Words Embeddings with Word2Vec

The aim of this project was to generate words embeddings using Word2Vec model, and to make domain classification using such generated embeddings. What we want to do is to associate to words that are semantically related (even different morphologically), similar vectors, the embeddings. I built the embeddings with machine learning and I used them for the document classification task. The description of my work is reported below.

• Part 1: Word2Vec

Among the proposed approaches I chose to implement *SkipGram*. Given a word as input to the neural network, I trained the network so to predict the context of that word. The effect of such training is the creation of the embeddings, which are the weights over the edges connecting input layer to the hidden layer.
The inputs of the neural network are the batches, they are dynamically generated from the whole dataset of words, taking into account the size of the window. This dataset is a list of words encoded as numbers. From the dataset, stop-words are excluded and punctuation is removed. Each encoding is stored in the vocabulary. Before creating the dataset, the documents get shuffled.
The neural network I used has a layer of size `batch_size` for the input batches. The embeddings matrix contains vectors of size `embedding_size` for each of the `vocabulary_size` words. Since I used noise-contrastive estimation loss, and since it is defined in terms of a logistic regression, I also defined the weights and biases for each word in the vocabulary. I defined an operation to look up the vectors for each of the words passed as input. I used the noise-contrastive estimation training loss to calculate the loss. To compute gradients and update the parameters I used Adagrad optimizer.
I did several tests tuning parameters, in order to get the highest accuracy possible. Table 1 shows my tests over a portion of the training dataset, in that case no epoch was used. My final choice of parameters is highlighted in Table 2, and it led to an accuracy of 42% over all the training dataset. Parameters that decisively changed the trend in my tests were: (1) the window size, I found that lower values led to a higher accuracy and a faster execution; (2) the Adagrad optimizer with learning rate 1, helped reaching the highest accuracy in less time. I found out that epochs work, meaning that training several times (but not too many) over the same dataset increases the accuracy. I plotted the loss and the accuracy of some of the most meaningful tests, look from Figure 1 to Figure 3.

• Part 2: Using your Vectors

After training my *best* model on the whole dataset, I played a lot with TensorBoard projector and I discovered some interesting properties of my embeddings, they are reported in the caption of Figures 4, 5 and 6. I used the code provided by the instructors to get the 5 most similar words of a given word, I compared the output with the similarities computed by TensorBoard and I report themhere below.
german: [*french, danish, dutch, swedish, norwegian*]; **astronaut**: [*astronauts, nasa, copilot, shuttle, spacecraft*]; **general**: [*generals, colonel, lieutenant, brigadier, officer*]; **food**: [*foods, nutrition, foodstuff, meat, drink*]; **cat**: [*cats, dog, rabbits, rabbit, pig*]; **eat**: [*consume, ate, eaten, eating, eats*]; **teach**: [*learn, teaching, taught, teaches, apply*]. They all evidently make sense.

• Part 3: Using your Vectors as Features for Domain Classification

In order to perform the domain classification task I read each file in the training dataset filtering out the words that are not it the vocabulary and those that appear less than a certain threshold in the document (I also tried to use only the first words, see Table 3). Using those words I computed the embedding of the document by calculating the mean (I also tried the minimum and the maximum, see Table 3) of their embeddings.
So for each of the files in the training dataset I associated an embedding. Given that the class of the embedding is the name of the directory, I performed a classification using a machine learning approach.
I tried to use support vector machines (exploiting Sklearn library), and neural networks (exploiting TensorFlow library) to perform logistic regression. Logistic regression with the neural network led to the best accuracy, of 84%. Table 3 shows the improvements in accuracy while trying some configurations. My neural network for multinomial logistic regression has an input layer of the size of the embedding of the document (150), the output layer has the size of the classes space (34). Training this network was quite immediate, few epochs were needed, in fact with only 50 epochs I could always reach the convergence. I trained this model with the documents in the `TRAINING` folder and I evaluated it with the documents in the `DEV` folders. Table 4 at the end of the document shows the confusion matrix of the evaluation of my best model on the development set. Figure 7 shows the recall, precision and F1-measure on the development set, for each domain.
Note that for more informations I'm totally available to give more detailed clarifications.

