Math Lectures

Objective:

Comparing Doc2Vec and Tf-idf models to classify text.

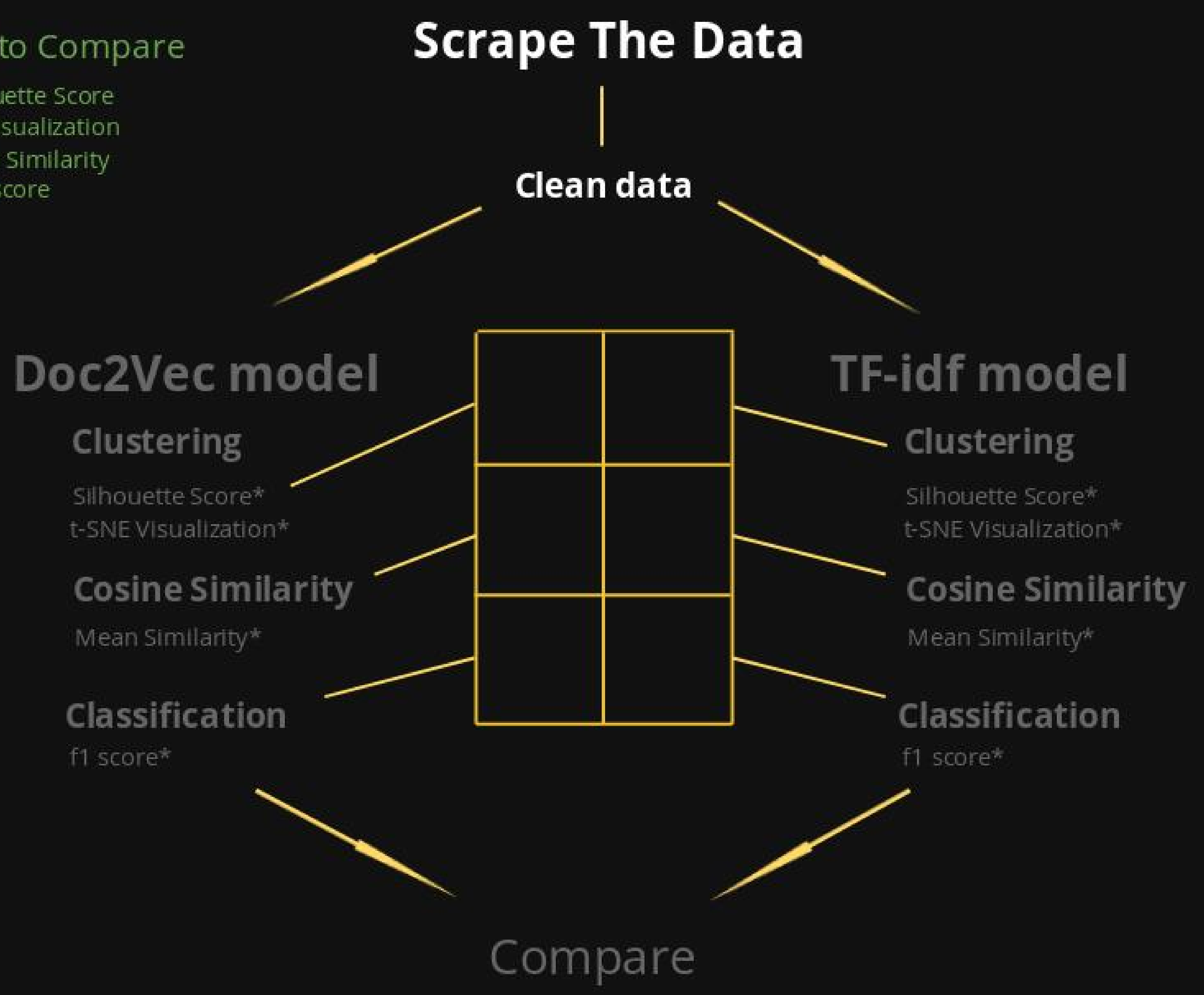
Proposed Flow

-Project Overview

- The Data
- Cleaning the data
- Doc2Vec model
- TF-idf model
- Comparison
- Application

Metrics to Compare

Silhouette Score
t-SNE Visualization
Mean Similarity
f1 score



The Data

-The data consists of subtitles from 860 maths lectures scraped

from youtube videos

Scraping the data

Get video ids

1. Find playlists on youtube
2. Youtube API call for video ids from playlists
3. Store video ids

```python
def get_vid_ids (play_lists):
    #generate data from call
    titles = []
    descriptions = []
    channelids = []
    vidids = []
    playlist_ids = []
    video_df = pd.DataFrame()
    for play_list in play_lists:
        #request playlist items
        pl_data = playlist_items_list_by_playlist_id(client,
                            part='snippet,contentDetails',
                            maxResults=50,
                            playlistId=play_list)

        #extract information about each video in the playlist

        for item in pl_data['items']:
                titles.append(item['snippet']['title'])
                descriptions.append(item['snippet']['description'])
                channelids.append(item['snippet']['channelTitle'])
                vidids.append(item['snippet']['resourceId']['videoId'])
                playlist_ids.append(item['snippet']['playlistId'])

    video_df['title'] = titles
    video_df['description'] = descriptions
    video_df['channelid'] = channelids
    video_df['videoids'] = vidids
    video_df['playlist_id'] = playlist_ids


    return video_df
```

Download CCs, extract information

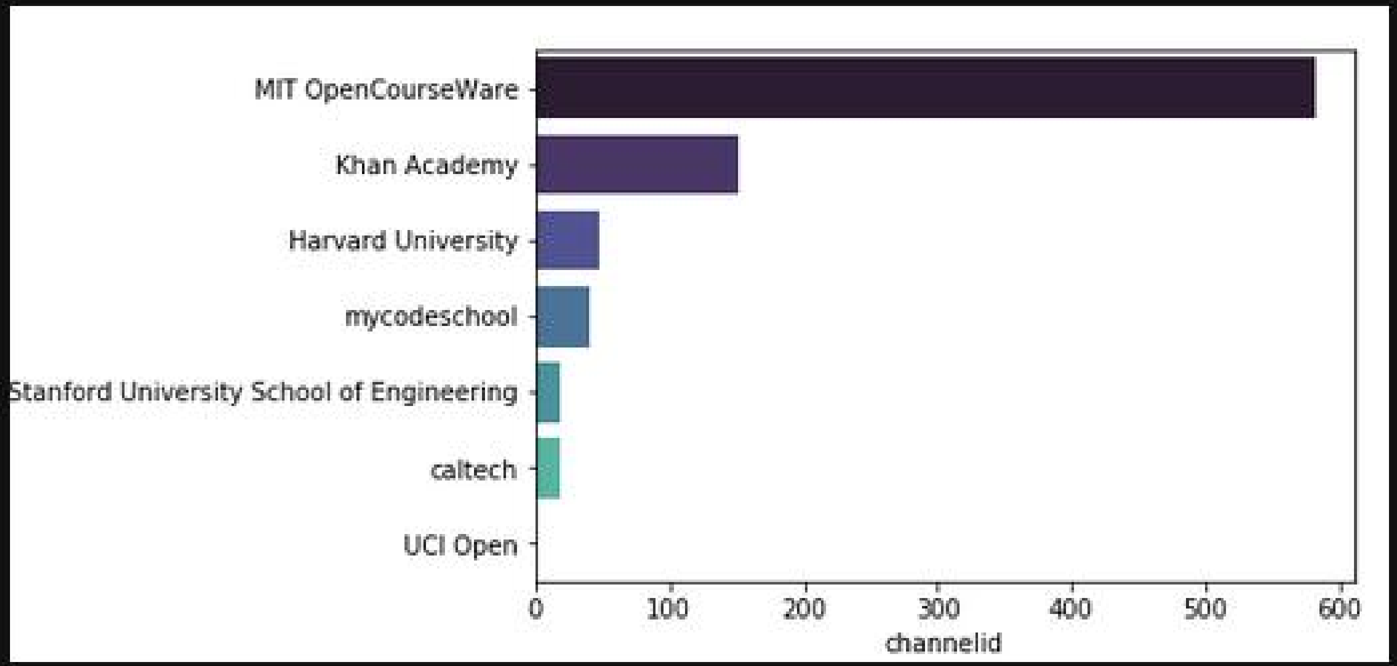
4. Use video id list with youtube -dl to download the subtitles
5. Convert files to csv
6. Store text from all csv files in one DataFrame

```python
def get_all_ccs(vids):
    base_url = 'https://www.youtube.com/watch?v='
    lang="en"
    for vid in vids:
        url = base_url + vid
        cmd = ["youtube-dl","--skip-download","--write-sub",
              "--sub-lang",lang,url]
        os.system(" ".join(cmd))
```

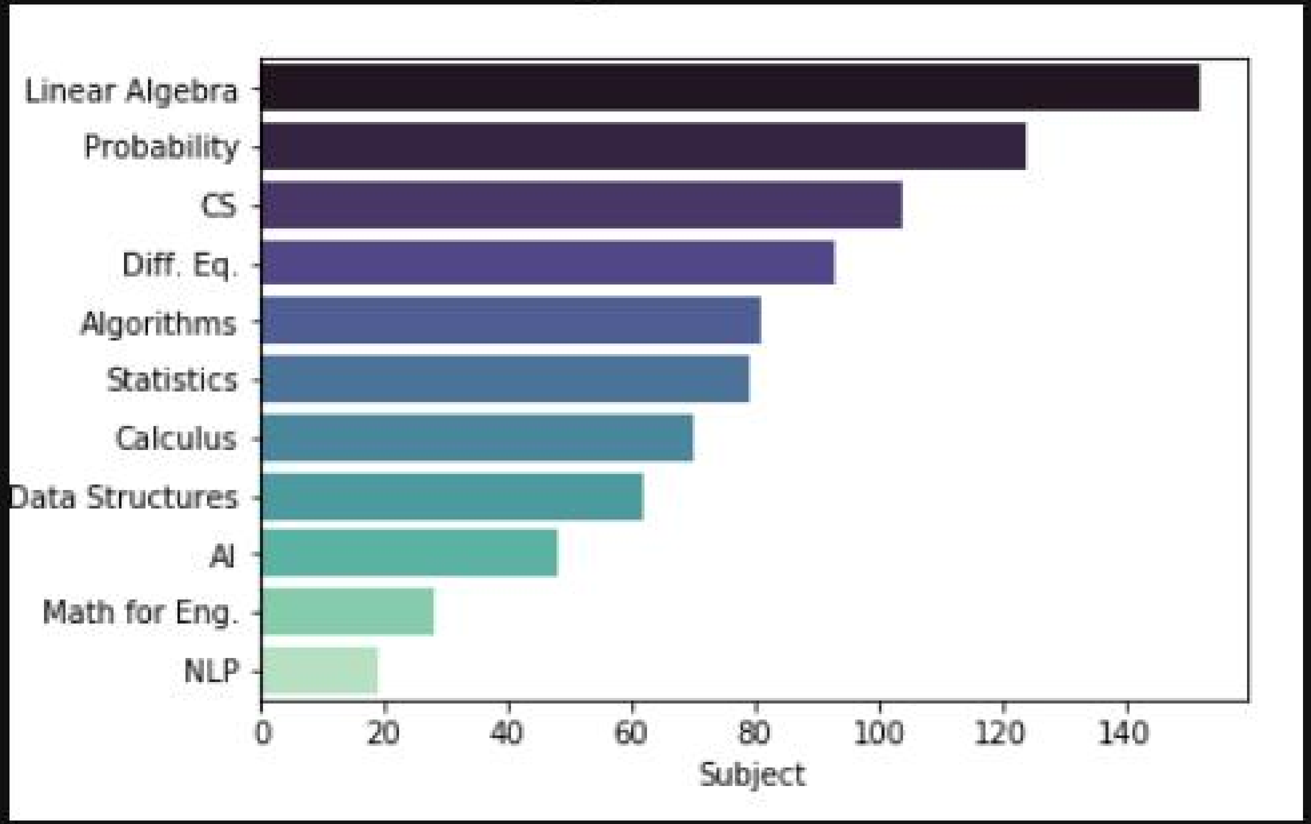
<table>
<tr><th></th><th>lecture_title</th><th>lecture_text</th><th>title</th><th>description</th><th>channelid</th></tr>
<tr><td>0</td><td>Lec39_MIT18.01SingleVariableCalculus,Fall2007-...</td><td>The following content is\nprovided under a Cre...</td><td>Lec 39 | MIT 18.01 Single Variable Calculus, F...</td><td>Lecture 39: Final review\nInstructor: David Je...</td><td>MIT OpenCourseWare</td></tr>
<tr><td>1</td><td>S01.0MathematicalBackgroundOverview--630YTQEuC...</td><td>In this sequence of segments,\nwe review some ...</td><td>S01.0 Mathematical Background Overview</td><td>MIT RES.6-012 Introduction to Probability, Spr...</td><td>MIT OpenCourseWare</td></tr>
</table>

