FINAL PROJECT

Daniel Polster 206703670
Anat Elashvili 314809195
Hadar Barak 315035865

Information Retrieve| Prof' Tsvi Kuflik

Introduction

This assignment is a collection of three tasks in information retrieval:
1. Creating a language model
2. Text classification
3. Text clustering

We'll be using methods learned in our course and the "Text Mining in Python" course to do these tasks. This will give us practical experience with different techniques. After we finish each task, we'll take a close look at the results to figure out why things turned out the way they did.

We used Python for all these tasks.

Inside the project's zip folder, you'll find the provided file with English stop words, and for each task, there's a separate Jupyter notebook file named "TASK i (1,2,3) – [Name of Task].ipynb".
For extra convenience, there's also an HTML and PDF file for each task.

Task 1 – Language Model

The relevant code is attached named "TASK 1- language modeling.ipynb" + html/ pdf

First, we need to break this sequence into individual words.
Therefore, in the first part, we were required to perform several operations to clean and polish the words in the text – like remove stop words, case folding, etc. Then calculate a language model for the text before and after cleaning.

We got a corpus containing all 57 documents under the topic "Social bias"- that is our "Train and Test" directory.

1. TOKENAZITION

First, we had to perform Tokenization.

To do this we created a function: tokenize_text(directory_path, separators)

We read the texts from the collection "Train and Test" using encoding of 'latin-1',
preform tokenization, and keep the tokens in a new file.

Note, the encoding 'latin-1' caused a few issues, which will be discussed at section 5.

In this operation, we defined the separators manually using an array:
separators = r'[;:!*$#%()&=.,\'\/\"?\s-]'

We did it manually as it said in the instructions. Moreover, space character such as "\n or \s or ' ' " is not enough. There are many separators available, if we used only space separator, we would maybe receive "? , …." As a single character, which does not help us analyze the directory/ texts (we learned that is the course "Text mining" as well).

2. CREATE A LANGUAGE MODEL

Second, we calculated a **language model** for the directory (before filtering).

We calculated the language model before any filtering, such as removing words that are usually more common than the relevant words that describe the texts. Also, we haven't removed stop words yet.

Therefore, we will expect to see characters and stop words that will be the common words in the text – especially stop words, as known are the most common words in texts.

We calculated how many times each token appears- words frequency in the list and divided by the total number of words in the list- –> word appearance probability.

We received that:

Vocabulary size: 37,408 (unique words)

Total words count: 495,900.
This represents the total count of all words in the corpus, including repeated occurrences. Each word is counted based on its frequency in the text.

The table of the 20 most common words that have the highest probability of appearing in the text:

	Word	Frequency	Probability
1	the	22309	0.045078
2	of	18014	0.036399
3	and	14165	0.028622
4	to	10095	0.020398
5	in	9495	0.019186
6	a	7105	0.014356
7	that	4960	0.010022
8	for	4456	0.009004
9	is	4408	0.008907
10	on	3094	0.006252
11	as	3081	0.006226
12	The	2866	0.005791
13	with	2840	0.005739
14	are	2542	0.005136
15	bias	2518	0.005088
16	or	2375	0.004799
17	social	2373	0.004795
18	by	2331	0.004710
19	be	2327	0.004702
20	were	2215	0.004476

Indeed, as we expected, we got those words like: the, of, and, to, a – stop words and conjunctions, are the most common in the texts.
We can also see words like bias – at 14th place and social – at 16th place, those can be a hint to the directory's subject, but it is not enough.
Therefore, almost nothing can be deduced from this about the collection of texts at this stage.

Therefore, we will perform cleaning operations on the text:

3. LINGUISTIC ACTIONS

a. Stop words removal

First, we read the stop words from the text file that was given to us "stop_words_english.txt". we converted all the words to lower case for better comparison later.

Then, with a loop, we went through all the tokens in the tokens array and removed the stop words (all the words and tokens in lower case to assure best comparison).

We received that:

Vocabulary size: 36,191

Total word count after removal: 263,856.
This represents the total count of all words in the corpus, including repeated occurrences after the removal of stop words. Each word is counted based on its frequency in the text.

