homeopath/amod', 'commun/conj', 'look/partmod', 'held/rcmod', 'british/amod',
'princess/nn', 'turner/nn', 'help/partmod', 'bai/nn', 'januari/appos', 'holi/amod', 'nearli/num',
'california/appos', 'look/rcmod', 'columbu/appos', 'claim/conj', 'preach/conj', 'care/conj', 'pass/rcmod',
'mediev/amod', 'playground/conj', 'greenwich/nn', 'region/nn', 'dispensari/conj', 'personnel/conj', 'coloni/conj',
'naval/nn', 'school/appos', 'militari/nn', 'earli/amod', 'cent/appos', 'teach/nn', 'twelv/num', 'canadian/nn',
'neighbor/amod', 'fashion/dep', 'rather/cc', 'thank/appos', 'angel/appos', 'deaf/amod', 'affili/dep', 'poor/amod',
'chariti/conj', 'attach/partmod', 'offici/amod', 'russian/amod', 'equip/partmod', 'establish/nn', 'ii/nn',
'subject/amod', 'recogn/rcmod', 'perpetu/amod', 'birmingham/nn', 'except/prep', 'sixti/num', 'qualiti/nn',
'anim/nn', 'reach/rcmod', 'singl/amod', 'make/rcmod', 'mai/appos', 'poorest/amod', 'accord/partmod', 'follow/prep',
'offic/conj', 'thing/conj', 'particip/amod', 'verit/amod', 'exist/amod', 'madison/nn', 'us/partmod', 'santo/nn',
'adjac/amod', 'hospit/nn', 'island/appos', 'pai/nn', 'color/nn', 'associ/partmod', 'closest/amod', 'gener/dep',
'irish/nn', 'school/conj', 'known/dep', 'done/rcmod', 'best/amod', 'scottish/amod', 'squar/appos', 'lectur/conj',
'befor/prep', 'nichola/nn', 'take/rcmod', 'better/amod', 'carolina/nn', 'high/amod', 'child/nn', 'resid/nn',
'enough/advmod', 'four/conj', 'privat/nn', 'unabl/amod', 'foreign/amod', 'kind/nn', 'follow/amod', 'land/nn',
'rude/amod', 'french/nn', 'friendship/conj', 'secular/amod', 'accompani/partmod', 'admir/amod', 'psychiatr/amod',
'men/conj', 'independ/conj', 'field/nn', 'howev/advmod', 'multipl/amod', 'worcest/nn', 'practic/conj',
'higher/amod', 'intellectu/amod', 'roman/nn', 'made/rcmod', 'propos/amod', 'third/amod', 'london/appos',
'real/amod', 'winchest/nn', 'art/nn', 'nice/amod', 'former/amod', 'grant/partmod', 'see/nn', 'cater/rcmod',
'corp/nn', 'sacr/nn', 'public/nn', 'liverpool/appos', 'sector/conj', 'side/nn', 'somewher/advmod', 'nation/nn',
'guild/conj', 'station/conj', 'orphanag/conj', 'del/nn', 'good/conj', 'miller/nn', 'institut/appos', 'foundat/conj',
'individu/amod', 'try/partmod', 'p/appos', 'wa/ccomp', 'decemb/dep', 'fledg/dep', 'hand/dep', 'hospic/conj',
'get/rcmod', 'greater/amod', 'ii/appos', 'per/prep', 'quiet/amod', 'vienna/nn', 'built/partmod', 'project/amod',
'bear/rcmod', 'util/partmod', 'receiv/rcmod', 'support/nn', 'copi/rcmod', 'good/nn', 'earliest/amod', 'onc/advmod',
'navi/nn', 'survei/amod', 'brighton/nn', 'order/conj', 'wa/advcl', 'carri/rcmod', 'lower/dep', 'poor/conj',
'villag/dep', 'ever/advmod', 'went/rcmod', 'overcrowd/amod', 'joseph/nn', 'give/rcmod', 'femal/amod', 'fall/nn',
'meet/rcmod', 'number/conj', 'unit/nn', 'walter/nn', 'graduat/nn', 'king/nn', 'longer/advmod', 'june/appos',
'long/nn', 'cross/dep', 'easi/amod', 'number/nn', 'dai/amod', 'line/nn', 'washington/appos', 'triniti/nn',
'albani/nn', 'sloan/nn', 'abov/prep', 'japanes/amod', 'central/amod', 'studi/dep', 'need/amod', 'municip/nn',