Inspecting the Data

Sources



Subjects



Cleaning the Data

Before

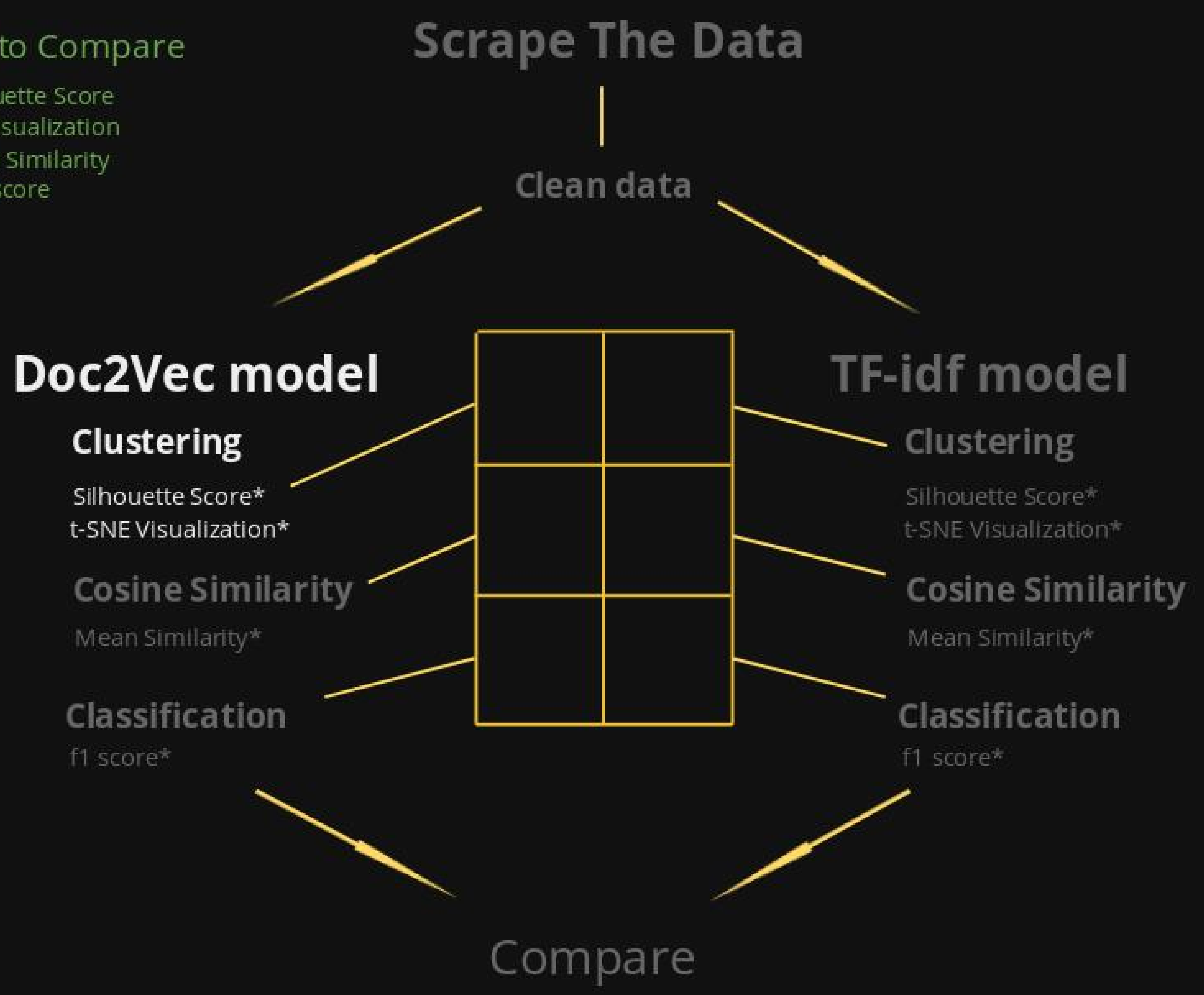
```
"The following content is\nprovided under a Creat
ive Commons license.\nYour support will help MIT
OpenCourseWare continue to offer\nhigh quality ed
ucational resources for free.\nTo make a donation
or to view additional materials from\nhundreds of
MIT courses, visit MIT OpenCourseWare at\nocw.mit
.edu. OK, so we're going to continue\nlooking at
what happens when we have non-independent variabl
es.\nSo, I'm afraid we don't take deliveries duri
ng class time,\nsorry. Please take a seat, thanks
.\n[LAUGHTER] [APPLAUSE]\nOK, so Jason, you pleas
e claim your package\nat the end of lecture. OK,\
nso last time we saw how to use Lagrange multipli
ers to find the\nminimum or maximum of a function
of several variables when the\nvariables are not
independent. And, today we're going to try\nto fi
gure out more about relations between the variabl
es,\nand how to handle functions that depend on s
everal variables\nwhen they're related. So, just
```

After

```
"OK ||Comma||  so  we are  going to continue ||Re
turn|| looking at what happens when we have non |
|Dash|| independent variables ||Period||  ||Retur
n|| So ||Comma||   I am  afraid we  do not  take
deliveries during class time ||Comma||  ||Return|
| sorry ||Period||  Please take a seat ||Comma||
thanks ||Period||  ||Return|| [LAUGHTER] [APPLAUS
E] ||Return|| OK ||Comma||  so Jason ||Comma||  y
ou please claim your package ||Return|| at the en
d of lecture ||Period||  OK ||Comma||  ||Return||
so last time we saw how to use Lagrange multiplie
rs to find the ||Return|| minimum or maximum of a
function of several variables when the ||Return||
variables are not independent ||Period||  And ||C
omma||  today  we are  going to try ||Return|| to
figure out more about relations between the varia
bles ||Comma||  ||Return|| and how to handle func
tions that depend on several variables ||Return||
when  they are  related ||Period||  So ||Comma||
just to give you an ||Return|| example ||Comma||
in physics ||Comma||  very often ||Comma||  ||Ret
```

Metrics to Compare

Silhouette Score
t-SNE Visualization
Mean Similarity
f1 score



Train a Doc2Vec Model

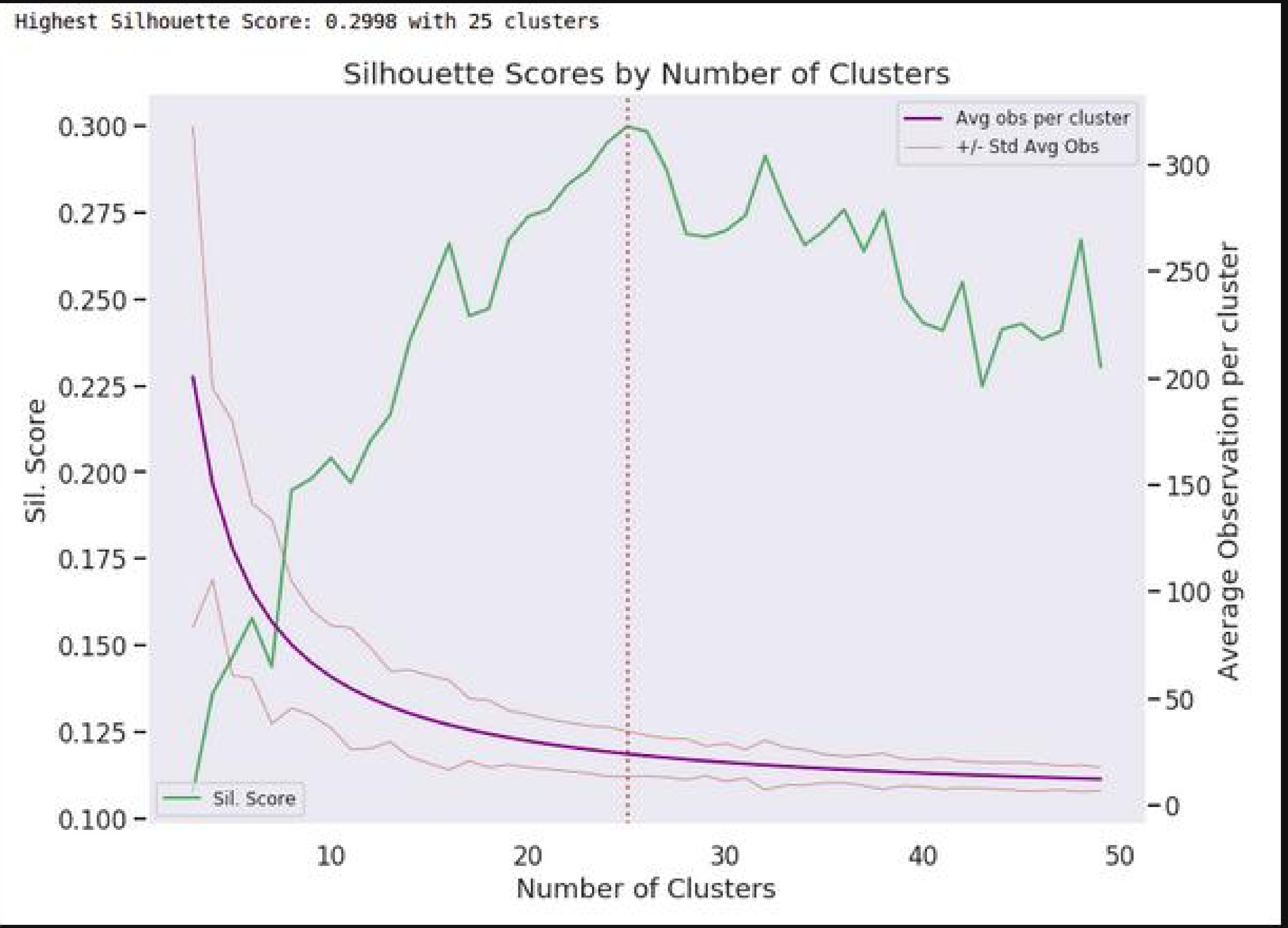
```
#max training epochs
max_epochs = 100

#train n epochs and save the model
t1 = time.time()
for epoch in range(max_epochs):
    print('iteration {0}'.format(epoch+1))
    model.train(tagged_tr,
                total_examples=model.corpus_count,
                epochs=model.iter)
    # decrease the learning rate
    model.alpha -= 0.0002
    # fix the learning rate, no decay
    model.min_alpha = model.alpha
    #print every 5 epochs
    if epoch%10 == 0:
        vecs = pd.DataFrame([model.docvecs[str(i)] for i in range(len(tagged_tr))])
        tsne_df = tsne.fit_transform(vecs,random_state=43)
        plt.figure(figsize=(12,9))
        sns.scatterplot(x=tsne_df[:,0],y=tsne_df[:,1],hue=train.Subject, legend='full')
        plt.legend(prop={'size': 8},bbox_to_anchor=[1,1])
        plt.show()

        fnclusts = []
        fsscores = []
        for no in range(6,20,1):
            t1 = time.time()
            fd2v_clusters = cluster.KMeans(n_clusters=no, random_state=43).fit_predict(vecs)
            fnclusts.append(no)
            fsscores.append(silhouette_score(vecs, fd2v_clusters, metric='cosine'))
        print('Best Number of Clusters: {}, Sillhouette score:{}'.format(fnclusts[np.argmax(fsscores)],max(fsscores)))

        d2v_clusters = cluster.KMeans(n_clusters=fnclusts[np.argmax(fsscores)], random_state=43).fit_predict(vecs)
        plt.figure(figsize=(12,9))
        sns.scatterplot(x=tsne_df[:,0],y=tsne_df[:,1],hue=d2v_clusters, legend='full', palette='rainbow')
        plt.legend(prop={'size': 8}, bbox_to_anchor=[1,1])
        plt.show()

print("done!")
```