We can see that the vocabulary size decreases from 37,408 to 36,191 – a decrease of 1,217 words.
As in a Total decrease of 231,044 appearances of stop words.
Therefore, we can see that stop words are a major part of the texts, but at this phase, nothing new can be deduced from them about the collection. (because we didn't do LM yet).

b. Case folding

We performed case folding, i.e. removing capital letters – by using .lower() method, and calculated the language model now.

We received that:

Vocabulary size: 31,134

Total word count after lowercase: 263,856
This represents the total count of all words in the corpus, including repeated occurrences. Each word is counted based on its frequency in the text.

We can see that the vocabulary size decreases from 36,191 to 31,134 – decrease of 5,057 words.
Therefore, we can see that upper case is not a major part of the texts, but still appears,
maybe part of them also appeared in lowercase (sort of duplicates), now we can have a better look at the LM we will calc in 4.

c. Stemming

If so, we will enter it into our word database, performing stemming using the Porter library. We did this for every word in every document database.

We used the Porter Stemmer library to perform the stemming: stemming = PorterStemmer().

We received that:

Vocabulary size: 24,045

Total words count after lowercase: 263,856
This is logical to us, because in stemming we don't change the total number of words, we just shorten their length – so the length of each word may change, but all the words remain in the corpus.

We can see that the vocabulary size decreases from 31,134 to 24,045 – a decrease of 7,089 words.

4. CREATE A NEW LANGUAGE MODEL for your collection after every step in 3

We created a language model for the directory after every Linguistic action, with the same operations as in section 2: We calculated how many times each token appears- word frequency in the list and divided by the total number of words in the list → word appearance probability.

After removing stop words

The table of the 20 most common words that have the highest probability of appearing in, the text:

	Word	Frequency	Probability
1	bias	2518	0.009543
2	social	2373	0.008994
3	al	1174	0.004449
4	â□□	1089	0.004127
5	study	813	0.003081
6	group	769	0.002914
7	data	677	0.002566
8	Journal	655	0.002482
9	gender	647	0.002452
10	Social	634	0.002403
11	review	575	0.002179
12	studies	570	0.002160
13	biases	556	0.002107
14	language	549	0.002081
15	participants	546	0.002069
16	â□□	514	0.001948
17	people	513	0.001944
18	groups	502	0.001903
19	model	492	0.001865
20	desirability	484	0.001834

Indeed, as we expected, there are no longer stop words, so we have more meaningful words that are in the top 20 most common, words like bias, social, model,…

We got that the word "bias" went from 14th place to 1st place, and "social" from 16th place to 2nd.

In the 3rd and 15th places we see: $\hat{a}$ [ ][ ] those are unfamiliar words to us (and to Python), which probably stems from the fact we used the encoding 'latin-1' in the reading of the texts from the collection.
The 3rd place is very high, and it can damage the analysis of the texts.

At this stage, more can be deduced from this about the collection of texts.

After case folding

The table of the 20 most common words that have the highest probability of appearing in, the text:

We can see that the words: "social" and "bias" keep the 1st and 2nd place, their frequencies have increased, which indicates they appeared both in lower and upper case. And both probabilities

	Word	Frequency	Probability
1	bias	3091	0.011715
2	social	3087	0.011700
3	al	1203	0.004559
4	â□□	1089	0.004127
5	study	1051	0.003983
6	gender	834	0.003161
7	group	820	0.003108
8	data	793	0.003005
9	journal	774	0.002933
10	language	771	0.002922
11	review	739	0.002801
12	studies	689	0.002611
13	participants	653	0.002475
14	biases	622	0.002357
15	desirability	599	0.002270
16	model	564	0.002138
17	people	548	0.002077
18	effects	526	0.001994
19	â□□	514	0.001948
20	behavior	513	0.001944

increase, for "bias"- the probability increases from 0.009543 to 0.011715 – a relatively large increase, for "social"- from 0.008994 to 0.011700.
we can see as well that the probabilities of "social" and "bias" are almost identical, which can hint that they show up together, maybe a common phrase- bigram.

In the 3rd and 4th places, we have the words "al" and " $\hat{a}$ [ ][ ] ", which from we can't learn much for now. The word "$\hat{a}$ [ ][ ]" went from 3rd place to 4th.