'dougla/nn', 'germantown/nn', 'committe/conj', 'makeshift/amod', 'blind/conj', 'expect/partmod', 'belong/partmod',
'coat/nn', 'york/conj', 'five/num', 'aliv/amod', 'italian/nn', 'person/amod', 'specialist/nn', 'alon/advmod',
'forti/num', 'spring/nn', 'notabl/advmod', 'place/appos', 'care/rcmod', 'goe/rcmod', 'prospect/nn', 'child/appos',
'negro/dep', 'quaint/amod', 'refer/partmod', 'profession/conj', 'increas/amod', 'eight/num', 'italian/amod',
'veri/advmod', 'dedic/partmod', 'todai/dep', 'old/nn', 'divis/nn', 'perform/partmod', 'jail/conj', 'bank/conj',
'harlem/nn', 'de/nn', 'heal/nn', 'two/num', 'addit/amod', 'centuri/dep', 'yet/advmod', 'nearer/amod', 'edward/nn',
'manhattan/nn', 'sector/nn', 'attach/amod', 'brief/amod', 'accredit/partmod', 'virginia/appos', 'observ/conj',
'born/ccomp', 'post/appos', 'frankfurt/nn', 'castl/nn', 'anoth/conj', 'georgetown/nn', 'rather/advmod', 'state/nn',
'usual/amod', 'eleg/amod', 'northern/nn', 'night/dep', 'street/conj', 'visit/partmod', 'prison/appos', 'evangel/nn',
'erect/rcmod', 'averag/partmod', 'factori/conj', 'found/amod', 'fair/amod', 'gener/conj', 'train/nn', 'marin/nn',
'relat/conj', 'throughout/prep', 'formal/amod', 'chines/amod', 'cold/amod', 'norwegian/amod', 'treat/partmod',
'european/amod', 'swiss/amod', 'ag/appos', 'octob/appos', 'branch/nn', 'haven/nn', 'southern/nn', 'cairo/nn',
'perman/amod', 'hall/appos', 'flourish/rcmod', 'militari/amod', 'professor/conj', 'lai/rcmod', 'continu/conj',
'librari/appos', 'protest/amod', 'juli/appos', 'neighbour/amod', 'becom/rcmod', 'monica/nn', 'therein/advmod',
'lower/nn', 'pennsylvania/nn', 'studi/rcmod', 'home/appos', 'percent/conj', 'government/amod', 'rough/amod',
'chariti/nn', 'fifti/num', 'veterinari/nn', 'facil/conj', 'south/amod', 'south/nn', 'finest/amod', 'univers/amod',
'california/nn', 'dresden/nn', 'aggress/amod', 'island/nn', 'repres/rcmod', 'ill/conj', 'place/conj',
'princip/amod', 'within/dep', 'barn/nn', 'rural/amod', 'commerci/nn', 'costli/amod', 'cater/partmod',
'particip/partmod', 'centr/appos', 'avail/amod', 'class/dep', 'miner/nn', 'got/rcmod', 'cherri/nn', 'santa/nn',
'commun/nn', 'even/advmod', 'famou/amod', 'first/nn', 'regiment/amod', 'accredit/amod', 'culver/nn',
'erect/partmod', 'introduct/nn', 'composit/amod', 'brilliant/amod', 'either/preconj', 'station/nn', 'chicago/nn',
'new/dep', 'contribut/rcmod', 'effici/amod', 'japanes/nn', 'marin/amod', 'three/num', 'spent/rcmod',
'known/partmod', 'portsmouth/nn', 'research/nn', 'kept/rcmod', 'huntington/nn', 'mai/dep', 'omaha/nn', 'school/nn',
'govern/conj', 'colorado/nn', 'percent/appos', 'attend/conj', 'field/conj', 'first/advmod', 'vancouv/nn',
'special/nn', 'countri/conj', 'onli/advmod', 'smith/nn', 'counti/nn', 'normal/conj', 'establish/infmod',
'regist/amod', 'hadassah/nn', 'system/conj', 'arabian/amod', 'countless/amod', 'base/conj', 'liverpool/nn',
'san/nn', 'brooklyn/appos', 'dignifi/amod', 'know/rcmod', 'nativ/amod', 'council/appos', 'medic/conj', 'centr/conj',
'isol/amod', 'ladi/nn', 'hill/nn', 'friendli/amod', 'nurseri/conj', 'thoma/nn', 'hagu/nn', 'old/dep', 'person/conj',