Clustering the Doc2Vec Vectors

KMeans

Highest Silhouette Score: .2998

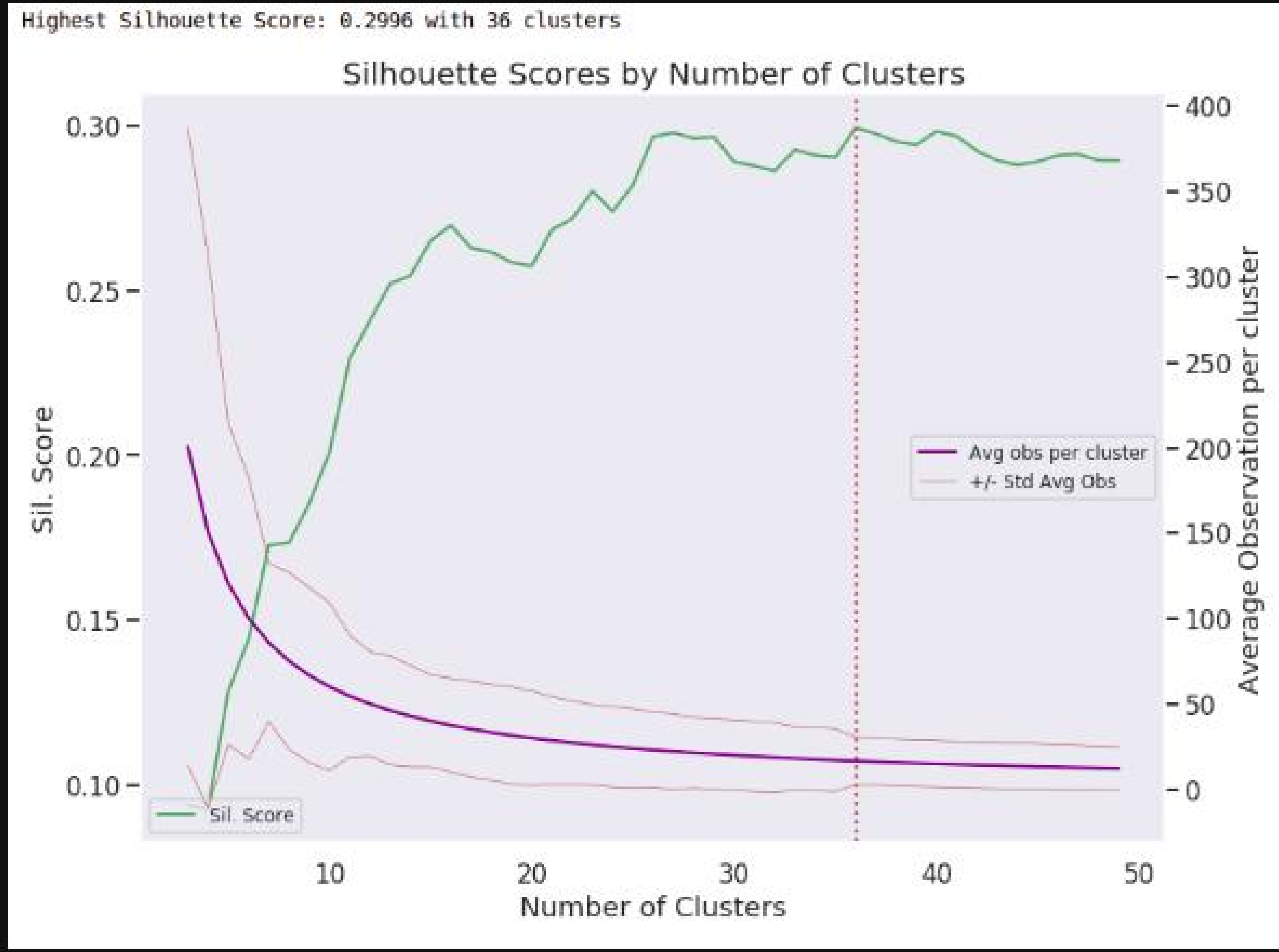
Number of Clusters: 25



Agglomerative

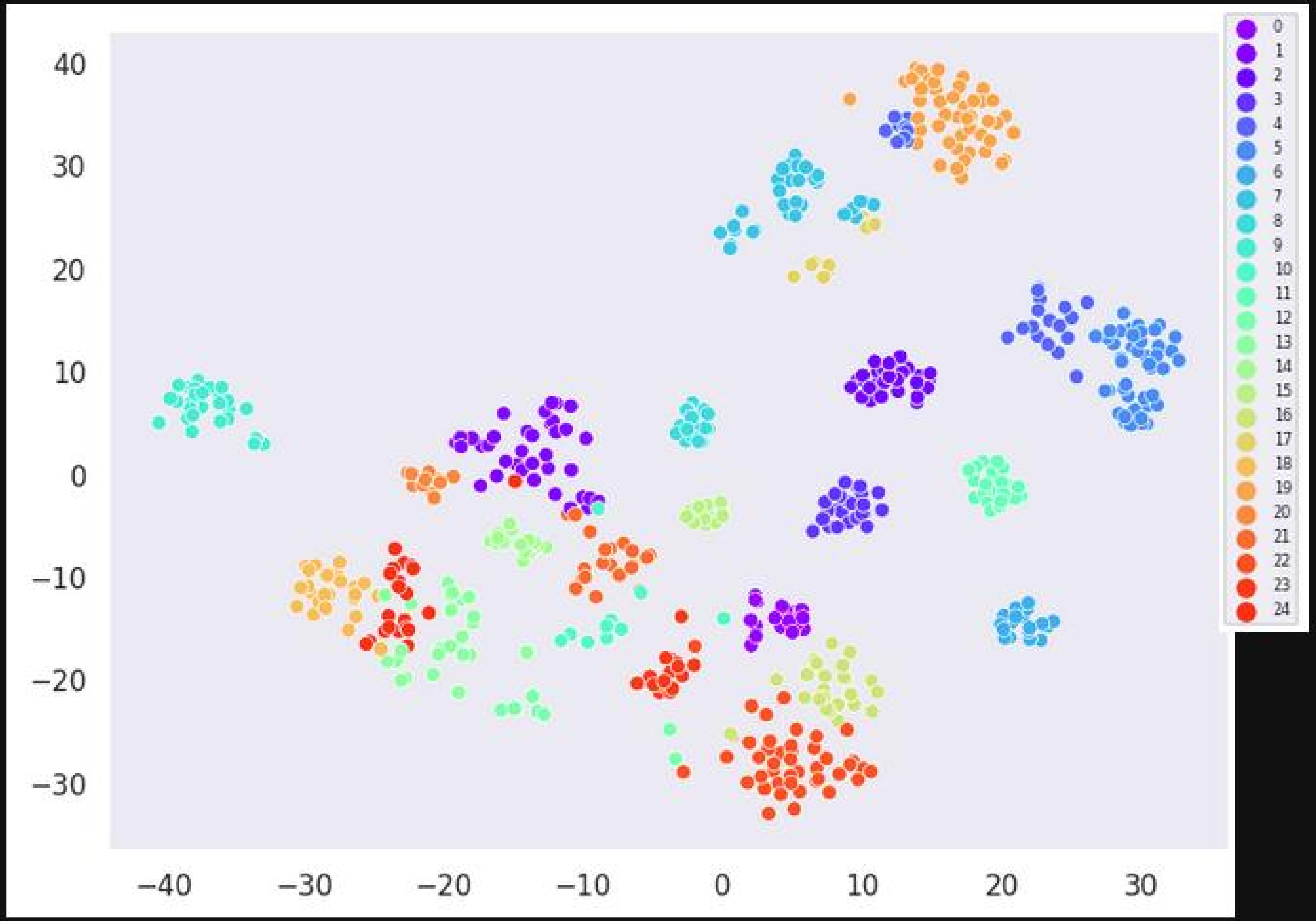
Highest Silhouette Score: .2996

Number of Clusters: 36

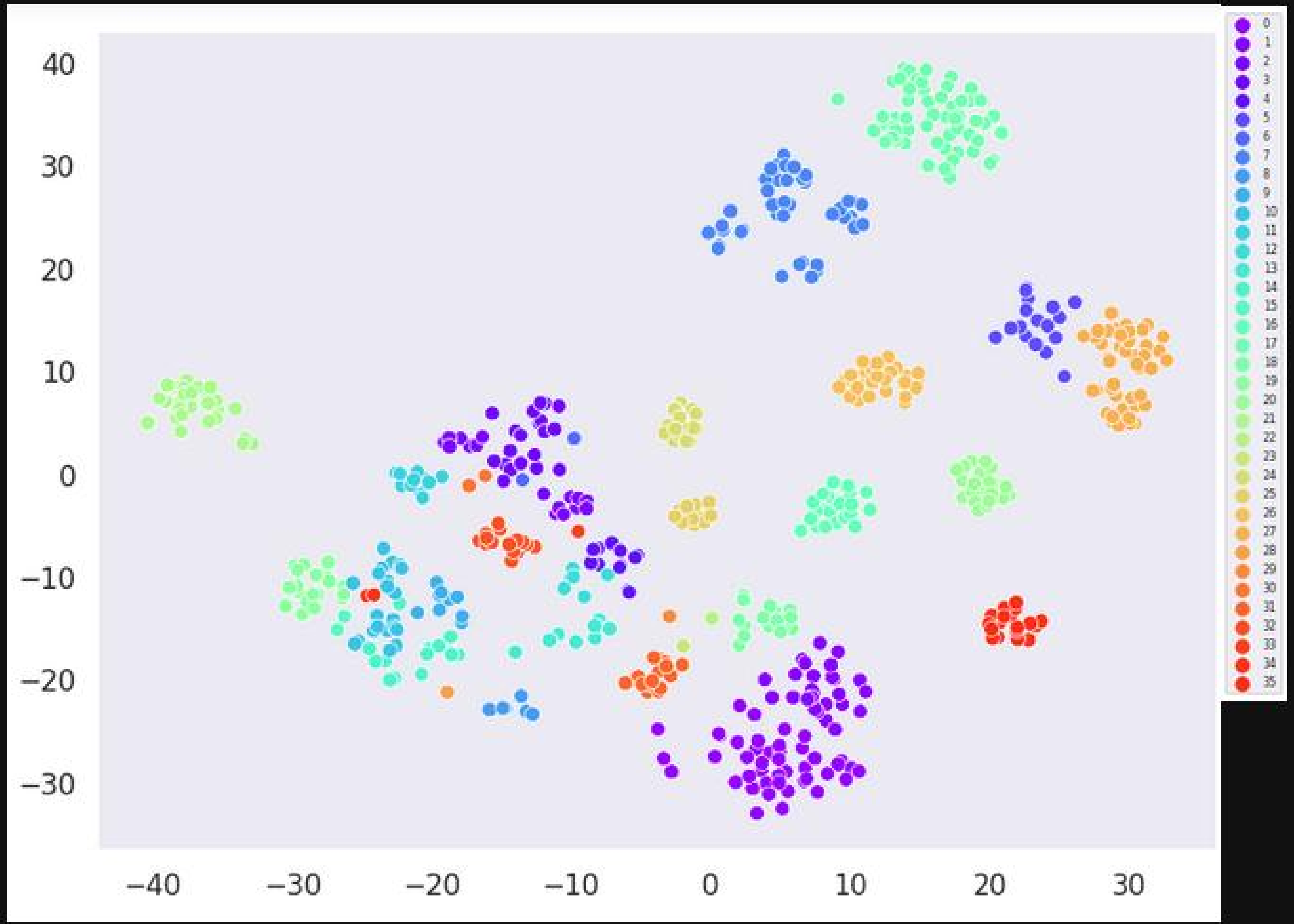


Visualizing Clusters with t-SNE decomposition

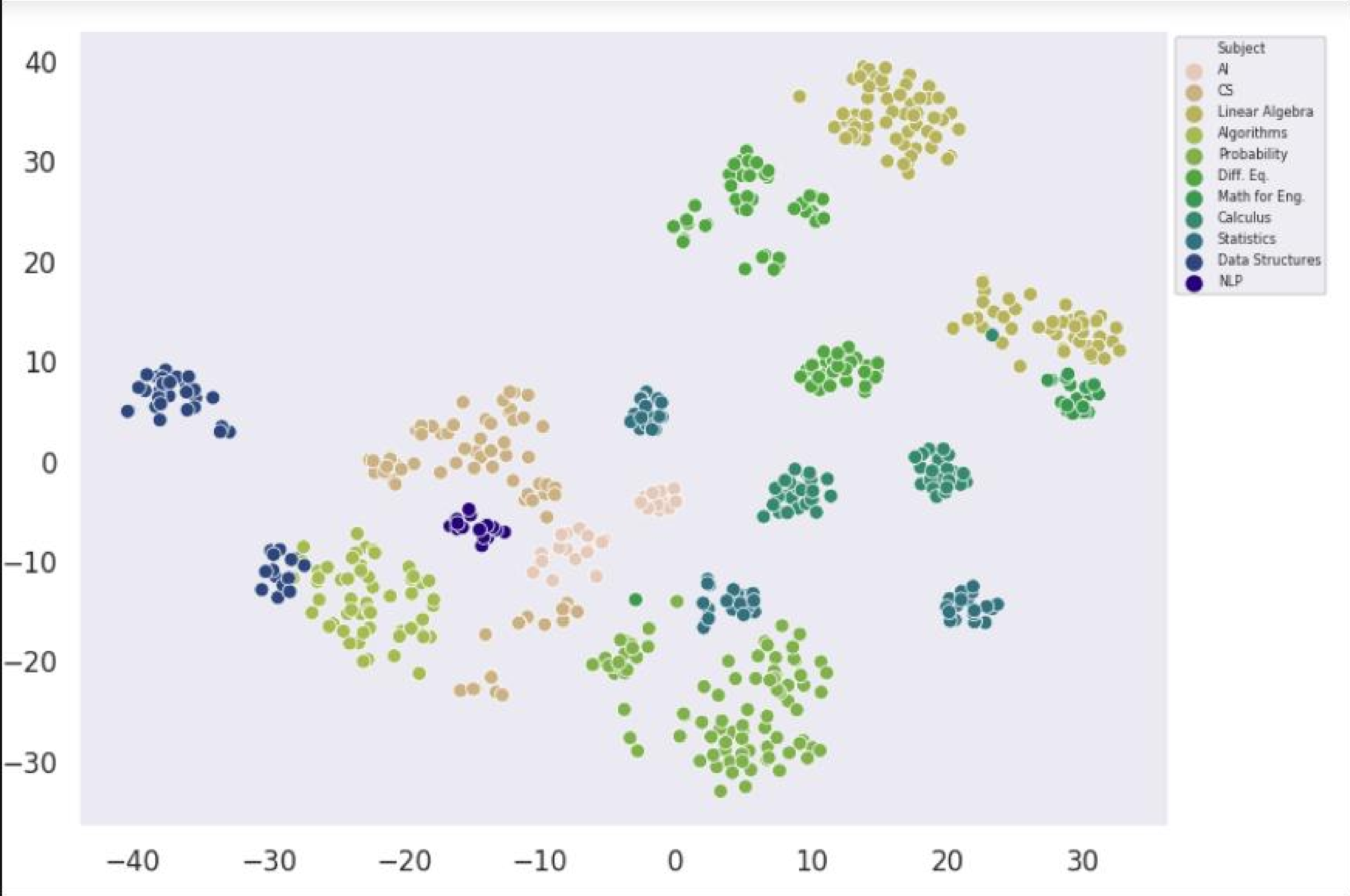
K Means



Agglomerative Clustering