Moreover, we can see that the word "journal" at 9th place, appeared before at 8th place with the capital "J" and its frequency increased from 655 to 744, as well as its probability- from 0.002452 to

0.002933, which indicates that the word appeared both in upper case and lower case before the lower-case operation.
From that we learn that lowercase can help us decrease "duplicates" of words, to get more information about common words in the directory.

After stemming

The table of the 20 most common words that have the highest probability of appearing in, the text:

	Word	Frequency	Probability
1	social	3462	0.013121
2	bia	3092	0.011719
3	studi	1814	0.006875
4	group	1339	0.005075
5	review	1263	0.004787
6	al	1215	0.000000
7	model	1107	0.004195
8	â□□	1089	0.004127
9	measur	1085	0.004112
10	report	1033	0.003915
11	desir	911	0.003453
12	particip	907	0.003437
13	gender	891	0.003377
14	journal	884	0.003350
15	bias	874	0.003312
16	question	851	0.003225
17	test	850	0.003221
18	behavior	843	0.003195
19	differ	816	0.003093
20	languag	802	0.003040

We can see that the word: "social" keeps the 1st place, its frequency increased by 1, from 3091 to 3092, so its probability remains almost the same.

The word "bia" in the 2nd place, stemmed from "bias", with a minor increase in the probability.
we can see as well that the probability of "social" is greater than of "bias".

At 8th place, the word: " $\hat{a}$ [ ][ ] ", went down from 4th place, the fact that this word still appears is unexpected.

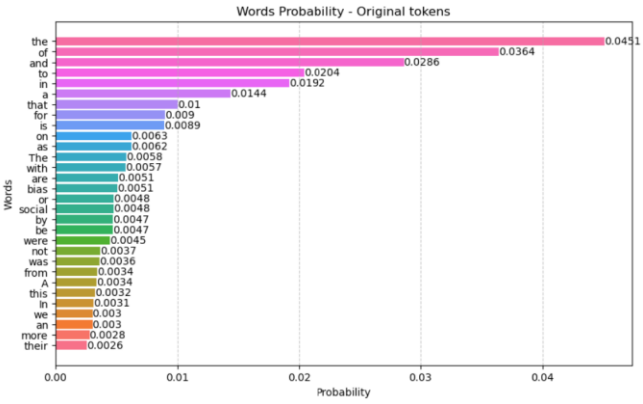
Moreover, we can see that word "review" went up from 11th place to 5th.

5. COMPARE AND DISCUSS THE DIFFERENCES

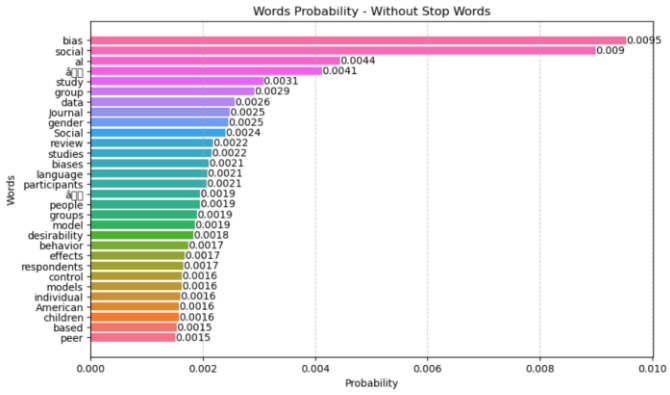
We will compare the results we received, starting from the initial model (before the cleanings) to the last model (in which we performed all the required cleanings).