	Window size	Embedding size	Batch size	Negative samples	Vocabulary size	Optimizer	Starting avg loss	End avg loss	Accuracy (%)	Correct Analogies	Running Time (min)
`Test 1`	4	150	100	20	10^5	`SGD 1.0`	25.51	3.81	16.2	3166	136
`Test 2`	-	-	200	-	-	-	25.25	4.08	15.9	3107	83
`Test 3`	-	-	500	-	-	-	25.25	4.91	5.6	1089	52
`Test 4`	-	50	100 ♠	-	-	-	25.20	3.76	9.5	1860	125
`Test 5`	-	300	-	-	-	-	26.16	3.92	17	3320	163
`Test 6`	-	150 ♠	-	-	2×10^4	-	9.29	3.49	12.2	2391	120
`Test 7`	-	-	-	-	10^5 ♠	`MOM 0.1`	42.88	4.73	14.9	2904	146
`Test 8`	-	-	-	-	-	`ADA 0.3`	28.22	3.71	15.8	3089	146
`Test 9`	-	-	-	-	-	`ADA 1.0` ♠	25.96	3.78	22.5	4403	147
`Test 10`	1	-	-	-	-	-	22.32	4.15	21.6	4218	39
`Test 11`	2	150	100	20	10^5	`ADA 1.0`	23.66	3.91	**25.6**	**5013**	75
`Test 12`	3	-	-	-	-	-	24.83	3.82	24.4	4777	140
`Test 13`	2 ♠	-	-	100	-	-	31.12	5.8	19.9	3887	52

Table 1: Part 1 - The hyper-parameters variations to perform grid search for task 1. On the left part there are the names of the tests; in the center the hyper-parameters variations; on the right the results of the tests. **Note** *(1)* The symbol "-" means that the value in the current cell is equal to the value in the above cell (recursively), it is used for better readability. *(2)* The symbol ♠ indicates that for this hyper-parameter, a decision has been taken: such value has been seen to perform best and it will be used from this test on. *(3)* The size of the whole dataset for all these tests is the same and it's set to: 77×10^6. *(4)* The accuracy is expressed as a percentage relative to the total amount of analogies, the 19544 analogies in the `questions-words.txt` file.

	Epochs	Dataset size (# words)	Window size	Embedding size	Batch size	Negative samples	Vocabulary size	Optimizer	Starting avg loss	End avg loss	Accuracy (%)	Correct Analogies	Running Time (min)
`Test 14`	3	77×10^6	2	150	100	20	10^5	`ADA 1.0`	23.68	10.39	38.4	7502	232
`Test 15`	-	-	1	-	-	-	-	-	22.32	10.42	31.4	6142	120
`Test 16`	3	`WHOLE`	2	150	100	20	10^5	`ADA 1.0`	23.6	11.6	**42.1**	**8243**	280

Table 2: Part 2 - The hyper-parameters variations to perform grid search for task 1. In these other experiments I introduced the notion of epoch and I increased the number of words in the dataset, by reading the whole dataset. In almost 5 hours I could reach an accuracy of 42%, the maximum ever measured during my tests. So the best configuration during my tests was the one of Test 16, this configuration allowed me to create the embeddings for the following task.