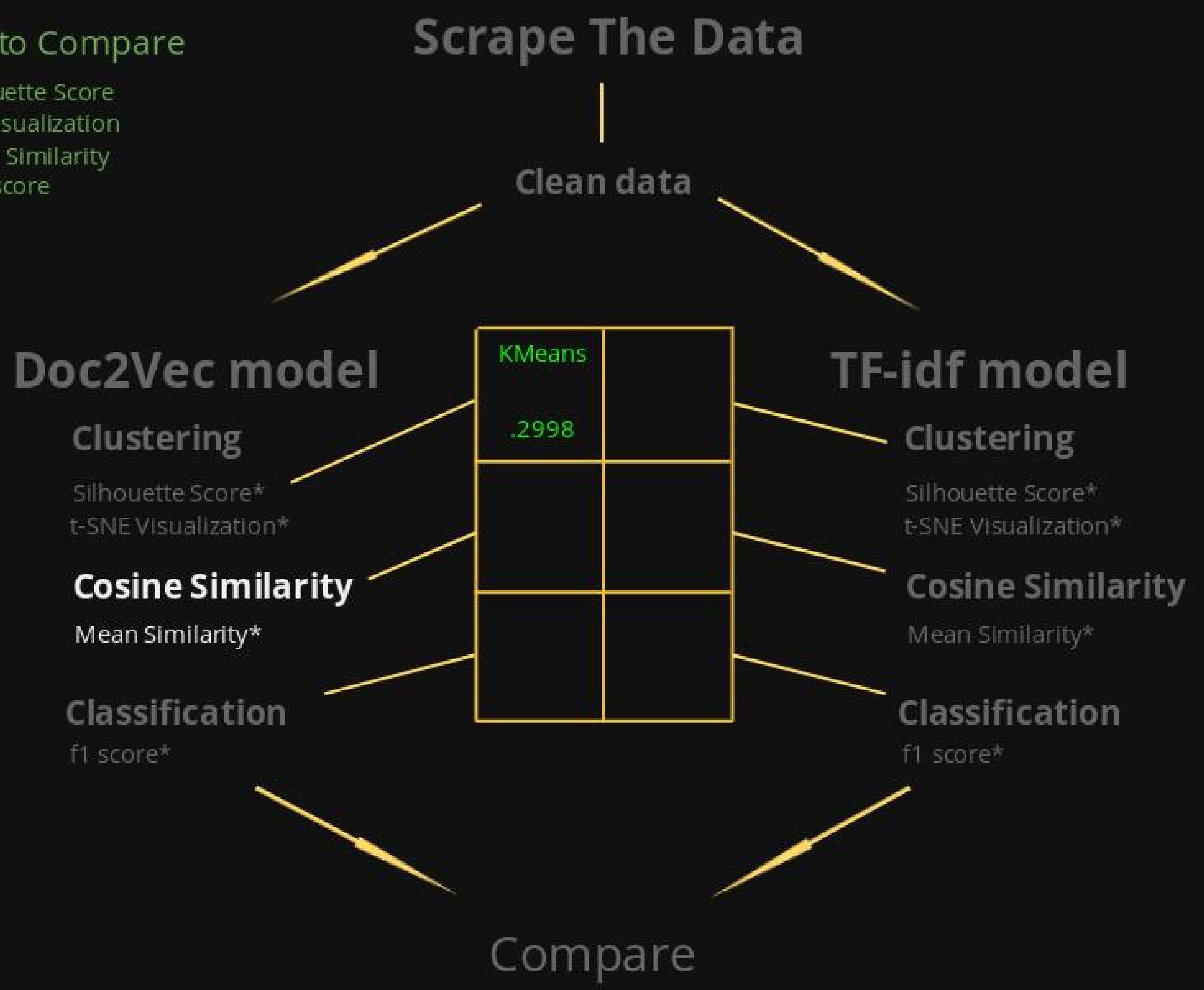
True Labels



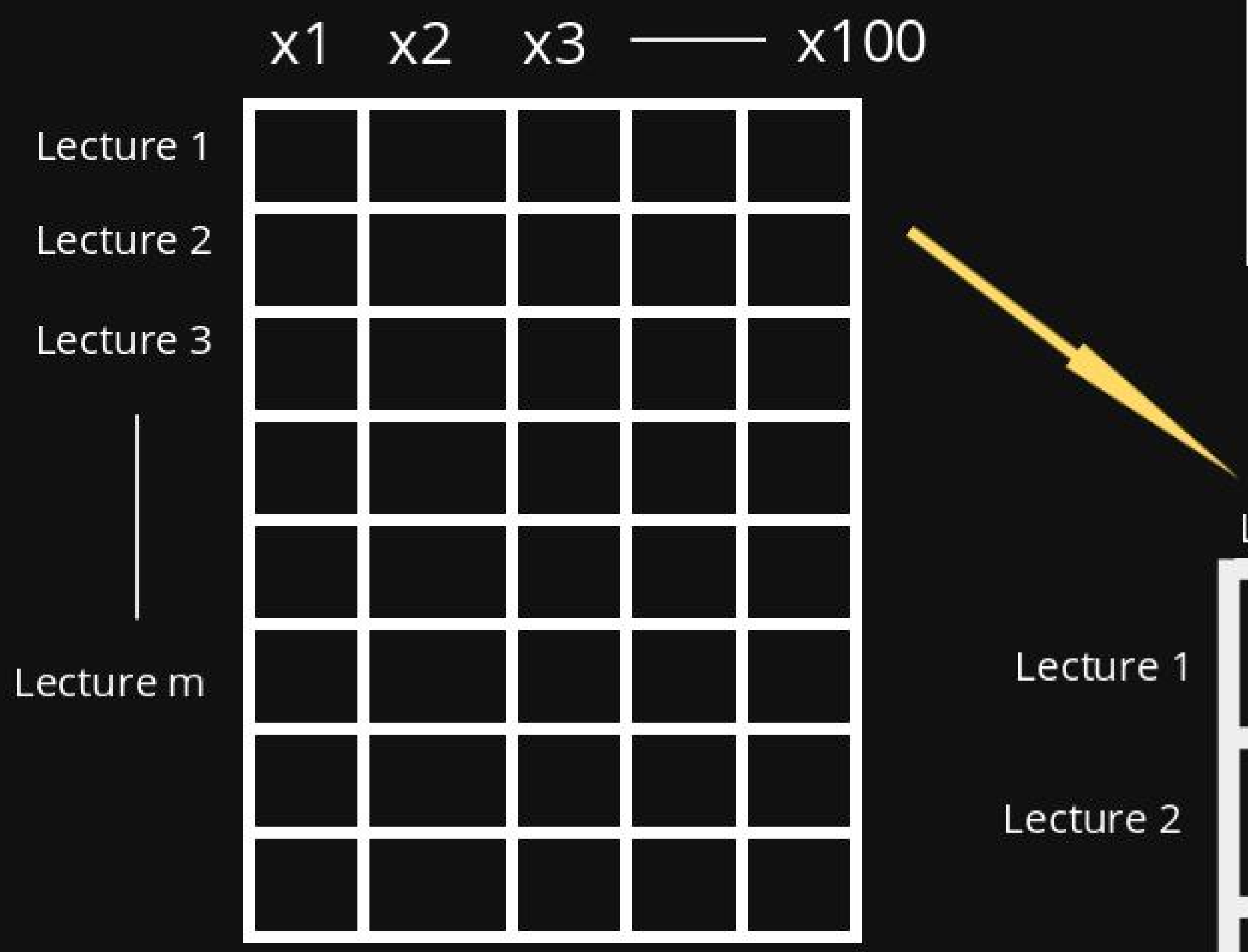
Subject
AI
CS
Linear Algebra
Algorithms
Probability
Diff. Eq.
Math for Eng.
Calculus
Statistics
Data Structures
NLP
40
30
20
10
0
−10
−20
−30
−40
−30
−20
−10
0
10
20
30

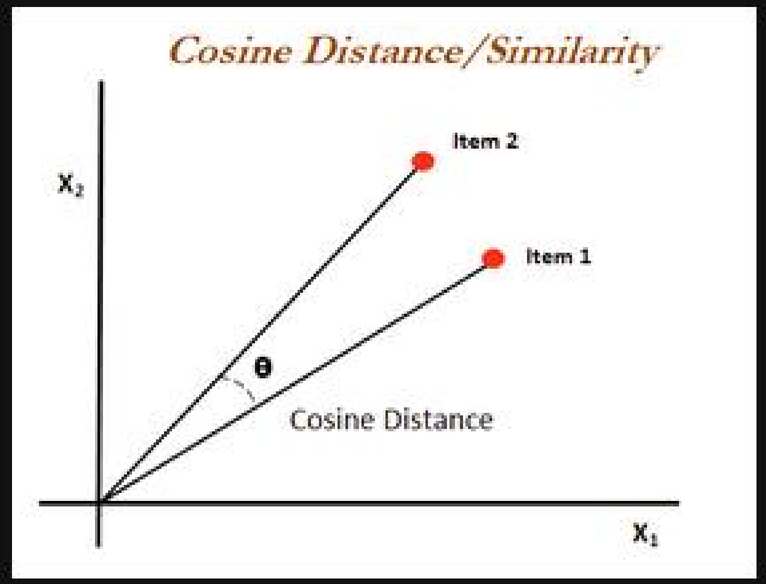
Metrics to Compare

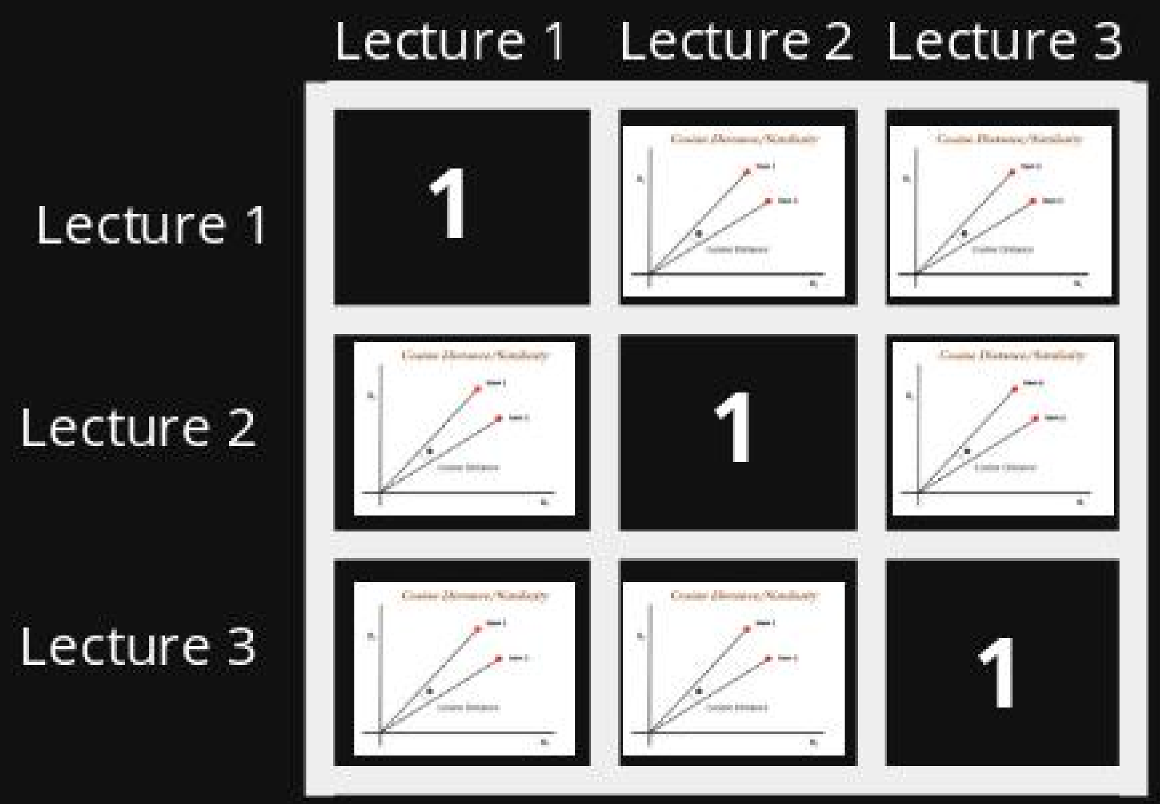
Silhouette Score
t-SNE Visualization
Mean Similarity
f1 score