From task 4 where we calculated a total of four language models at each step we performed, we plotted bar plot for each step- after each Linguistic action:

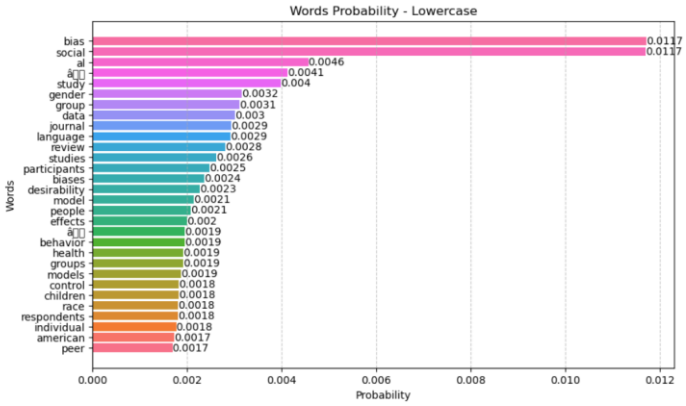
1. Initial



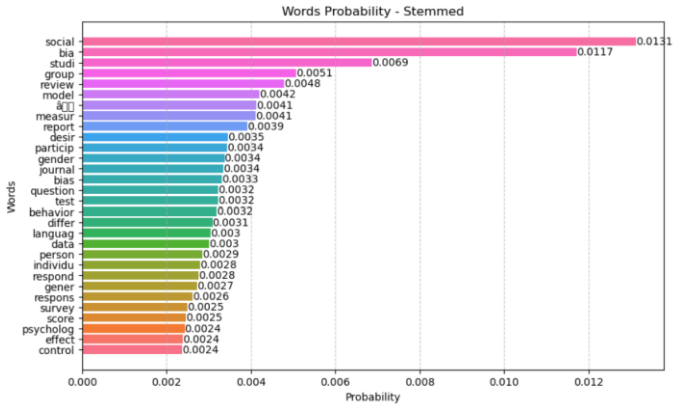
2. After stop word removal



3. After lower case



4. After stemming



(The results of the analysis in the graphs are identical to those in the table at every stage. The graphs display 30 words as opposed to 20 in the tables)

First, we will note that the number of different words in our dictionary is smaller from model to model. We started with an initial number of words of 37, 408 and at the end of the process we got 24,045 different words in our dictionary.

We will note that after removing stop words, the number of different words in our dictionary decreased by 1,217 words. We reduced the size of the dictionary by about 4%.

After that, we performed case folding on the words. We replaced uppercase letters with lowercase, and we got that the size of our dictionary is now 31,134, which means we reduced the size of the original dictionary by an additional 14% .

After that, we performed stemming for each word. Therefore, we received that there are now 24,045 different words in the dictionary. Therefore, we significantly reduced the size of the dictionary after performing this filtering.

overall reducing of about 35% the size of the dictionary.

In addition, it can be seen that in the language model after all the cleanings, we got that now the two most common words in the texts are our directory's name: social, bia(s).

We did expect to get this result, after all during the directory contains texts related to social bias.

In addition, we will note that other common words in the texts are words such as: gender, report, journal, and more…

These are words that relate to the chosen topic and indeed we would expect to see them in the texts.

Note, the word/ token: : $\hat{a}$ [ ][ ] appears in high probability even after all the Linguistic actions, we did not expect to see word "like that", we assume that this is due to use of encoding= 'latin-1' while reading the texts from the directory.
The fact that this word is placed high probably indicates that it is a meaningful word that can greatly help us in analyzing the understanding of the subject of the directory, and the fact that we do not "manage" to see what it is may have a negative effect on the analysis.

Note, that we tried using encoding = 'utf-8', to see if we could get a different result but without success, we also tried to filter out non-English words, but since the result stayed the same, we removed that part from our code. This strengthens our hypothesis that it happened due to the encoding feature.

Moreover, before cleaning the text we did expect to see stop words, which at this stage we have not removed yet, as the most common occurrences in the texts. Not like punctuation marks, which we removed in the tokenization phase.
However, these are words that do not describe texts on specific topics and therefore they did not help to understand the subject of the texts. In the language model after the cleanings, we got results from which we can learn about our text database in a better way.

After performing the stemming, words like Bias became bia, a word that is not defined in the English language and we got it because of cutting its suffix by the stemming operation. In other words, we must understand that these operations can be problematic in some cases, because we may lose the meaning of words in the text.

CONCLUSION

We must understand that there is a "Trade-off" in every cleaning operation. We reduce the size of the dictionary as well as the size of the posting, but we may lose information and the meaning of the words in the texts.

Task 2 – Text Classification

In this part, we were required to take an additional directory- "Negative examples" to perform text classification algorithms.
We will see the results we got and discuss them.