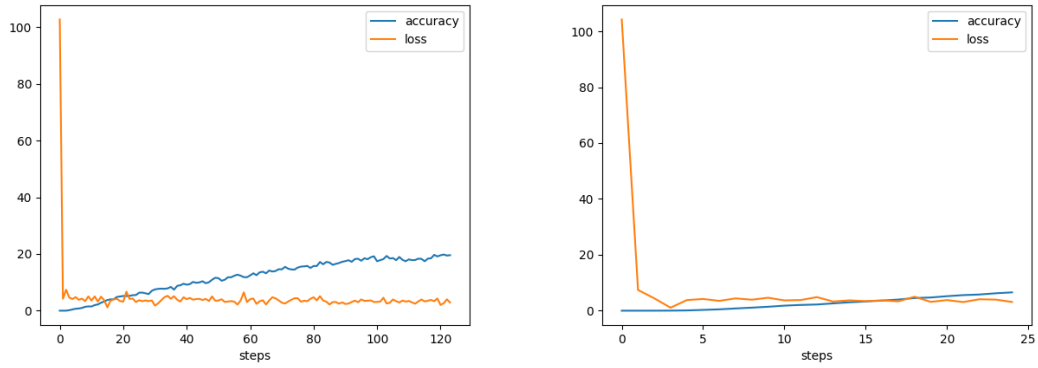

Figure 1: Test 1 on the left, Test 3 on the right. These two and the following four plots show the variation in accuracy and loss during the different tests, when the number of steps increase. For these two tests, on the left the `batch_size` is set to 100 and 500 on the right. Increasing the number of words in the batch didn't lead to better results. The curve of the accuracy in Test 3 tends always to remain lower with respect to Test 2. This trend motivated me to choose 100 as a `batch_size`.

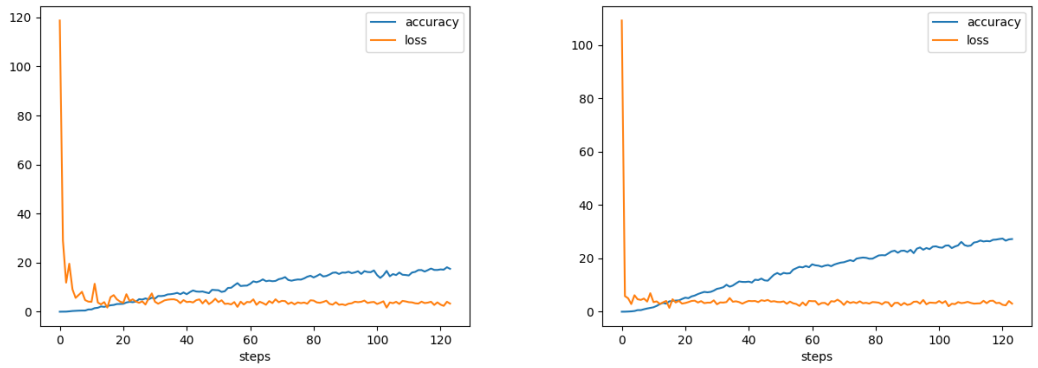

Figure 2: Test 7 on the left, Test 9 on the right. Keeping all parameters unchanged except for the optimizer changed the resulting accuracy a lot! From Test 7 with Momentum optimizer with learning rate 0.1 to Test 9 with Adagrad optimizer with learning rate 1 I gained 7.6 percentage points in accuracy. This motivated me to chose it as the definitive optimizer.