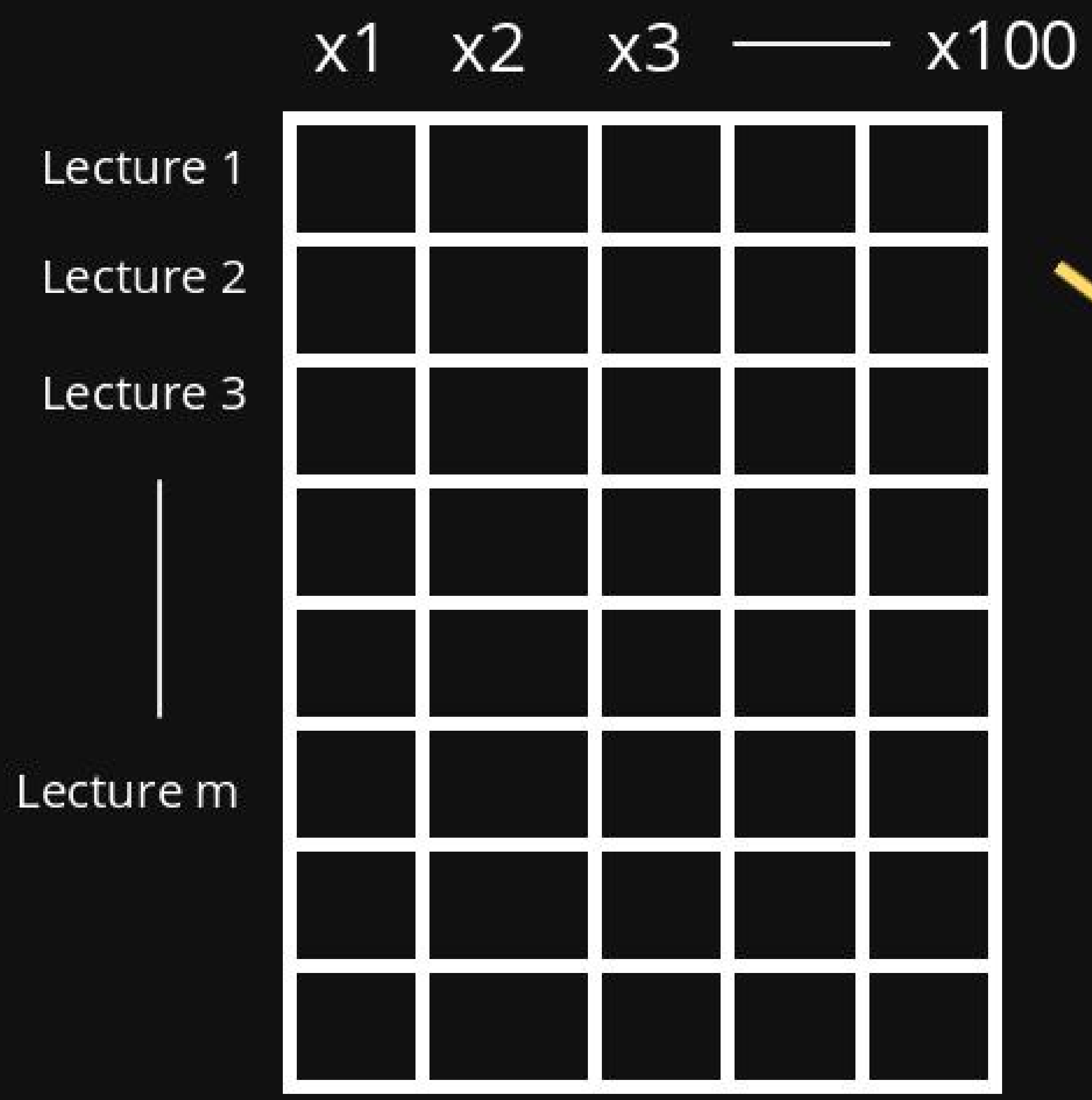
Cosine Similarity Matrix

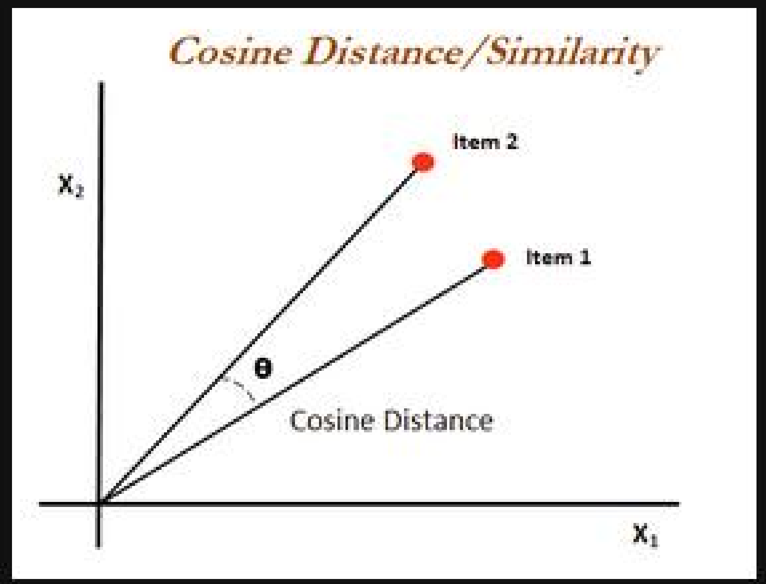


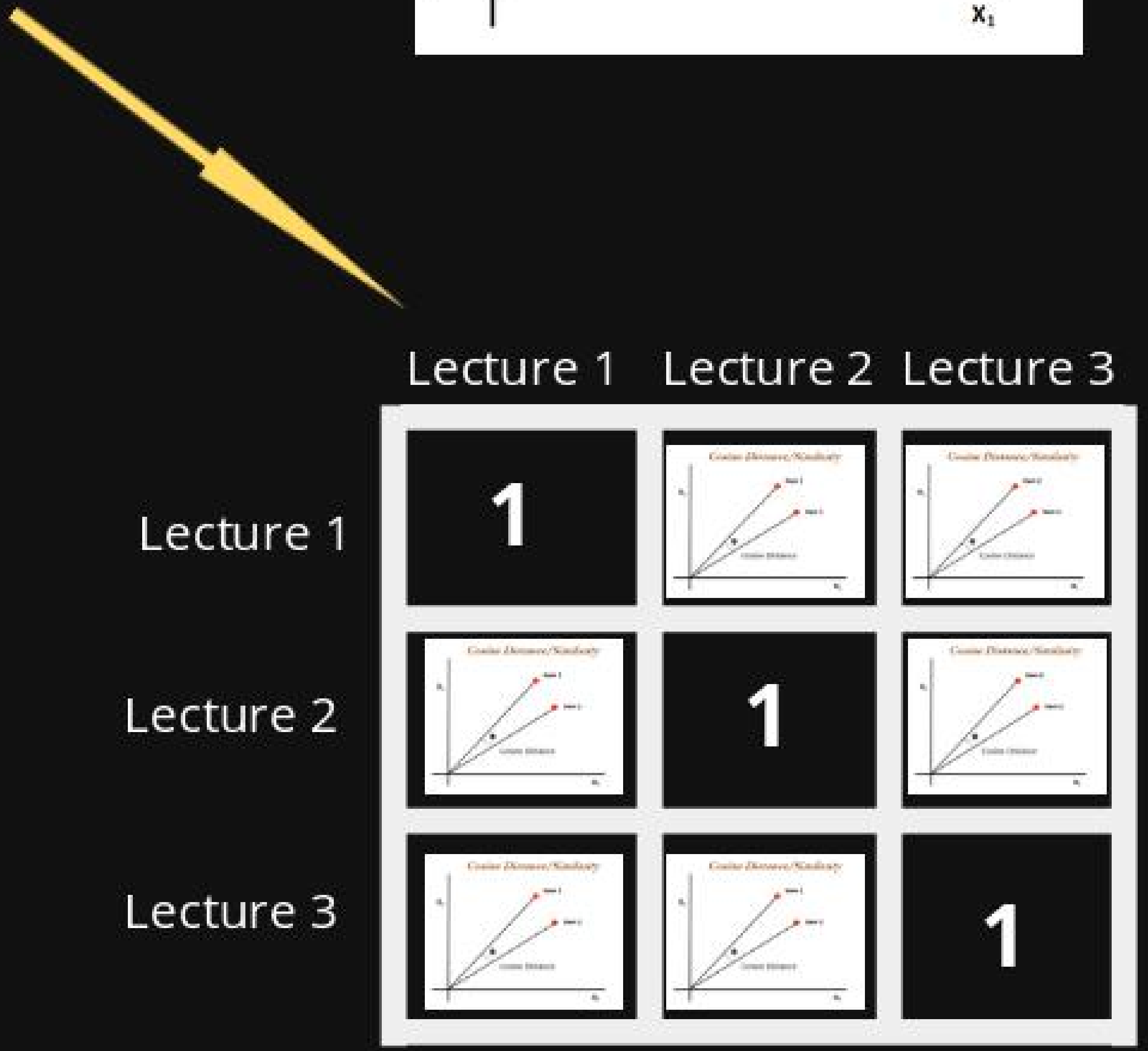




Cosine Similarity Matrix







Calculate Mean Similarity

```
#calculate the mean similarity between lectures
d2v_fullsim.insert(0, 'mean_similarity', d2v_fullsim.mean(axis=1))
```

Look up the most similar

```
#test the response on a lecture
lecture = train.title[62]
d2v_fullsim[[lecture,'Subject','mean_similarity']].sort_values(by=[lecture],ascending=False)[:15]
```

title title	17. Succinct Structures I	Subject	mean_similarity
17. Succinct Structures I	1.000000	Data Structures	0.101751
15. Static Trees	0.751220	Data Structures	0.116894
16. Strings	0.715761	Data Structures	0.109921
11. Integer Models	0.706285	Data Structures	0.111308

Look at the *most* similar

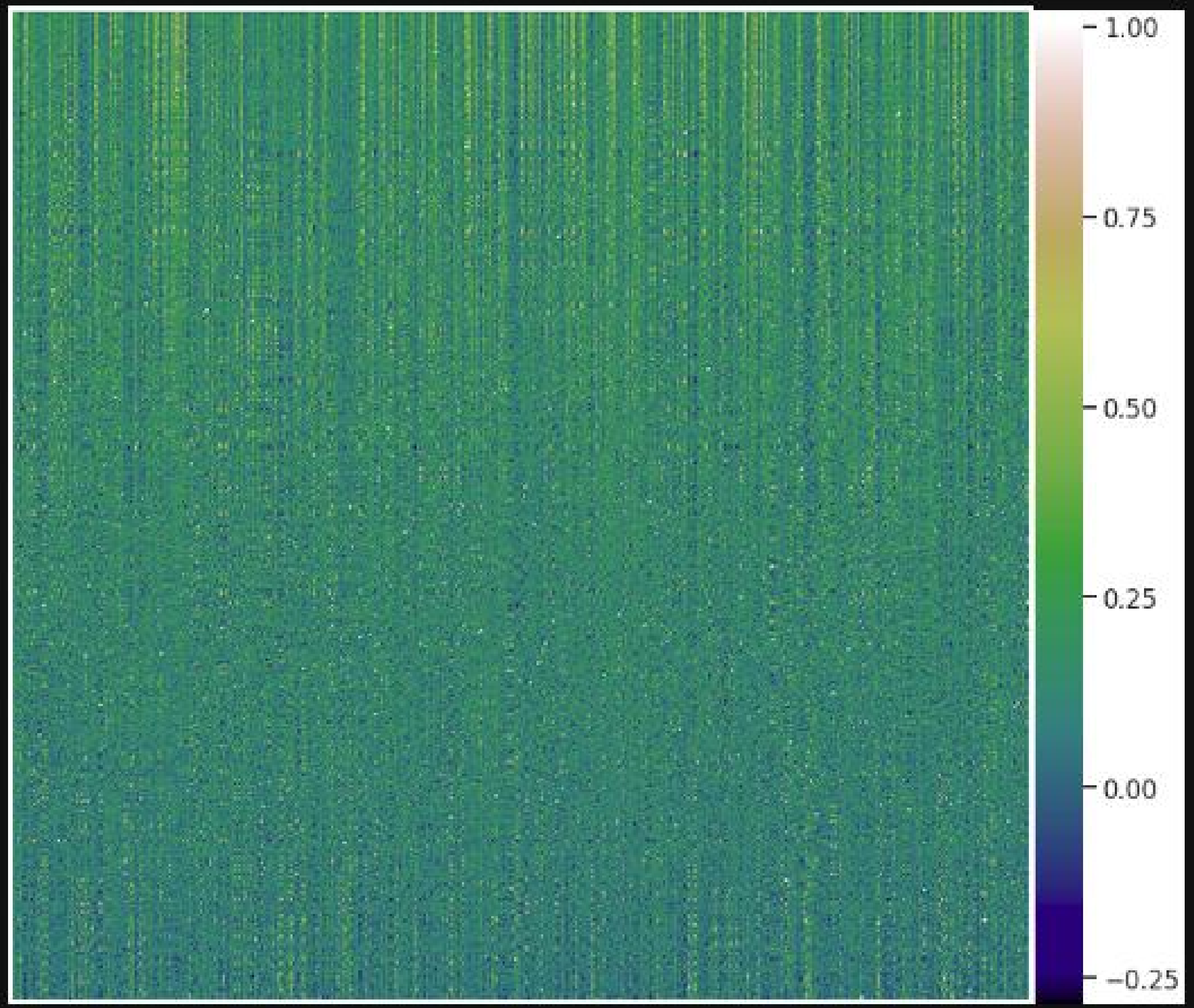
```
#what is the highest mean similarity?
d2v_fullsim.sort_values(by='mean_similarity',ascending=False)[['mean_similarity','Subject']][:10]
```

title title	mean_similarity	Subject
2nd order linear homogeneous differential equations 1 | Khan Academy	0.219275	Diff. Eq.
Using the Laplace transform to solve a nonhomogeneous eq | Laplace transform | Khan Academy	0.218520	Diff. Eq.
Undetermined coefficients 2 | Second order differential equations | Khan Academy	0.217208	Diff. Eq.

Average Mean Similarity: .1445

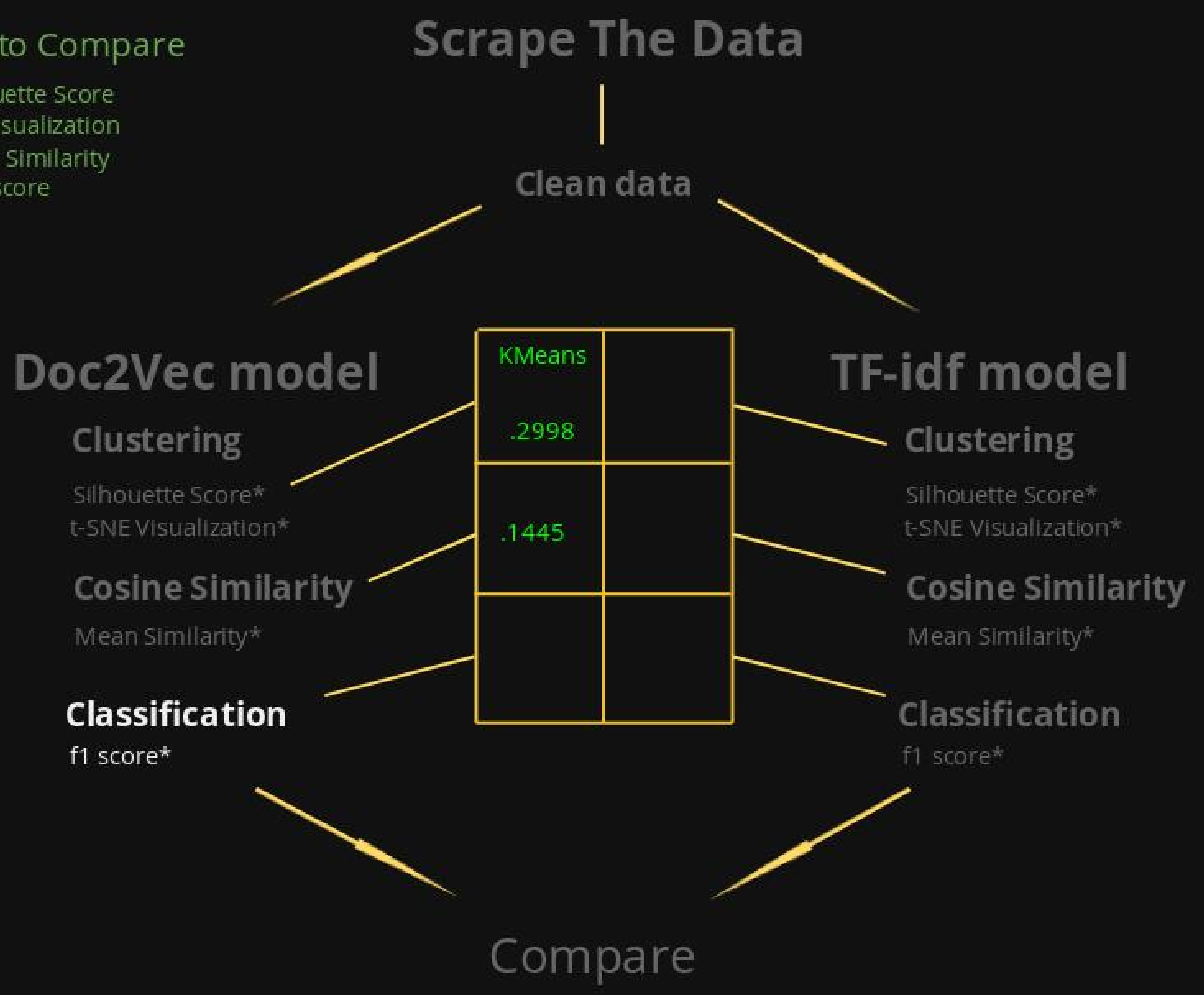
```
d2v_fullsim.groupby('Subject')['mean_similarity'].mean().mean()
```