'minnesota/nn', 'irish/amod', 'fort/nn', 'detroit/nn', 'mental/amod', 'wayn/nn', 'manag/rcmod', 'frank/conj',
'uk/nn', 'older/amod', 'salisburi/nn', 'turkish/amod', 'outof/dep', 'connect/partmod', 'unnecessari/amod',
'treat/rcmod', 'bloomington/nn', 'dealt/rcmod', 'help/rcmod', 'erect/amod', 'publick/nn', 'dormitori/conj',
'loui/nn', 'boston/appos', 'specialis/amod', 'eighteen/num', 'town/dep', 'ignor/rcmod', 'earli/advmod',
'therapeut/amod', 'journal/appos', 'der/nn', 'baptist/nn', 'workhous/nn', 'seen/rcmod', 'dwell/conj', 'imagin/amod',
'proud/conj', 'offer/ccomp', 'includ/partmod', 'dai/nsubj', 'parker/nn', 'base/nn', 'pai/rcmod', 'affili/amod',
'english/amod', 'worst/amod', 'continu/amod', 'dublin/appos', 'understand/rcmod', 'separ/amod', 'chines/nn',
'illinoi/nn', 'huge/amod', 'children/nn', 'fifth/nn', 'sick/amod', 'dai/conj', 'refus/rcmod', 'vol/appos',
'australian/amod', 'cross/nn', 'outsid/dep', 'suburban/amod', 'oper/amod', 'call/partmod', 'refus/ccomp',
'two/appos', 'contain/rcmod', 'quit/advmod', 'lincoln/nn', 'gener/nn', 'estim/amod', 'replac/infmod',
'shown/partmod', 'wonder/amod', 'case/appos', 'gentil/amod', 'accredit/rcmod', 'episcop/nn', 'polic/nn',
'english/nn', 'ideal/amod', 'expect/rcmod', 'kept/partmod', 'emori/nn', 'certifi/amod', 'isra/amod', 'assur/rcmod',
'citi/dep', 'visit/infmod', 'delhi/appos', 'market/conj', 'humbl/amod', 'younger/amod', 'support/rcmod',
'secondari/amod', 'camp/nn', 'road/nn', 'servic/appos', 'mission/nn', 'endow/partmod', 'classifi/rcmod',
'israel/nn', 'survei/dep', 'ar/rcmod', 'carri/partmod', 'provid/partmod', 'area/nn', 'decidedli/advmod',
'detent/nn', 'goodli/amod', 'administr/conj', 'adventist/nn', 'go/rcmod', 'spanish/amod', 'set/rcmod', 'like/prep',
'remain/amod', 'street/appos', 'two/nn', 'ontario/nn', 'examin/conj', 'seven/dep', 'present/amod', 'firm/conj',
'natur/dep', 'better/advmod', 'success/amod', 'cadet/nn', 'cumberland/nn', 'univers/appos', 'bridg/conj',
'need/dep', 'chichest/appos', 'innumer/amod', 'due/amod', 'power/conj', 'nobl/amod', 'teach/amod', 'anthoni/nn',
'manchest/nn', 'splendid/amod', 'run/rcmod', 'come/rcmod', 'corpor/amod', 'last/amod', 'develop/rcmod', 'brick/nn',
'richmond/nn', 'maintain/rcmod', 'treatment/nn', 'provid/nn', 'support/conj', 'august/appos', 'minneapoli/nn',
'without/prep', 'less/advmod', 'manag/partmod', 'dure/prep', 'complain/partmod', 'six/num', 'oper/nn', 'york/dep',
'found/infmod', 'old/amod', 'ar/conj', 'base/dep', 'establish/partmod', 'waysid/nn', 'bellevu/nn', 'post/nn',
'view/nn', 'finish/rcmod', 'hollywood/nn', 'cambridg/appos', 'coat/dep', 'washington/dep', 'cours/dep',
'charit/amod', 'empir/nn', 'modern/amod', 'chanc/conj', situat/partmod', 'spend/rcmod', 'mention/dep', 'beach/nn',
'barrack/conj', 'visit/rcmod', 'yorkshir/nn', 'treatment/conj', 'lakesid/nn', 'denomin/amod', 'care/partmod',
'illinoi/appos', 'london/conj', 'ro/appos', 'built/infmod', 'pretend/amod', 'fallen/rcmod', 'devot/partmod',
'includ/rcmod', 'tini/amod', 'regul/conj', 'tent/nn', 'soviet/amod', 'pauper/nn', 'reformatori/conj', 'began/rcmod',