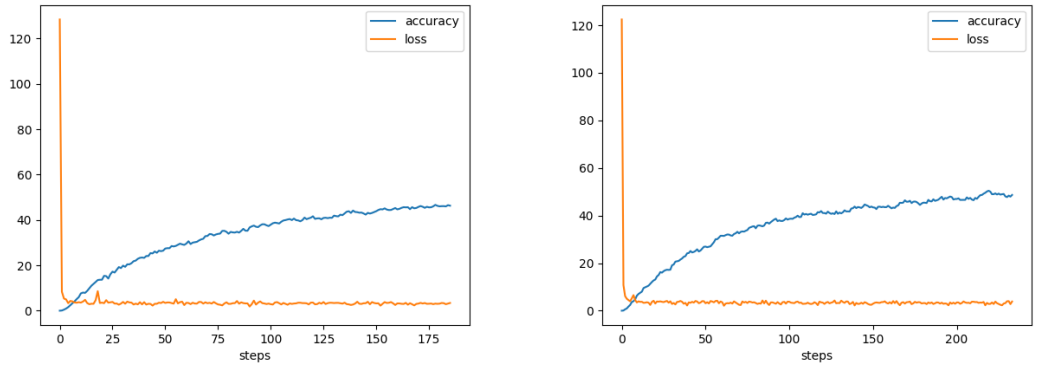

Figure 3: Test 14 on the left, Test 16 on the right. These last two plots show the accuracy and loss for the two best tests overall. It is nice to see that the curves of both tests tend to become flat with time, this because the 3 epochs tend to add less informations to the embeddings over time and the accuracy simply remains stable. Accuracy for Test 16 is higher, remarking that adding new data for training helps increasing the accuracy.

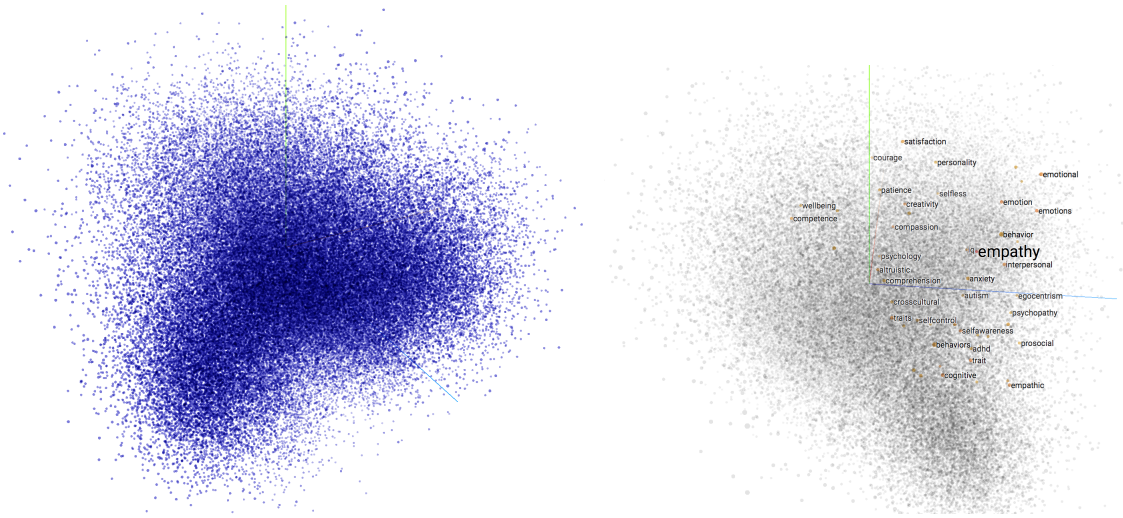

Figure 4: These are the embeddings projected in a 3D space by TensorBoard. On the left there's an unbounded projection, each of the words in vocabulary is represented by a dot in the 3D space. If one clicks on a dot, or searches for a specific word in the vocabulary, TensorBoard isolates it and its neighboring words, the most similar ones. In the right figure, I searched for the word *Empathy* and a specified number of neighboring words appear on the plot.