0.14450145113135823



Metrics to Compare

Silhouette Score
t-SNE Visualization
Mean Similarity
f1 score



Document Classification with Doc2Vec

Creating testing data from the model

```python
# Extract vectors from doc2vec model
X_train = pd.DataFrame([model.docvecs[str(i)] for i in range(len(tagged_tr))])
y_train = train.Subject

# Extract test values
X_test = pd.DataFrame([model.infer_vector(tagged_test[i][0]) for i in range(len(tagged_test))])
y_test = test.Subject
```

Use 'infer_vector' to generate test set vectors.

```python
model.infer_vector(tagged_test[1][0])
```

```
array([ 0.77725834,  1.0438219 , -0.5684751 ,  1.2010766 , -1.317795  ,
        1.0853385 , -0.8922233 , -0.5825623 , -0.8660668 ,  1.7947867 ,
        1.1740577 ,  1.1709765 ,  1.0654398 , -0.00828252, -0.17509806,
        1.941227  , -0.4855255 , -1.0252407 ,  0.10240954,  1.3939342 ,
       -0.18964739, -0.77770597, -1.3215303 , -0.5187984 , -1.059191  ,
```

Test set vectors

```python
X_test.head()
```

	0	1	2	3	4	5	6	7	8	9	...
0	1.187475	0.315449	-0.118757	0.984394	-1.165851	0.306538	-1.166884	-1.232077	-0.186273	-0.072497	...
1	0.937598	1.352486	-0.575855	1.330695	-1.452793	0.929479	-0.978539	-0.424853	-0.758674	1.844291	...
2	-0.616663	-2.237037	-0.326477	0.442156	1.103491	1.012133	0.714710	-0.855507	1.316695	-1.475574	...

Classification with Doc2Vec

SVC = .71

```
svc_df.sort_values(by='f1',ascending=False)[:10]
```

	f1	recall	precision	kernel	df_shape	C
0	0.71	0.71	0.88	linear	ovr	0.5
11	0.71	0.71	0.88	linear	multinomial	1.0
1	0.71	0.71	0.88	linear	ovr	1.0
19	0.71	0.71	0.88	linear	multinomial	30.0
18	0.71	0.71	0.88	linear	multinomial	25.0

Random Forest = .7

```
rfc_df.sort_values(by='f1',ascending=False)[:10]
```

	f1	recall	precision	n_ests	max_depths	min_leaf	min_split	ctriteron
513	0.7	0.69	0.88	100	6	2	3	entropy
658	0.7	0.70	0.88	300	12	2	4	entropy
733	0.7	0.70	0.89	500	7	5	3	entropy
734	0.7	0.70	0.89	500	7	5	4	entropy
514	0.7	0.69	0.88	100	6	2	4	entropy

Gradient Boosting = .51

```
gbc_df.sort_values(by='f1',ascending=False)
```

	f1	recall	precision	n_ests	max_depths	min_leaf	min_split	loss
397	0.51	0.54	0.70	900	2	5	3	deviance
109	0.51	0.54	0.70	300	2	5	3	deviance
206	0.51	0.54	0.70	500	2	5	4	deviance
302	0.51	0.54	0.70	700	2	5	4	deviance
301	0.51	0.54	0.70	700	2	5	3	deviance

Logistic Regression = .95

```
lrc_df.sort_values(by='f1',ascending=False)[:20]
```

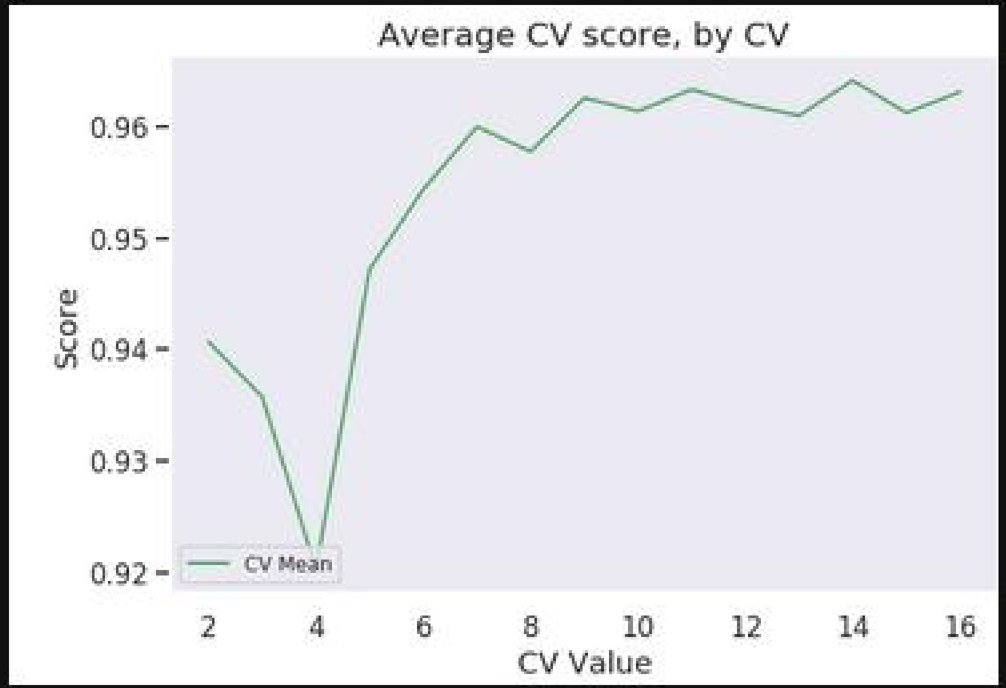
	f1	recall	precision	solver	penalty	class	C
59	0.95	0.95	0.95	saga	l2	multinomial	30
58	0.95	0.95	0.95	saga	l2	multinomial	25
57	0.95	0.95	0.95	saga	l2	multinomial	20
56	0.95	0.95	0.95	saga	l2	multinomial	17
55	0.95	0.95	0.95	saga	l2	multinomial	15

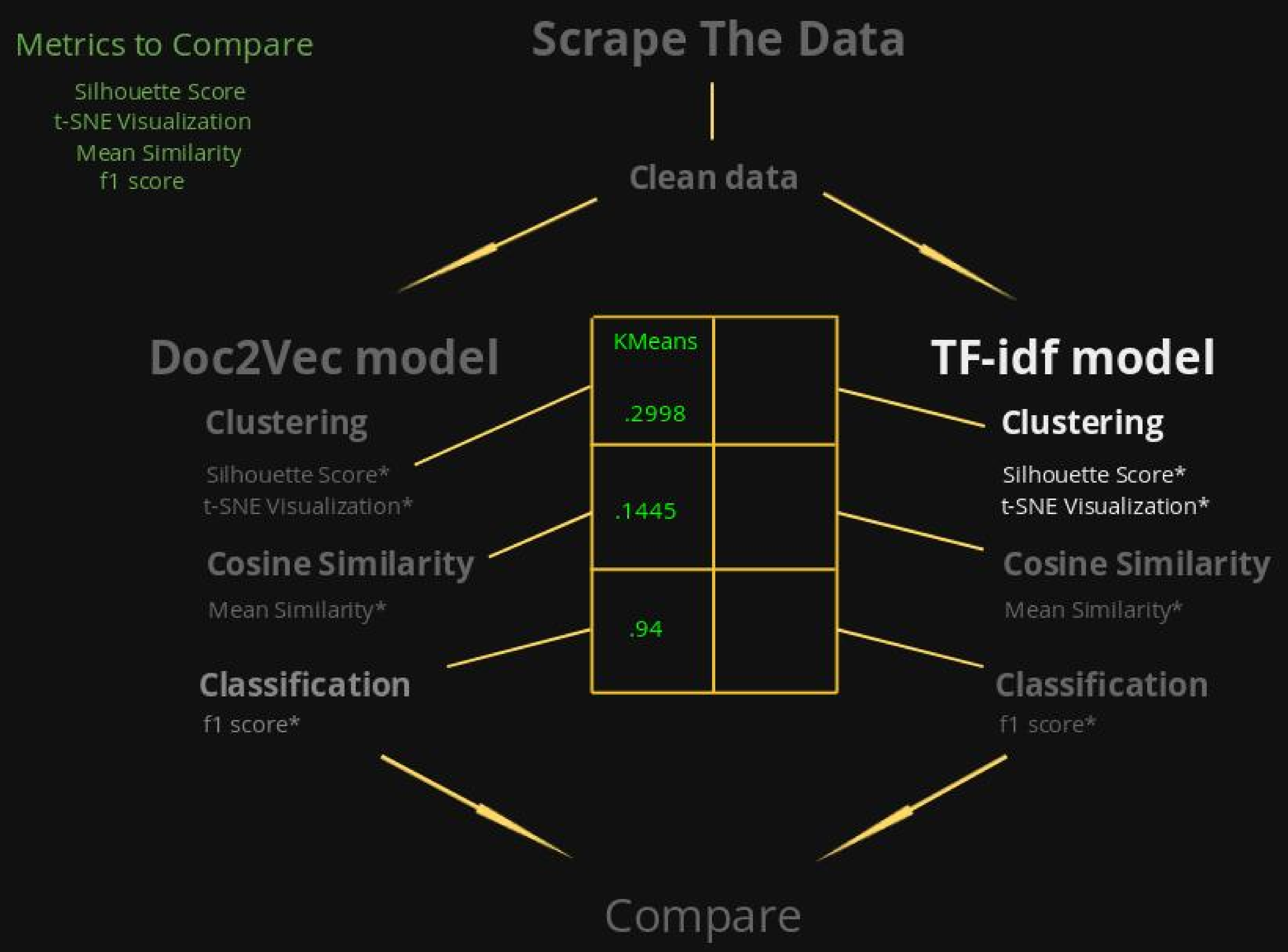
Classification with Doc2Vec Logistic Regression

Highest f1 score: .94

	0	1	2	3	4	5	6	7	8	9	10
AI	16	0	0	0	0	0	0	0	0	0	0
Algorithms	0	19	1	0	2	0	0	0	0	0	0
CS	0	1	28	1	0	0	0	0	0	0	1
Calculus	0	0	0	22	0	0	0	0	0	0	0
Data Structures	0	0	0	0	18	0	0	0	0	0	0
Diff. Eq.	0	0	0	0	0	23	0	0	0	0	0
Linear Algebra	0	0	0	1	0	0	50	0	0	0	1
Math for Eng.	0	0	0	0	0	0	1	9	0	0	0
NLP	0	0	0	0	0	0	0	0	4	0	0
Probability	0	0	4	0	0	0	0	0	0	30	2
Statistics	0	0	0	0	0	0	0	0	0	0	24

	precision	recall	f1-score	support
AI	1.00	1.00	1.00	16
Algorithms	0.95	0.86	0.90	22
CS	0.85	0.90	0.88	31
Calculus	0.92	1.00	0.96	22
Data Structures	0.90	1.00	0.95	18
Diff. Eq.	1.00	1.00	1.00	23
Linear Algebra	0.98	0.96	0.97	52
Math for Eng.	1.00	0.90	0.95	10
NLP	1.00	1.00	1.00	4
Probability	1.00	0.83	0.91	36
Statistics	0.86	1.00	0.92	24
micro avg	0.94	0.94	0.94	258
macro avg	0.95	0.95	0.95	258
weighted avg	0.95	0.94	$\boxed{0.94}$	258




Metrics to Compare
Silhouette Score
t-SNE Visualization
Mean Similarity
f1 score
Scrape The Data
Clean data
Doc2Vec model
Clustering
Silhouette Score*
t-SNE Visualization*
Cosine Similarity
Mean Similarity*
Classification
f1 score*
KMeans
.2998
.1445
.94
TF-idf model
Clustering
Silhouette Score*
t-SNE Visualization*
Cosine Similarity
Mean Similarity*
Classification
f1 score*
Compare