'york/appos', 'depart/appos', 'street/dep', 'onli/amod', 'virtual/advmod', 'merci/nn', 'first/amod', 'accid/conj',
'well/conj', 'care/nn', 'field/dep', 'maryland/appos', 'dai/nn', 'time/dep', 'hospit/conj', 'non/dep',
'complet/amod', 'magdalen/nn', 'proper/amod', 'lutheran/nn', 'small/amod', 'tennesse/nn', 'undesir/amod',
'rang/partmod', 'roosevelt/nn', 'countri/nn', 'privat/amod', 'provinci/nn', 'fund/conj', 'chase/nn', 'march/appos',
'attain/rcmod', 'austrian/amod', 'union/nn', 'dollar/nn', 'civil/amod', 'licens/amod', 'doctor/conj',
'springfield/nn', 'around/prep', 'lectur/rcmod', 'heart/nn', 'tokyo/nn', 'bronx/nn', 'transport/conj',
'perfect/amod', 'compani/nn', 'monast/amod', 'unknown/amod', 'compar/prep', 'profess/conj', 'board/nn',
'concert/conj', 'evalu/conj', 'regular/amod', 'model/appos', 'approach/dep', 'river/nn', 'joint/amod',
'frequent/amod', 'welsh/nn', 'industri/amod', 'emploi/partmod', 'mobil/amod', 'twenti/num', 'name/advmod',
'daili/advmod', 'perform/rcmod', 'air/nn', 'upon/prep', 'us/nn', 'enough/amod', 'stanford/nn', 'compar/amod',
'around/num', 'compani/conj', 'museum/conj', 'henri/nn', 'hopkin/nn', 'heart/conj', 'capit/nn', 'massachusett/nn',
'suffolk/nn', 'progress/amod', 'sponsor/dep', 'somat/amod', 'class/nn', 'administr/nn', 'music/conj', 'manner/conj',
'mexican/amod', 'ha/rcmod', 'still/advmod', 'assist/conj', 'place/rcmod', 'access/amod', 'war/nn', 'like/dep',
'someth/conj', 'normal/amod', 'offer/partmod', 'cuban/nn', 'near/prep', 'angel/nn', 'almost/advmod', 'buffalo/nn',
'scatter/partmod', 'ford/nn', 'nonprofit/nn', 'bore/rcmod', 'home/conj', 'second/amod', 'good/amod', 'note/rcmod',
'septemb/appos', 'mount/nn', 'practic/advmod', 'box/appos', 'long/amod', 'role/conj', 'suppli/rcmod', 'church/nn',
'quarterli/conj', 'lexington/nn', 'social/conj', 'elizabeth/nn', 'queen/nn', 'auckland/nn', 'hous/partmod',
'seminari/nn', 'lawrenc/nn', 'texa/nn', 'parti/conj', 'kind/conj', 'requir/partmod', 'grant/nn', 'louisiana/nn',
'sad/amod', 'mendota/nn', 'hospit/appos', 'result/rcmod', 'organ/conj', 'given/amod', 'shore/nn', 'coloni/nn',

```
'awai/advmod', 'frequent/advmod', 'prevent/infmod', 'special/amod', 'cemeteri/conj', 'civilian/amod', 'medic/nn',
'philippin/nn', 'besid/prep', 'fairfield/nn', 'region/amod', 'morrow/dep', 'differ/amod', 'convent/conj',
'dispos/conj', 'thei/nsubj', 'ohio/nn', 'cathol/amod', 'primari/amod', 'displai/rcmod', 'among/prep', 'need/rcmod',
'fulli/advmod', 'distinguish/rcmod', 'much/amod', 'remain/rcmod', 'baltimor/appos', 'month/dep', 'prior/amod',
'littl/amod', 'baltimor/nn', 'paper/conj', 'magazin/conj', 'england/nn', 'proprietari/amod', 'arab/amod', 'west/nn',
'approxim/num', 'level/nn', 'outsid/prep', 'centuri/nn', 'gener/advmod', 'life/conj', 'fail/rcmod', 'oper/conj',
'congress/nn', 'mean/conj', 'excel/amod', 'beauti/amod', 'paid/rcmod', 'attend/nn', 'osteopath/amod', 'jose/nn',
'michael/nn', 'pratt/nn', 'african/amod', 'set/partmod', 'grant/rcmod', 'busi/amod', 'okayama/nn', 'met/rcmod',
'orderli/amod', 'hope/nn', 'vii/nn', 'monasteri/conj', 'realli/advmod', 'like/amod', 'year/dep', 'garden/conj',