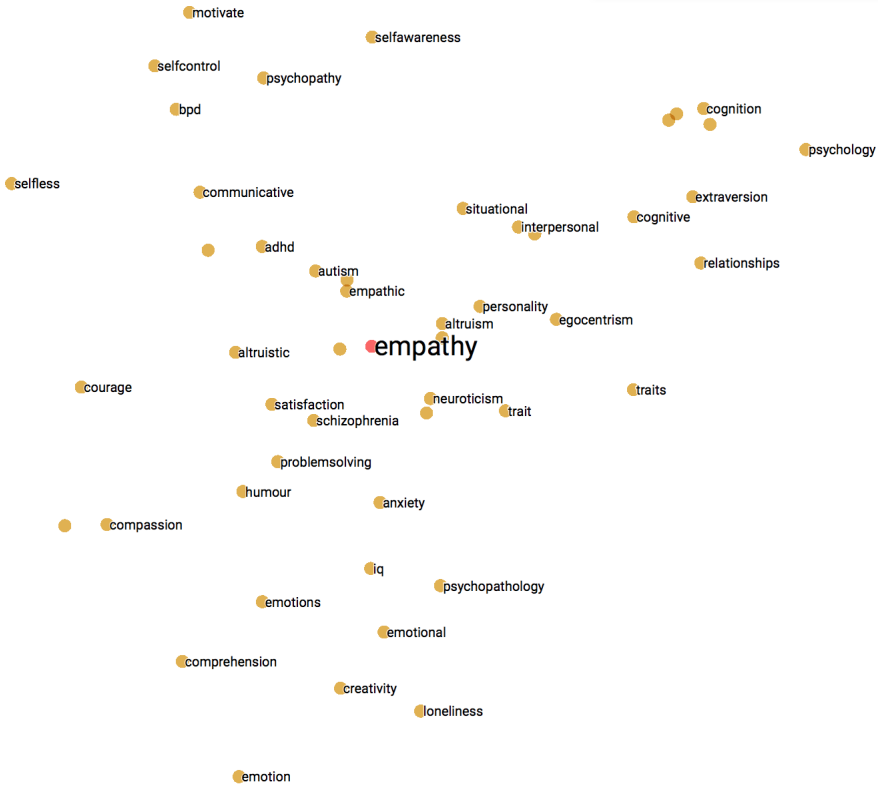

Figure 5: After searching a word in the dictionary one can chose wether to isolate the words in a 3D space ora a 2D space, as shown above for the word *Empathy*. For as I trained my embeddings the closest words to the word *Empathy* are: *emotion, cognitive, emotional, compassion, anxiety, humor, sincerity, positivity* and so on. They all make sense, they are closely related to *Empathy*.

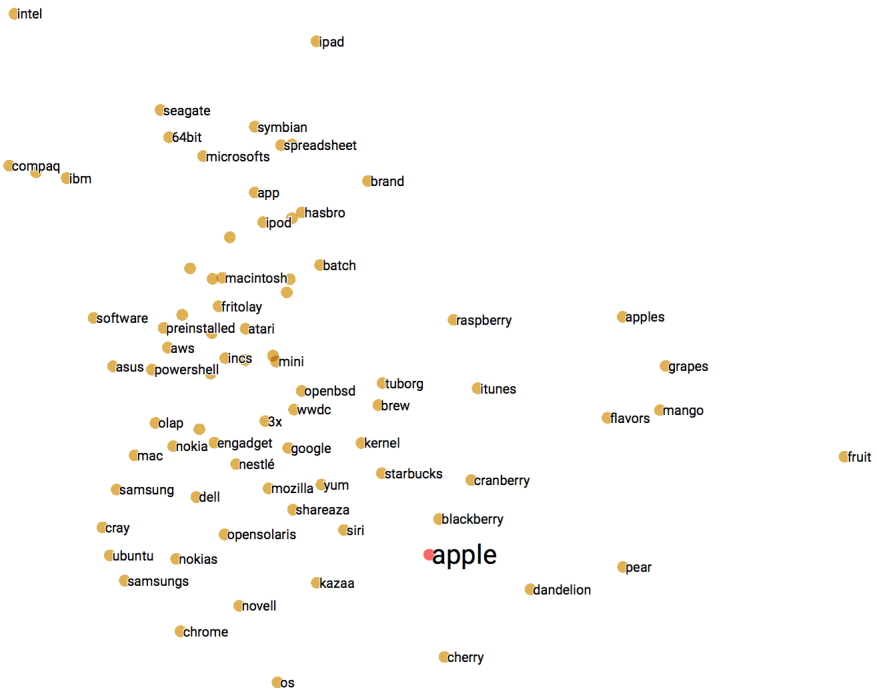

Figure 6: This is an experiment I made. I isolated the word *Apple* from my vocabulary. The closest words were very different, essentially because this word is ambiguous and it appears in very different contexts. So one could play with TensorBoard and ask to project the neighbors onto the "*tech*"-"*nature*" vector as an x axis. Note the result above, words related to the brand *Apple* are clustered on the left, words related to the fruit *apple* are clustered on the right. It was very interesting.