In order to perform these actions, we used Python with sklearn - machine learning library, and from there we imported the needed methods.

1. DOWNLOADED THE DIRECTORIES

Use the "Train and Test" and the "Negative examples" directories.

2. LABEL THE DOCUMENTS

The texts are labeled into 2 categories:

- Documents that belong to our category- these are the texts in the "Train and Test" directory.
- Documents that do not belong to our category- these are the texts in the "Negative examples" directory.

We created a function that reads and labels the texts: load_data(directory, label)
which iterates on the texts and labels them: 1= for docs in "Train and Test", 0= for docs in "Negative example". We then save the positive texts (with label 1), and negative texts (with label 0). We also save the titles/ file names of each text (for better analyze later in the task)

We then ran four classification algorithms as required by the task.

3. PERFORM LINGUISTIC OPERATIONS

We first combine all the positive and negative examples.

Then**,** we perform linguistic operations by creating a function: preprocess_document(document) that does tokenization and removal of stop words at once.

Note that here we used from nltk.corpus import stop words for the removal, not as in TASK 1.

that is the course "Text mining" as well).

4. RUN THE MODELS

a. Chose four types of classifiers

The classifiers we choose to work with:

1 – 'Multinomial Naive Bayes'

2 – 'SVM'

3 – 'Rocchio'

4 – 'K Nearest Neighbors'

b. Performed 10-fold cross-validation

First, we split the data into train set – 90% and test set- 10%.
Then, we created a function that performs 10-fold cross-validation for each classifier:
perform_cross_validation(classifier_name, classifier, X_train, y_train, all_filenames).
In the function, we vectorize the texts- to term frequency vectors, we join them into one string and convert the string to an array (we do this to allow the work of the clustering operations).
Moreover, in the function, we "calculate" misclassified docs for analyze later in the task.

(that is from the course "Text mining" as well).

5. RESULTS AND DISCUSSION

Multinomial Naïve Bayes

Accuracy: 0.9078
Precision: 0.9161
Recall: 0.9062
F1-Score: 0.9063
Misclassified Documents: 9

We got 1 document that was mistakenly classified as relevant.

We got 8 relevant documents that were mistakenly classified as irrelevant.

	Title	True Label	Predicted Label
0	A Neural Marker...	1	0
1	Addressing Bias in...	1	0
2	Analyzing social biases...	1	0
3	Casuistry and Social...	1	0
4	Methods to Reduce...	1	0
5	SOCIAL DESIRABILITY BIAS...	1	0
6	Bias mitigation for...	0	1
7	Human Trust Modeling...	1	0
8	Mitigating gender bias...	1	0

SVM

Accuracy: 0.8979
Precision: 0.9012
Recall: 0.8958
F1-Score: 0.8949
Misclassified Documents: 10

We got 8 documents that were mistakenly classified as relevant.

We got 2 relevant documents that were mistakenly classified as irrelevant.

	Title	True Label	Predicted Label
0	A practical tool...	0	1
1	American J Political...	0	1
2	Negative social bias...	0	1
3	Quantifying Social Biases...	0	1
4	Social Biases in...	0	1
5	AI Fairness 360_...	0	1
6	Balanced datasets are...	0	1
7	Bias mitigation with...	0	1
8	Human Trust Modeling...	1	0
9	Individualised responsible artificial...	1	0

Rocchio

Accuracy: 0.8756
Precision: 0.8798
Recall: 0.8750
F1-Score: 0.8738
Misclassified Documents: 12

We got 9 documents that were mistakenly classified as relevant.

We got 3 relevant documents that were mistakenly classified as irrelevant.

	Title	True Label	Predicted Label
0	American J Political...	0	1
1	Applied Linguistics, Social...	0	1
2	Casuistry and Social...	1	0
3	Social Biases in...	0	1
4	SOCIAL DESIRABILITY BIAS...	1	0
5	Towards Understanding and...	0	1
6	Unlearning implicit social...	0	1
7	Balanced datasets are...	0	1
8	Human Trust Modeling...	1	0
9	Medicine and the...	0	1
10	Mitigating bias in...	0	1
11	Mitigating political bias...	0	1

KNN

Accuracy: 0.7811
Precision: 0.8169
Recall: 0.7812
F1-Score: 0.7704
Misclassified Documents: 21

We got 19 documents that were mistakenly classified as relevant.