'especi/advmod', 'big/amod', 'affect/partmod', 'intellig/dep', 'toronto/nn', 'fit/partmod', 'professor/appos',
'medic/dep', 'accept/rcmod', 'dai/num', 'receiv/partmod', 'lo/nn', 'two/dep', 'virtual/amod', 'provid/conj',
'gener/appos', 'faith/conj', 'veri/amod', 'rate/conj', 'take/partmod', 'midland/nn', 'montreal/nn', 'bristol/nn',
'situat/conj', 'show/rcmod', 'vincent/nn', 'serv/rcmod', 'industri/conj', 'regina/nn', 'c/nn', 'distribut/partmod',
'depart/conj', 'admit/rcmod', 'christ/nn', 'affili/partmod', 'crowd/amod', 'peopl/conj', 'academ/amod', 'imperi/nn',
'uppsala/nn', 'chicago/appos', 'fertil/amod', 'show/partmod', 'found/rcmod', 'sheffield/appos', 'decent/amod',
'francisco/appos', 'shop/conj', 'melbourn/nn', 'given/rcmod', 'alreadi/advmod', 'prison/nn', 'master/dep',
'infrastructur/conj', 'extra/dep', 'fellow/dep', 'sentiment/amod', 'citi/conj', 'societi/conj', 'serv/infmod',
'mission/conj', 'connect/rcmod', 'dozen/num', 'motiv/dep', 'base/partmod', 'function/partmod', 'abl/amod',
'governor/nn', 'therapi/conj', 'protect/conj', 'german/amod', 'riversid/nn', 'press/conj', 'expens/conj', 'el/nn',
'german/appos', 'close/advmod', 'clinic/nn', 'describ/rcmod', 'color/amod', 'insist/rcmod', 'brompton/nn',
'main/amod', 'massachusett/appos', 'ar/ccomp', 'magnific/amod', 'mani/amod', 'exuber/amod', 'receiv/infmod',
'barrack/nn', 'nineteen/num', 'physician/conj', 'pure/amod', 'equip/conj', 'cleveland/nn', 'lenox/nn', 'readi/dep',
'oper/dep', 'expos/rcmod', 'american/amod', 'origin/amod', 'denver/nn', 'academi/conj', 'themselv/dep',
'district/nn', 'household/conj', 'becam/rcmod', 'baylor/nn', 'local/conj', 'greatest/amod', 'receiv/ccomp',
'vernon/nn', 'conduct/partmod', 'royal/nn', 'violat/conj', 'canadian/amod', 'receiv/amod', 'exist/rcmod',
'dead/amod', 'walton/nn', 'secur/nn', 'chester/nn', 'half/predet', 'squar/nn', 'damn/amod', 'suitabl/amod',
'given/partmod', 'neutral/amod', 'ordinari/amod', 'april/appos', 'open/conj', 'larg/amod', 'longer/amod',
'gymnasium/conj', 'philadelphia/nn', 'staff/conj', 'therefor/advmod', 'till/prep', 'subsid/partmod', 'rochest/nn',
'grace/nn', 'variou/amod', 'primit/amod', 'put/rcmod', 'fewer/amod', 'restaur/conj', 'communist/amod',
'suppli/conj', 'term/nn', 'park/conj', 'missionari/nn', 'fly/nn', 'municip/amod', 'morgan/nn', 'clear/nn',
'manual/appos', 'great/amod', 'sever/amod', 'farm/nn', 'center/nn', 'dc/appos', 'six/dep', 'egyptian/amod',
'newton/nn', 'rate/dep', 'forc/rcmod', 'five/dep', 'temporari/amod', 'birmingham/appos', 'nearbi/advmod',
'moscow/nn', 'deaco/nn', 'suppos/rcmod', 'almost/num', 'author/conj', 'nearbi/amod', 'local/amod', 'select/amod',
'naval/amod', 'pharmaci/conj', 'know/partmod', 'metropolitan/nn', 'sussex/nn', 'facil/appos', 'clinic/conj',
'wai/nn', 'open/rcmod', 'dublin/nn', 'liber/amod', 'care/infmod', 'colleg/conj', 'delight/rcmod', 'scienc/nn',
'adult/nn', 'monro/appos', 'state/dep', 'methuen/appos', 'ward/nn', 'jersei/nn', 'week/dep', 'dispensari/appos',
'virginia/nn', 'outli/amod', 'free/nn', 'water/nn', 'plu/cc', 'summari/amod', 'record/conj', 'night/nn',