	First x words	First x most frequent words	First x files	Mean word embeddings	Maximum word embeddings	Minimum word embeddings	Accuracy (%)	Total tested	Correctly classified	Wrongly classified
Test 1	WHOLE	○	1000	●	○	○	81	24546	19842	4704
Test 2	50	○	-	●	○	○	77	-	18942	5604
Test 3	○	50	-	●	○	○	74	-	18252	6294
Test 4	○	20	-	●	○	○	81	-	19900	4646
Test 5	○	-	-	○	●	○	68	-	16854	7692
Test 6	○	-	-	○	○	●	71	-	17662	6884
Test 7	○	-	WHOLE	●	○	○	85	-	20768	3778

Table 3: Parameters variation to perform domain classification task. This table refers to the training with the neural network solving the logistic regression problem. The first and the second columns are mutually exclusive, they indicate how I chose the words form the document in order to create the embedding of the document. Then I show how many files I used for training, how I manipulated the words embeddings (taking the mean, max or min). And in the last part are reported the results of the evaluation over all the classes.

	ANIMALS	ART	BIOLOGY	BUSINESS	CHEMISTRY	COMPUTING	CULTURE	EDUCATION	ENGINEERING	FARMING	FOOD	GAMES	GEOGRAPHY	GEOLOGY	HEALTH	HERALDRY	HISTORY	LANGUAGE	LAW	LITERATURE	MATHEMATICS	MEDIA	METEOROLOGY	MUSIC	NUMISMATICS	PHILOSOPHY	PHYSICS	POLITICS	RELIGION	ROYALTY	SPORT	TEXTILE	TRANSPORT	WARFARE
ANIMALS	1077	4	83	3	2	0	0	0	1	2	2	0	27	7	14	1	1	2	1	1	0	3	0	0	0	1	1	0	2	1	2	1	0	2
ART	0	602	1	0	0	0	0	1	2	1	3	0	129	0	1	0	0	0	1	2	0	4	0	6	0	0	1	1	29	25	5	4	1	22
BIOLOGY	41	3	637	0	12	0	0	1	0	1	12	0	4	1	41	0	0	0	0	3	0	1	0	1	1	5	6	0	5	0	0	0	0	1
BUSINESS	1	1	0	162	0	5	0	1	1	0	3	1	9	0	3	0	0	0	3	1	3	7	0	0	2	3	2	0	1	0	3	2	1	2
CHEMISTRY	0	0	21	3	425	1	0	1	13	1	10	0	5	4	35	0	0	1	0	1	0	0	0	0	0	0	45	0	2	0	0	1	0	0
COMPUTING	0	1	1	5	1	476	0	1	3	0	1	4	0	0	1	0	0	3	1	0	2	2	0	4	0	1	1	0	2	1	2	0	1	1
CULTURE	0	0	0	0	0	0	1	0	0	0	0	0	3	0	1	0	0	0	3	1	0	0	0	0	0	2	0	4	0	0	1	0	0	0
EDUCATION	0	3	2	2	0	2	0	138	0	0	0	0	31	0	4	0	0	1	0	1	1	5	0	3	0	3	5	3	10	2	5	0	0	1
ENGINEERING	1	2	2	3	7	7	0	0	106	0	1	0	3	0	0	0	0	0	0	0	5	3	1	0	0	0	23	1	0	0	2	0	0	0