We got 2 relevant documents that were mistakenly classified as irrelevant.

	Title	True Label	Predicted Label
0	American J Political...	0	1
1	British J of...	0	1
2	Exploring Social Desirability...	0	1
3	MODEL BIAS IN...	0	1
4	Negative social bias...	0	1
5	Quantifying Social Biases...	0	1
6	Social Bias and...	0	1
7	Social Biases in...	0	1
8	Towards Understanding and...	0	1
9	Unlearning implicit social...	0	1
10	AI Fairness 360_...	0	1
11	Anatomizing bias in...	0	1
12	Balanced datasets are...	0	1
13	Establishing Data Provenance...	0	1
14	Generative adversarial networks...	0	1
15	Human Trust Modeling...	1	0
16	Medicine and the...	0	1
17	Mitigating bias in...	0	1
18	Mitigating bias in...	0	1
19	Mitigating gender bias...	1	0
20	Modeling epistemological principles...	0	1

The result we got is that the best method is to use Multinomial Naïve Bayes, and the worst is KNN.

As we learn, KNN relies on distance calculation to decide the nearest neighbors. When we have a text with a large vocabulary, all the points become close to each other, so the meaning of nearest is smaller.
Moreover, we have a noise word in our dictionary - $\hat{a}$ [ ][ ], as we know KNN is likely to be affected. If this word is weighted similarly to more informative words, its presence can lead to incorrect nearest neighbor calculations, significantly affecting both the accuracy and the reliability of the classifications.

Multinomial Naïve Bayes is a probability based method. It is more tolerant than the other because of the independence assumption. As we can see the noise word we have, is not destroy the model as we got in KNN. Another advantage of this assumption is that This assumption allows MNB to estimate probabilities from small training sets more effectively.

Rocchio gives a worse result than Multinomial Naïve Bayes as we accepted. Although Rocchio is relatively simple among the models, it does not deal well with classifications that are non-linear.

SVM classifies data by finding an optimal line or hyperplane that maximizes the distance between each class in an N-dimensional space. It can handle both linear and nonlinear classification tasks, which can explain why it gives such a good result.

CONCLUSION

As we accepted in the previous task, the unrecognized words have an impact on the models performance. In cases like this, and when the dataset is not large, we will recommend using Multinomial Naïve Bayes or SVM.

Task 3 – Text Clustering

1. CREATE 4 CLUSTERS

Take documents from 2 additional directories: AI explainability, and Algorithmic fairness.
We created a function to read all docs from all 4 directories: 'Social Bias', 'Bias Mitigation', 'AI Explainability', and 'Algorithmic Fairness'.

While reading, the function is doing tokenization and stemming - this took a while, due to the fact we have more than 200 texts overall.

We saved the documents and labels.

2. PERFORM CLUSTERING

We performed clustering Python with sklearn- machine learning library, and from there we imported the needed methods.

a. Perform clustering using K-means and 4 clusters

First, we converted the clusters matrix to tf-idf using X =vectorizer.fit_transform(documents) where, vectorizer= TfidfVectorizer(stop_words='english'), this also removes stop words, so that we can better distinguish between the clusters.

To be able to see results, we reduced the dimension to 2D using LSA algorithm (we learned in "Text mining") using TruncatedSVD(n_components=2, random_state=0).

Then, we performed K-MEANS clustering to 4 clusters, with init= 30 – number of performing with random starting point= "roots"