'eleven/num', 'administ/partmod', 'albert/nn', 'choos/rcmod', 'interest/conj', 'home/nn', 'design/partmod',
'univers/nn', 'allow/rcmod', 'green/nn', 'african/nn', 'thirteen/num', 'manag/dep', 'belong/rcmod', 'wa/rcmod',
'western/amod', 'board/conj', 'citi/appos', 'oxf/conj', 'brazilian/amod', 'polit/conj', 'chapel/conj', 'term/dep',
'delhi/nn', 'control/nn', 'free/conj', 'either/advmod', 'belgian/amod', 'domest/conj', 'princ/nn', 'nurs/conj',
'magnet/nn', 'dai/appos', 'la/nn', 'refer/rcmod', 'provis/conj', 'graduat/dep', 'maintain/partmod', 'columbu/nn',
'view/conj', 'stai/dep', 'gener/amod', 'live/conj', 'includ/prep', 'method/conj', 'assist/partmod', 'dental/nn',
'enthusiast/amod', 'like/conj', 'smallest/amod', 'butler/nn', 'manila/nn', 'agreeabl/amod', 'surgic/amod',
'bureau/conj', 'ma/appos', 'simpli/advmod', 'polic/conj', 'grundi/nn', 'relationship/conj', 'columbia/nn',
'refug/conj', 'common/amod', 'metropolitan/amod', 'stori/dep', 'celebr/amod', 'less/amod', 'establish/conj',
'profit/dep', 'averag/amod', 'remain/partmod', 'foundat/nn', 'cathol/nn', 'templ/conj', 'northampton/nn',
'strong/amod', 'spaciou/amod', 'vanderbilt/nn', 'whose/rcmod', 'sumptuou/amod', 'religi/amod', 'polici/conj',
'tertiari/amod', 'indian/nn', 'st/nn', 'subsequ/amod', 'short/amod', 'black/nn', 'desir/conj', 'taken/rcmod',
'white/amod', 'public/amod', 'shown/rcmod', 'oversea/amod', 'want/conj', 'kyoto/nn', 'agenc/conj', 'rhode/nn',
'christian/nn', 'involv/partmod', 'organ/appos', 'grim/amod', 'prison/conj', 'hudson/nn', 'thousand/num',
'victorian/amod', 'templ/nn', 'littl/nn', 'food/appos', 'seventeen/num', 'major/amod', 'found/partmod',
'state/amod', 'medicin/conj', 'cost/partmod', 'greek/amod', 'sea/nn', 'oldest/amod', 'forc/nn', 'subsequ/advmod',
'often/advmod', 'refin/amod', 'hall/nn', 'elig/amod', 'post/dep', 'program/conj', 'depart/nn', 'jewish/amod',
'numer/amod', 'educ/conj', 'well/cc', 'siblei/nn', 'canal/conj', 'holmwood/nn', 'newer/amod', 'februari/appos',
'viz/appos', 'robert/nn', 'group/conj', 'appear/rcmod', 'littl/advmod', 'take/infmod', 'name/partmod',
'fourteen/num', 'bethlehem/nn', 'romant/amod', 'three/dep', 'oblig/conj', 'account/dep', 'franciscan/amod',
'glasgow/nn', 'along/prep', 'pari/nn', 'admit/partmod', 'chief/nn', 'mason/nn', 'feder/nn', 'built/rcmod',
'face/partmod', 'fill/rcmod', 'logan/nn', 'thei/nsubjpass', 'readi/amod', 'franci/nn', 'provinci/amod', 'new/nn',
'stai/rcmod', 'aid/dep', 'fine/amod', 'fill/partmod', 'jefferson/nn', 'fifteen/num', 'similar/amod', 'moral/amod',
'fund/amod', 'companionship/conj', 'establish/rcmod', 'holi/nn', 'aid/conj', 'thoroughli/advmod', 'place/dep',
'usa/dep', 'fountain/nn', 'man/dep', 'westminst/nn', 'southampton/nn', 'independ/nn', 'tradition/advmod',
'ottawa/nn', 'agn/nn', 'european/nn', 'practition/conj', 'emerg/nn', 'said/rcmod', 'languag/appos', 'margaret/nn',
'free/amod', 'nativ/nn', 'accord/prep', 'offer/rcmod', 'mennonit/nn', 'brought/rcmod', 'nearest/amod',
'run/partmod', 'asylum/appos', 'took/rcmod', 'harper/nn', 'arrang/conj'
```

We can see that the two words 15.8% of their unified features as common features.