Tf-idf and Clustering

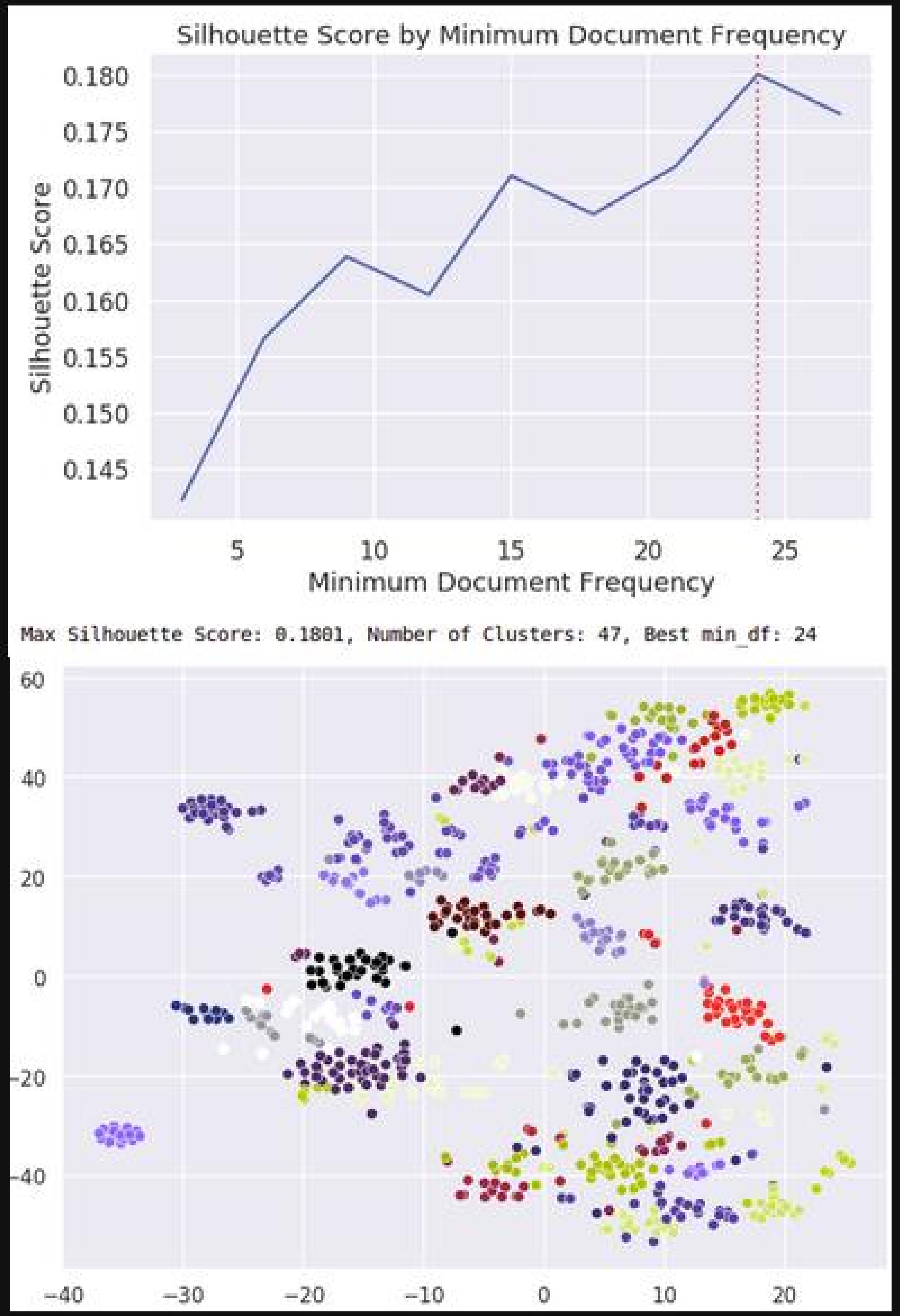
Try different values for min_df

for each tf-idf iteration, cluster vectors from 8-50 clusters

```
min_dfs = []
n_clusts = []
sil_scores = []

sns.set(style='darkgrid', context='talk')
tsne = TSNE(2,random_state=43)
for number in range(3,30,3):
    vectorizer = TfidfVectorizer(max_df=0.50, # drop words that occur in more 50% of the sentences
                                 min_df=number, # only use words that appear at least 25
                                 stop_words='english', #use english stopwords
                                 lowercase=True, #lowercase
                                 use_idf=True, #idf
                                 norm=u'l2',  #normalization
                                 smooth_idf=True)#add 1 to all words to prevent 0 division

    X_idf = vectorizer.fit_transform(X)
    vecs = X_idf.todense()
    tsne_df = tsne.fit_transform(vecs)
    plt.figure(figsize=(12,9))
    #drawscatter plot for each tf-idf iteration
    sns.scatterplot(x=tsne_df[:,0],y=tsne_df[:,1],hue=y, legend='full', palette='gist_earth')
    plt.legend(prop={'size': 8},bbox_to_anchor=[1,1])
    plt.show()

    #Cluster the vectors from 8 to 50 clusters
    fnclusts = []
    fsscores = []
    for no in range(8,50,3):
        agglo = cluster.AgglomerativeClustering(n_clusters=no, affinity='cosine',linkage='average').fit_predict(vecs)
        fnclusts.append(no)
        fsscores.append(silhouette_score(vecs, agglo, metric='cosine'))
    #for each round of clustering, print the best performer and t-SNE
    print("tfidf min_df: {}".format(number))
    print('Best Number of Clusters: {}, Sillhouette score:{}'.format(fnclusts[np.argmax(fsscores)],max(fsscores)))

    agglo = cluster.AgglomerativeClustering(n_clusters=fnclusts[np.argmax(fsscores)]).fit_predict(vecs)
    plt.figure(figsize=(12,9))
    sns.scatterplot(x=tsne_df[:,0],y=tsne_df[:,1],hue=d2v_clusters, legend='full', palette='gist_stern')
    plt.legend(prop={'size': 8}, bbox_to_anchor=[1,1])
    plt.show()
    min_dfs.append(number)
    n_clusts.append(fnclusts[np.argmax(fsscores)])
    sil_scores.append(max(fsscores))
```

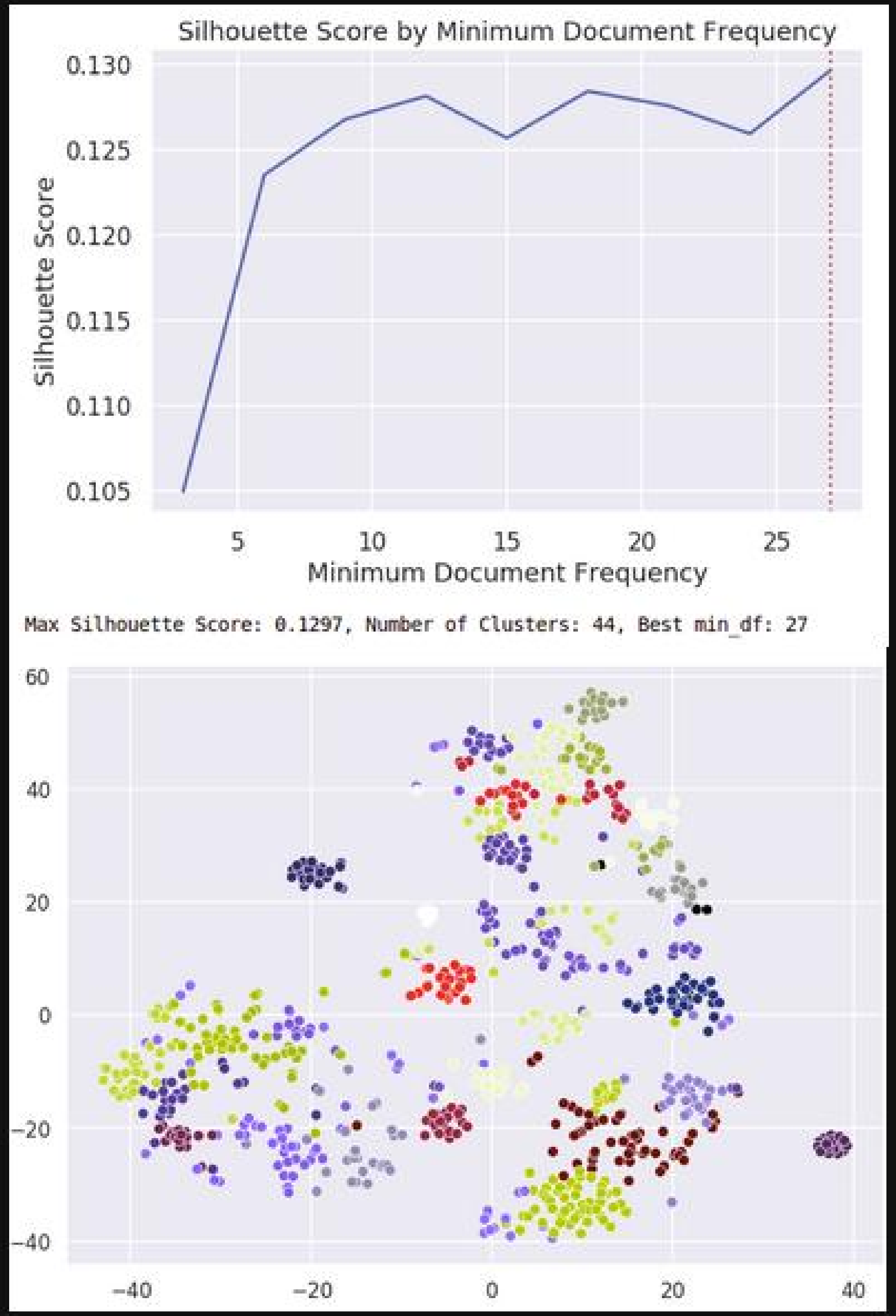
K Means

Highest Silhouette Score: .1801



Agglomerative

Highest Silhouette Score: .1297



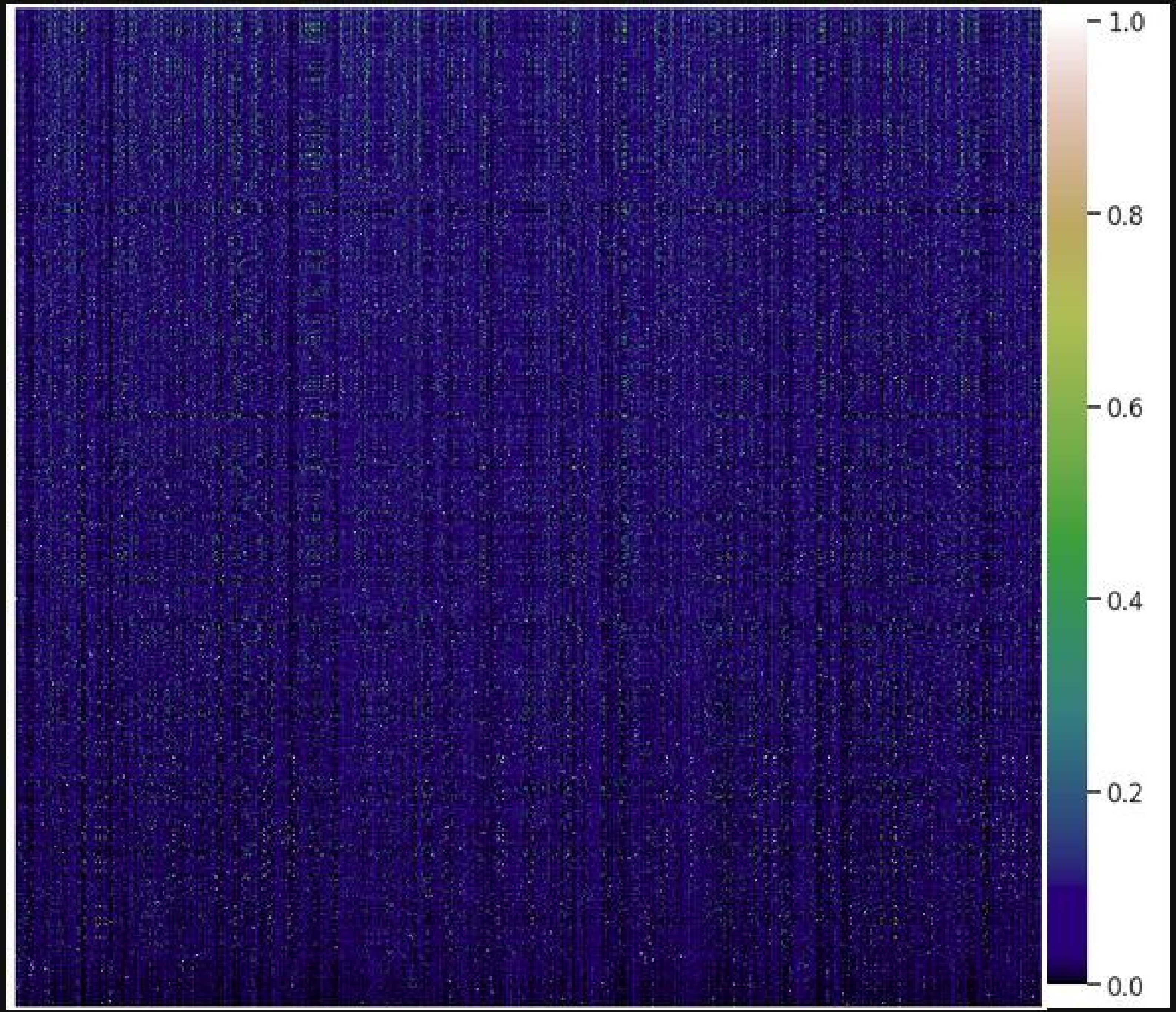
Actual Labels

Cosine Similarity

Average Mean Similarity: .065

```
tf_sim.groupby('Subject')['mean_similarity'].mean().mean()
```

0.06552274759985961



Look up related items

```
#test the response on a lecture
lecture = lectures.title[62]
tf_sim[[lecture,'Subject','mean_similarity']].sort_values(by=[lecture],ascending=False)[:10]
```

title	17. Succinct Structures I	Subject	mean_similarity
title			
17. Succinct Structures I	1.000000	Data Structures	0.056662
18. Succinct Structures II	0.761937	Data Structures	0.060340
13. Integer Lower Bounds	0.652982	Data Structures	0.065812
14. Sorting in Linear Time	0.627629	Data Structures	0.064835

Look at *most* similar

```
#what is the highest mean similarity?
tf_sim.sort_values(by='mean_similarity',ascending=False)[['mean_similarity','Subject']][:10]
```

title	mean_similarity	Subject
title		
Lec 34 | MIT 18.02 Multivariable Calculus, Fall 2007	0.113506	Calculus
Lec 28 | MIT 18.03 Differential Equations, Spring 2006	0.111477	Diff. Eq.
Review Session: Midterm Review	0.111110	NLP
Lec 3 | MIT 18.02 Multivariable Calculus, Fall 2007	0.110760	Calculus
Lecture 22: Transformations and Convolutions | Statistics 110	0.108052	Statistics