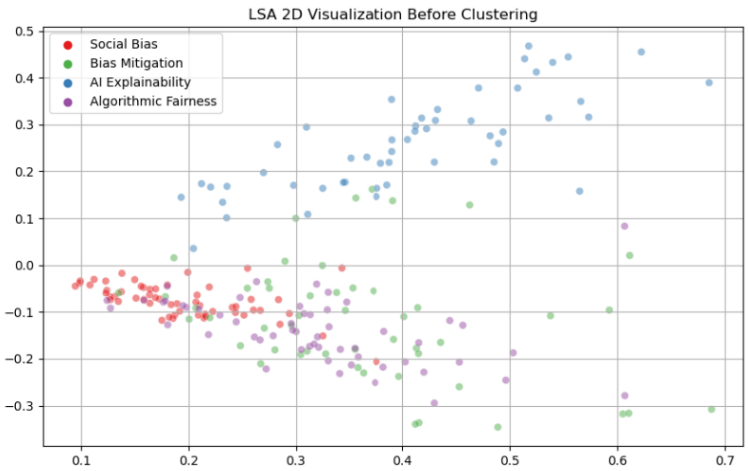
After the clustering, we used LSA algorithm again to reduce the dimension of the result, and we found centroids.

Results appear in b.

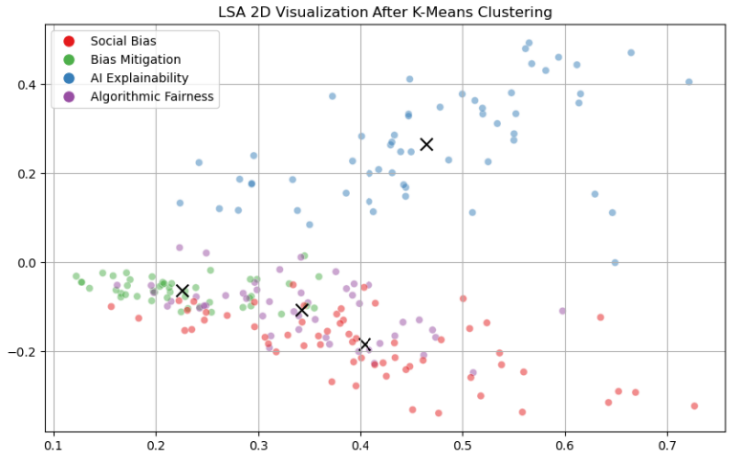
b. Analyze the results

We analyzed the results by plotting scatter plots before and after the clustering.

<u>Before clustering:</u>



After clustering:



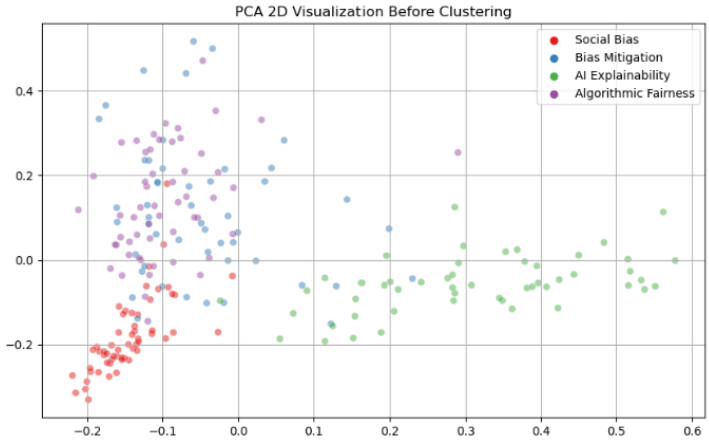
By visual viewing, the results seem to be not very good, we can see that by the colors of the clusters, there is an overlapping between the clusters. For example, we can see that besides the blue dots (AI Explainability), all the centroids are surrounded by more than one color, which means more than one cluster.

The centroids are relatively well located.

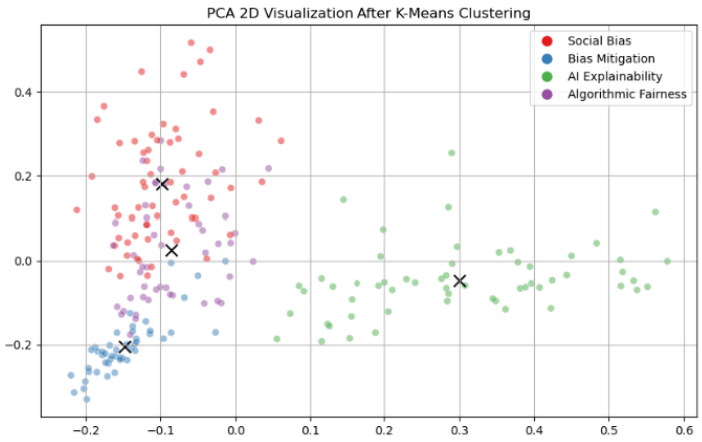
This was unexpected to us, especially due the fact we initialized init=30.