Although school and hospital are not similar except for the fact that they are both buildings, their use in a sentence is similar and we can see that by the amount of common features they have in a sentence.
In this case the weak point of a pure syntactic similarity measurments comes into place.

False-Negative predicted noun-pairs: (Actual: True, Predicted: False):

```
Word pair: carnivore, lizard
Common feature count (intersection): 50
Uncommon feature count (symetric difference): 548

Common features: 'bird/conj', 'activ/amod', 'land/nn', 'feed/rcmod', 'live/dep', 'true/amod', 'big/amod',
'whose/rcmod', 'two/num', 'veri/amod', 'even/advmod', 'occasion/advmod', 'ar/rcmod', 'terrestri/amod',
'found/partmod', 'nocturn/amod', 'onli/amod', 'eat/rcmod', 'dwell/dep', 'small/amod', 'live/partmod', 'wild/amod',
'larg/amod', 'onli/advmod', 'mani/amod', 'live/rcmod', 'typic/amod', 'variou/amod', 'diurnal/amod', 'smaller/amod',
'eat/dep', 'mammal/conj', 'common/amod', 'male/amod', 'like/dep', 'larger/amod', 'relat/amod', 'sever/amod',
'littl/amod', 'size/dep', 'largest/amod', 'either/preconj', 'first/amod', 'known/partmod', 'great/amod',
'adult/amod', 'modern/amod', 'move/dep', 'solitari/amod', 'nearli/advmod'
```

We can see that the two words 8% of their unified features as common features.
In this case the model predicted False (non-similar) although a lizard is mostly carnivorous.
We assume that although the percentage of common features is relatively high, the amount of total features for those two words was not enough to produce a precise enough vector representation and therefore the model classified the wrong class.
Possibly, by performing an expirements on more corpus files will produce a more accurate co-occurrence vector representation that will conclude in a correct classification of this word pair.

```
Word pair: system, television
Common feature count (intersection): 366
Uncommon feature count (symetric difference): 16173

Common features: 'new/amod', 'twelv/num', 'six/num', 'ordinari/amod', 'five/num', 'indian/nn', 'line/dep',
'great/amod', 'swoop/partmod', 'fine/amod', 'unit/nn', 'larg/amod', 'much/amod', 'three/num', 'onli/preconj',
'well/cc', 'rang/dep', 'without/prep', 'four/num', 'sever/amod', 'wa/rcmod', 'air/nn', 'gun/nn', 'effici/amod',
'oper/amod', 'subson/amod', 'carri/rcmod', 'enemi/amod', 'base/dep', 'know/rcmod', 'head/partmod', 'attack/rcmod',
'take/rcmod', 'eight/num', 'seem/rcmod', 'good/amod', 'modern/amod', 'fly/dep', 'twenti/num', 'soviet/amod',
'ten/num', 'seen/rcmod', 'war/nn', 'superson/amod', 'singl/amod', 'fell/rcmod', 'like/prep', 'wing/dep',
'fly/partmod', 'reconnaiss/nn', 'seat/dep', 'hundr/num', 'tank/conj', 'brought/rcmod', 'four/dep', 'addit/amod',
'sent/partmod', 'thousand/num', 'engin/dep', 'small/amod', 'onli/amod', 'two/num', 'first/amod', 'mani/amod',
'stand/partmod', 'alli/nn', 'enemi/nn', 'long/amod', 'model/nn', 'air/dep', 'thirti/num', 'ar/rcmod', 'typic/amod',
'super/nn', 'look/rcmod', 'italian/amod', 'happen/rcmod', 'japanes/amod', 'wick/amod', 'men/conj', 'red/nn',
'even/advmod', 'whose/rcmod', 'us/rcmod', 'real/amod', 'also/advmod', 'sixteen/num', 'state/nn', 'littl/amod',
'power/amod', 'transport/conj', 'took/rcmod', 'time/nn', 'seven/num', 'armi/nn', 'fifteen/num', 'forc/nn',
'destroi/partmod', 'marvel/amod', 'russian/amod', 'thought/rcmod', 'american/amod', 'german/amod', 'capabl/amod',
'corp/nn', 'come/partmod', 'strang/amod', 'fire/nn', 'aircraft/conj', 'reliabl/amod', 'make/rcmod', 'beauti/amod',
'british/amod', 'onli/advmod', 'success/amod', 'nine/num', 'practic/amod', 'french/nn', 'jet/nn', 'five/dep',
'fastest/amod', 'two/dep', 'dai/dep', 'class/dep', 'near/prep', 'last/amod', 'hand/dep'
```

We can see that the two words 2% of their unified features as common features.