FARMING	3	0	10	7	10	1	0	0	4	29	7	0	5	1	10	0	0	0	1	0	0	0	0	1	0	0	2	2	1	0	0	0	0	1
FOOD	0	1	28	2	5	0	0	0	1	1	199	0	11	0	2	0	0	0	0	0	0	3	0	2	0	0	0	0	0	0	0	3	0	0
GAMES	0	0	0	2	0	9	0	0	0	0	1	285	0	0	0	0	0	1	0	0	0	30	0	3	0	0	2	2	1	0	16	0	0	2
GEOGRAPHY	18	62	2	0	0	0	0	3	0	5	2	0	3576	20	0	1	3	0	4	2	0	3	5	1	0	1	0	10	15	15	23	2	25	29
GEOLOGY	3	0	2	0	6	0	0	0	0	0	0	0	25	152	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	1
HEALTH	2	0	17	2	10	0	0	3	0	0	1	0	1	0	526	0	0	1	2	0	0	2	0	1	0	3	1	0	2	0	2	0	1	0
HERALDRY	0	2	0	0	0	0	0	0	0	0	0	0	24	0	1	110	0	0	1	2	0	0	1	1	1	0	0	4	0	5	4	1	2	7
HISTORY	1	3	3	0	0	0	0	1	0	0	0	0	39	4	0	0	81	5	2	2	0	0	0	0	0	0	3	9	7	38	0	0	0	34
LANGUAGE	1	2	1	1	1	7	0	6	0	0	0	0	29	0	0	1	2	290	0	3	4	5	0	3	0	1	2	2	9	1	2	0	0	3
LAW	0	0	0	6	0	2	0	1	0	1	0	0	10	0	4	0	0	2	106	2	0	5	0	1	0	0	0	5	5	0	0	0	0	5
LITERATURE	0	9	2	5	0	5	0	3	0	0	0	0	2	4	0	1	0	1	6	0	375	0	77	0	10	0	1	9	3	35	12	8	0	0	3
MATHEMATICS	0	2	2	3	8	15	0	1	5	0	0	2	3	1	2	1	1	5	0	2	421	3	0	2	1	2	52	1	3	9	6	0	0	11
MEDIA	1	5	0	1	1	3	0	0	0	0	0	10	3	0	0	0	0	0	1	9	1	2184	0	25	0	0	7	3	2	0	6	0	0	11
METEOROLOGY	0	1	1	0	0	0	0	0	0	0	0	0	65	7	1	0	0	0	0	2	0	0	22	0	1	0	11	1	1	0	0	0	1	5
MUSIC	1	6	0	1	0	2	0	1	1	0	1	1	12	0	0	0	0	1	0	4	0	38	0	1765	0	3	2	0	10	2	8	0	1	1
NUMISMATICS	0	0	0	3	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	1	0	0	48	0	0	1	1	0	0	0	0	0
PHILOSOPHY	2	6	4	7	2	4	0	1	0	0	0	0	2	1	16	0	0	5	7	10	7	4	0	0	0	164	9	7	25	4	3	1	0	3
PHYSICS	0	1	5	1	15	3	0	3	9	0	0	0	7	6	0	0	1	0	0	9	4	38	1	1	0	2	1093	0	13	7	1	0	0	3
POLITICS	0	5	1	2	0	1	0	0	0	1	0	0	41	1	1	1	2	2	14	2	0	6	0	6	0	1	1	443	8	11	6	0	1	23
RELIGION	0	25	1	0	0	0	0	1	0	0	0	2	18	1	1	0	0	2	0	8	1	6	0	1	0	9	4	6	712	18	1	1	1	4
ROYALTY	0	19	0	5	0	0	0	0	1	0	0	0	31	0	0	2	7	2	2	3	0	3	0	5	0	0	8	26	31	639	2	0	0	25
SPORT	0	0	0	3	0	0	0	1	0	0	0	3	17	0	3	0	0	0	1	1	0	4	0	1	0	0	0	1	3	1	2857	0	1	2
TEXTILE	1	4	1	1	2	2	0	0	2	0	0	0	3	0	4	0	0	0	0	0	0	5	0	0	0	0	0	0	3	0	2	88	0	1
TRANSPORT	0	7	0	4	0	1	0	1	8	0	0	0	227	0	2	0	0	0	1	0	1	2	0	1	0	0	2	3	2	2	5	0	214	12
WARFARE	1	6	0	4	0	2	0	1	11	0	1	0	143	2	0	1	8	0	2	4	0	11	0	1	0	2	8	28	9	34	4	0	13	765