Due to this result, **we performed clustering using K-means once again**, this time using PCA algorithm for reducing the dimensions.

Before clustering:



After clustering:



By visual viewing, the results seem to better look than the LSA method, but still the results are not very good.
For example, looking at 'Social Bias' – in red color before the clustering, all the red dots are somehow close to the Y and X axis (close to 0,0 on the plot), after we can see that the red dots are farther than the X axis, more "speeded / long located from each other" but still close to the Y axis.

The centroids are relatively well located.

Again, this was unexpected to us, especially due the fact we initialized init=30.

We continued to other evaluation metrics:

Accuracy: 0.8028

That is, we received that the percentage of success is about 80%, that is - in 80% of the docs the document was classified into its correct cluster, a relatively high percentage.
In contrast, in about 20% of the docs, the algorithm was wrong.

We looked for more metrics to evaluate the results, we used scikit library documentation from:
https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation
section: 2.3.11. Clustering performance evaluation

Adjusted Rand Index: 0.5919

Adjusted Rand Index is a measure that is an improvement to the Rand Index we learned in class. Is a measure of the similarity between two data clustering. The main difference is that ARI adjusts for chance, which makes it a more reliable measure when comparing clusterings of different sizes and distributions, where a score close to 1 indicates a perfect match between the clusterings.

In our case, ARI of 0.5919 indicates on a moderate to good agreement between the cluster labels and the true labels.

Adjusted Mutual Information: 0.6035

Adjusted Mutual Information is also a measure that compares two clusterings, but it is based on the amount of shared information. It adjusts for chance much like the ARI, ensuring that a score of 0 means the clusters are independent of the true labels.

In our case, AMI of 0.6036 indicates a good agreement between the clustering output and the actual labels.

Homogeneity: 0.6087

Homogeneity checks if each cluster contains only members of a single class.
In our case, 0.6087 indicates about a good level of homogeneity, as we can see in the plot above.

Completeness: 0.6108

Completeness checks if all docs of a given class are assigned to the same cluster.
In our case, 0.6108 indicates that most data points from a single class are grouped into the same cluster.

V-measure: 0.6097

V-measure is the harmonic mean of homogeneity and completeness. It is an overall measure of how successfully the samples have been assigned to clusters.
In our case, 0.6098 indicates that overall good performance but shows that there is still room for improvement in achieving perfect clustering.

CONCLUSION

Our algorithm shows quite good performance with potential for further optimization.

As we learned, the K-means algorithm is an algorithm that belongs to the flat approach among the two clustering approaches. The algorithm cluster documents into clusters according to Euclidean (normalized) distance. In other words, the algorithm searches for a gathering regarding the centroids. The goal of the algorithm is to create clusters where the distances between the documents in the same cluster are very small and between the clusters, the distances are very large.

The process is an iterative process where each time the centroid of each cluster is updated, therefore there may be shifts of the centroids. We performed only 1 iteration in each method (LSA, PCA), but with 30 "roots"- that is equivalent to running k-means 30 iterations.
The result is the best clustering performance achieved within the 30 start points.

Since starting points are randomly selected within the algorithm and the iterative process begins, there is a certain problem with the algorithm: it is not certain that we have selected documents that will give us the most optimal result and we do not know when the algorithm will converge.

In the results, we can see in the visualizations that in each cluster all the documents are not very close to each other and the distance between the clusters is relatively small as well besides 'AI Explainability' in both LSA and PCA visualizations, has large distance from the other 3 after the clustering.

We need to be aware of the fact that the visualizations we create can be very misleading, the 'real' dimensions are huge, and presenting in 2D it doesn't reflect 100% of the results.

In conclusion, K-means is expensive to calculate, is not robust, and therefore has high complexity. When choosing documents, we will prefer to choose documents in the center and not distant documents in order to find the optimal result. However, we received an 80% success rate (Accuracy), meaning that despite its shortcomings, a relatively good job was done.