In this case we assume that the classification given in the Golden-Standard is not accurate.
Semantically, television and system usually do not appear in the same context in a sentence and we can see that by the low percentage of common features those two words appeared with in the corpus.

```
Word pair: aeroplane, fighter
Common feature count (intersection): 127
Uncommon feature count (symetric difference): 1259

Common features: 'new/amod', 'twelv/num', 'six/num', 'ordinari/amod', 'five/num', 'indian/nn', 'line/dep',
'great/amod', 'swoop/partmod', 'fine/amod', 'unit/nn', 'larg/amod', 'much/amod', 'three/num', 'onli/preconj',
'well/cc', 'rang/dep', 'without/prep', 'four/num', 'sever/amod', 'wa/rcmod', 'air/nn', 'gun/nn', 'effici/amod',
'oper/amod', 'subson/amod', 'carri/rcmod', 'enemi/amod', 'base/dep', 'know/rcmod', 'head/partmod', 'attack/rcmod',
'take/rcmod', 'eight/num', 'seem/rcmod', 'good/amod', 'modern/amod', 'fly/dep', 'twenti/num', 'soviet/amod',
'ten/num', 'seen/rcmod', 'war/nn', 'superson/amod', 'singl/amod', 'fell/rcmod', 'like/prep', 'wing/dep',
'fly/partmod', 'reconnaiss/nn', 'seat/dep', 'hundr/num', 'tank/conj', 'brought/rcmod', 'four/dep', 'addit/amod',
'sent/partmod', 'thousand/num', 'engin/dep', 'small/amod', 'onli/amod', 'two/num', 'first/amod', 'mani/amod',
'stand/partmod', 'alli/nn', 'enemi/nn', 'long/amod', 'model/nn', 'air/dep', 'thirti/num', 'ar/rcmod', 'typic/amod',
'super/nn', 'look/rcmod', 'italian/amod', 'happen/rcmod', 'japanes/amod', 'wick/amod', 'men/conj', 'red/nn',
'even/advmod', 'whose/rcmod', 'us/rcmod', 'real/amod', 'also/advmod', 'sixteen/num', 'state/nn', 'littl/amod',
'power/amod', 'transport/conj', 'took/rcmod', 'time/nn', 'seven/num', 'armi/nn', 'fifteen/num', 'forc/nn',
'destroi/partmod', 'marvel/amod', 'russian/amod', 'thought/rcmod', 'american/amod', 'german/amod', 'capabl/amod',
'corp/nn', 'come/partmod', 'strang/amod', 'fire/nn', 'aircraft/conj', 'reliabl/amod', 'make/rcmod', 'beauti/amod',
'british/amod', 'onli/advmod', 'success/amod', 'nine/num', 'practic/amod', 'french/nn', 'jet/nn', 'five/dep',
'fastest/amod', 'two/dep', 'dai/dep', 'class/dep', 'near/prep', 'last/amod', 'hand/dep'
```

We can see that the two words 9% of their unified features as common features.
Same as in the first case of the False-Negative predictions, although some aeroplane are type of fighter aeroplanes, the model produces a wrong prediction.
We assume that although the percentage of common features is relatively high, the amount of total features for those two words was not enough to produce a precise enough vector representation and therefore the model classified the wrong class.
Possibly, by performing an expirements on more corpus files will produce a more accurate co-occurrence vector representation that will conclude in a correct classification of this word pair.

Examples And Resources

- After compiling the project - project JAR files can be found in the projects target directory.
- WEKA GUI download: WEKA GUI download page link
- Example for the `inputs.txt` text file format needed to run the project can be found in the projects GitHub resources directory.
- Example for the Golden Standard word pair classification text file can be found under the name `word-relatedness.txt` in the projects GitHub resources directory.
- An example of how the 4 co-occurrence vectors (Plain, Relative, PMI, T-Test) including all the lexemes relevant features looks like can be found in the analysis folder under the name `co_occurrence_vectors_example.txt` .
- WEKA classifier ARFF input file example can be found in the results directory of the project under the name `word_pair_similarity.arff` .