Table 4: The confusion matrix. This matrix is useful to understand how things can go wrong when classifying documents only focusing on the document embedding. In fact it is possible to see that some domains are more easily wrongly classified. Some domains can be very ambiguous, for example the classification of the 841 documents of class `ART_ARCHITECTURE_AND_ARCHEOLOGY` was correct only 602 times because for 129 times it was wrongly classified as being of class `GEOGRAPHY_AND_PLACES`, which is reasonable since many documents of this class can discuss about pieces art in various geographic areas.

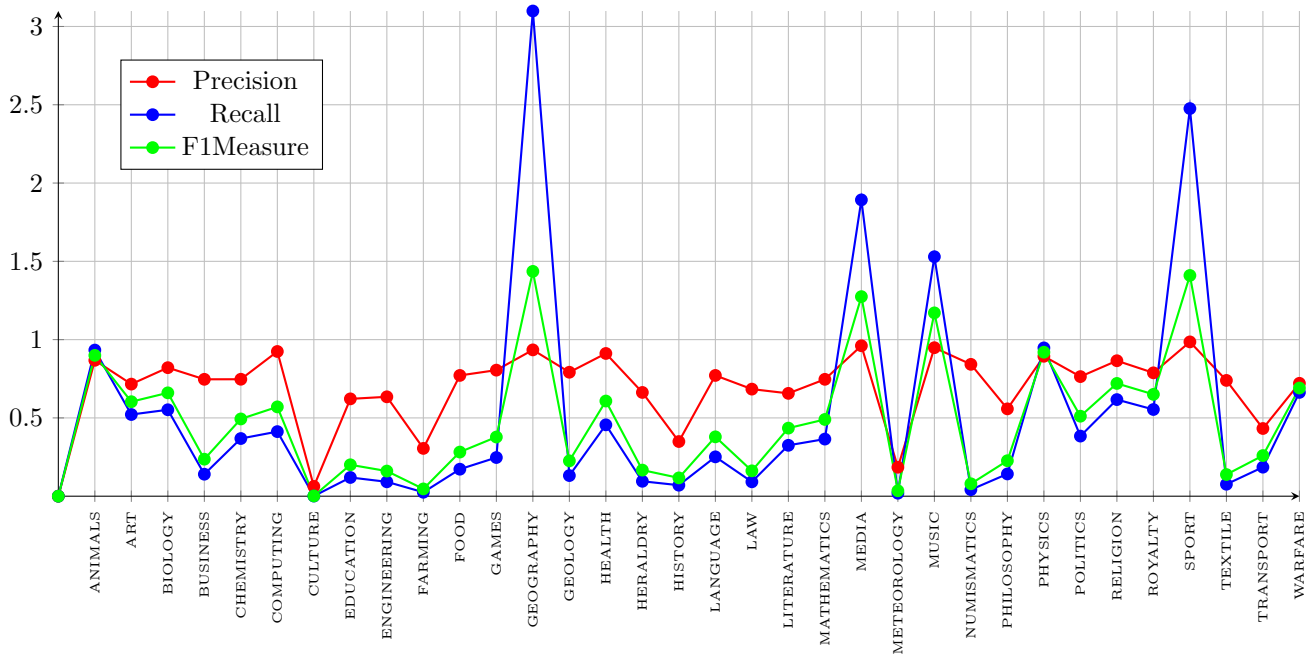

Figure 7: Precision, Recall and F1-measure The plot shows the variation in precision, recall, and F1-measure for each domain.