Metrics to Compare

Silhouette Score
t-SNE Visualization
Mean Similarity
f1 score

Scrape The Data

Clean data

Doc2Vec model

Clustering

Silhouette Score*
t-SNE Visualization*

Cosine Similarity

Mean Similarity*

Classification

f1 score*

KMeans	KMeans
.2998	.1800
.1445	.065
.94	

TF-idf model

Clustering

Silhouette Score*
t-SNE Visualization*

Cosine Similarity

Mean Similarity*

Classification

f1 score*

Compare

Classification modeling with tf-idf

Random Forest = .92

```
rfc_df.sort_values(by='f1',ascending=False)
```

	f1	recall	precision	n_ests	max_depths	min_leaf	min_split	ctriteron
852	0.92	0.90	0.95	700	12	3	2	entropy
567	0.92	0.90	0.96	100	12	3	5	entropy
566	0.92	0.90	0.96	100	12	3	4	entropy
565	0.92	0.90	0.96	100	12	3	3	entropy
564	0.92	0.90	0.96	100	12	3	2	entropy

SVC = .95

```
svc_df.sort_values(by='f1',ascending=False)[:5]
```

	f1	recall	precision	kernel	df_shape	C	min_df
241	0.95	0.94	0.96	linear	ovr	2	20
219	0.95	0.95	0.96	linear	multinomial	30	18
52	0.95	0.95	0.95	linear	multinomial	5	10
212	0.95	0.95	0.96	linear	multinomial	5	18
213	0.95	0.95	0.96	linear	multinomial	8	18

Gradient Boosting = .90

```
gbc_df.sort_values(by='f1',ascending=False)
```

	f1	recall	precision	n_ests	max_depths	min_leaf	min_split
33	0.90	0.89	0.91	500	6	5	2
17	0.90	0.89	0.92	300	7	5	5
62	0.90	0.89	0.91	700	6	5	5
61	0.90	0.89	0.91	700	6	5	3
60	0.90	0.89	0.91	700	6	5	2

Logistic Regression: .95

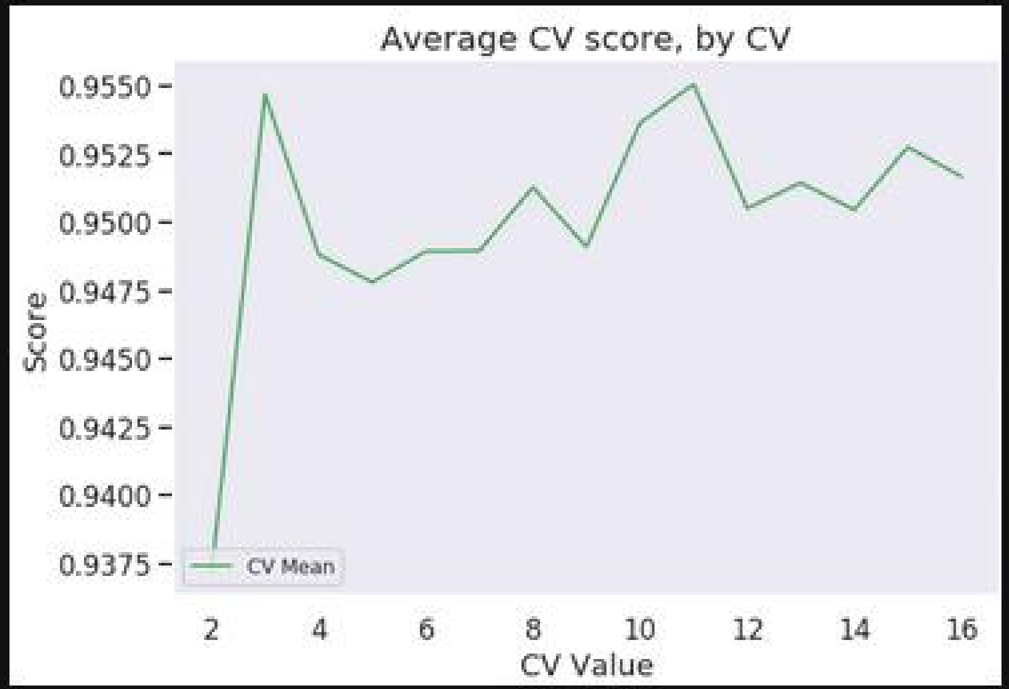
```
lrc_df.sort_values(by='f1',ascending=False).head()
```

	f1	recall	precision	solver	penalty	class	C	min_df
49	0.95	0.95	0.96	newton-cg	l2	multinomial	30	8
59	0.95	0.95	0.96	saga	l2	multinomial	30	8
39	0.95	0.95	0.96	lbfgs	l2	multinomial	30	8
96	0.94	0.93	0.95	lbfgs	l2	multinomial	17	10

Classification with Tf-idf Vectors Highest f1 score: .95
Logistic Regression

	0	1	2	3	4	5	6	7	8	9	10
AI	16	0	0	0	0	0	0	0	0	0	0
Algorithms	0	20	0	0	2	0	0	0	0	0	0
CS	1	2	27	0	0	0	0	0	0	1	0
Calculus	0	0	0	22	0	0	0	0	0	0	0
Data Structures	0	2	0	0	16	0	0	0	0	0	0
Diff. Eq.	0	0	0	1	0	22	0	0	0	0	0
Linear Algebra	0	0	0	0	0	0	52	0	0	0	0
Math for Eng.	0	0	0	0	0	0	2	8	0	0	0
NLP	0	0	0	0	0	0	0	0	4	0	0
Probability	0	0	0	0	0	0	0	0	0	36	0
Statistics	0	0	0	0	0	0	2	0	0	1	21

	precision	recall	f1-score	support
AI	0.94	1.00	0.97	16
Algorithms	0.83	0.91	0.87	22
CS	1.00	0.87	0.93	31
Calculus	0.96	1.00	0.98	22
Data Structures	0.89	0.89	0.89	18
Diff. Eq.	1.00	0.96	0.98	23
Linear Algebra	0.93	1.00	0.96	52
Math for Eng.	1.00	0.80	0.89	10
NLP	1.00	1.00	1.00	4
Probability	0.95	1.00	0.97	36
Statistics	1.00	0.88	0.93	24
micro avg	0.95	0.95	0.95	258
macro avg	0.95	0.94	0.94	258
weighted avg	0.95	0.95	$\boxed{0.95}$	258



Metrics to Compare

- Silhouette Score
- t-SNE Visualization
- Mean Similarity
- f1 score

Scrape The Data

Clean data

Doc2Vec model

Clustering

Silhouette Score*
t-SNE Visualization*

Cosine Similarity

Mean Similarity*

Classification

f1 score*

TF-idf model

Clustering

Silhouette Score*
t-SNE Visualization*

Cosine Similarity

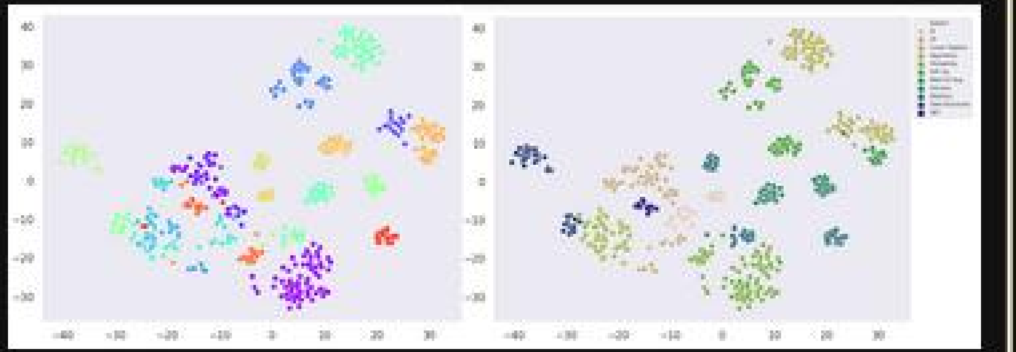
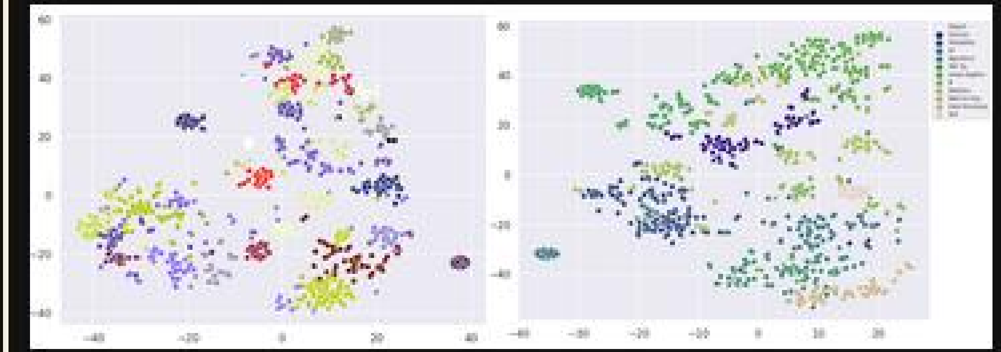
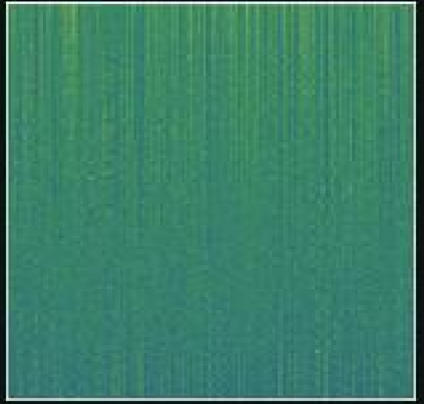
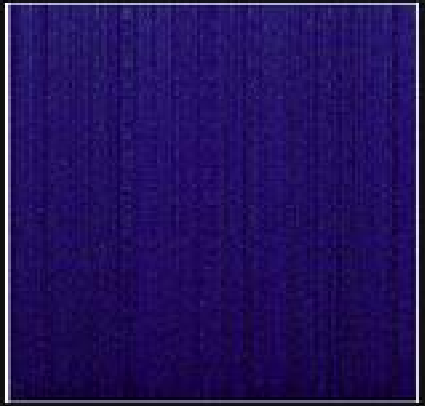
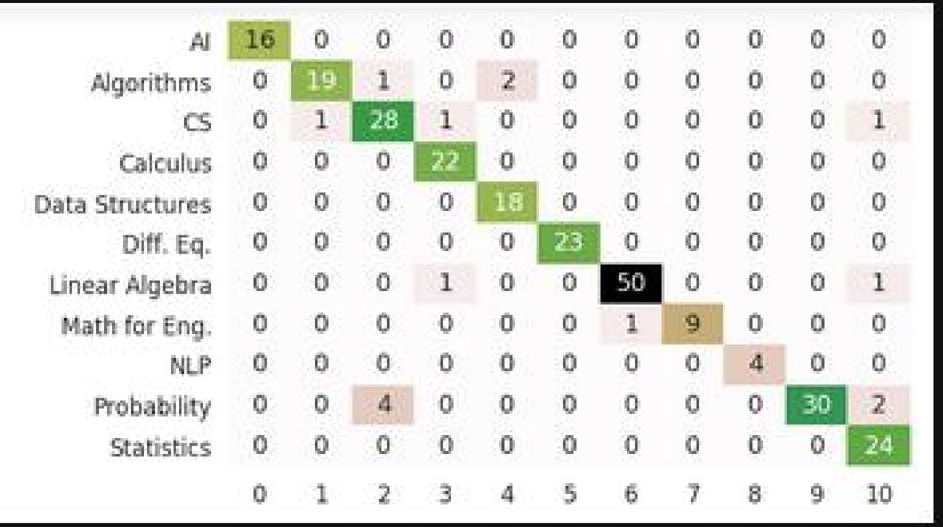
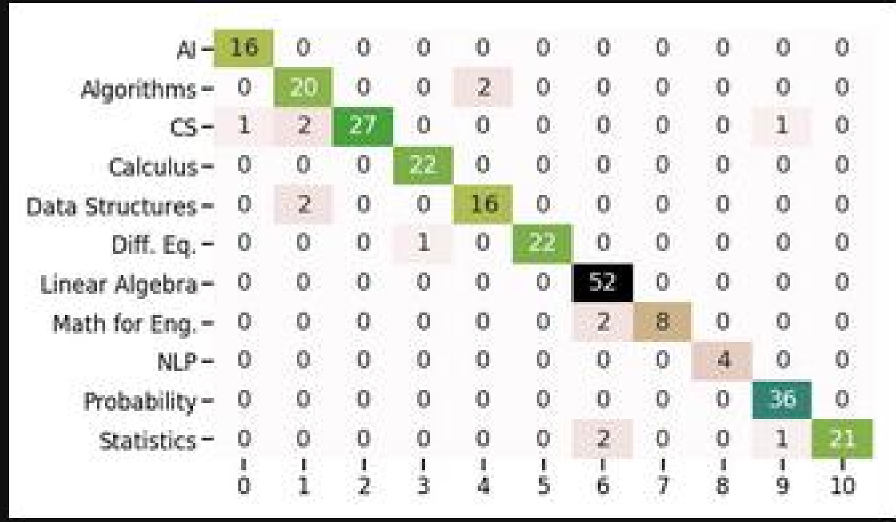
Mean Similarity*

Classification

f1 score*

KMeans	KMeans
.2998	.1800
.1445	.065
.94	.95

Compare

Doc2Vec	Compare	Tf-idf
	**Clustering**	
.2998	Silhouette Score	.1801
 Mean Similarity .1445	**Cosine Similarity** Mean Similarity	Mean Similarity .065 
	**Classification**	
.94	Avg f1	.95

Use Cases

Clustering can be used on unlabeled data. This can be useful for determining resource allocation for new tasks.

Example: *"We have thousands of unsorted emails in our info@ourco.com inbox. We want to see how we can most efficiently categorize these emails"*

Cosine Similarity is useful for identifying items are similar to each other.

Example: *"We want to deliver a premium experience for our users; we want a way to suggest additional products/items that our users are likely to consume."*

Text Classification is useful subject identification and document routing.

Example: *"We want a way to efficiently assign helpdesk tickets to the person who is best fit to answer"*

Conclusions

-Doc2Vec is better for clustering and establishing similarity

-Overall the tf-idf model outperformed the Doc2Vec model in the classification task.

Next Steps

-Collect more data

-Use a pre-trained word embedding as part of a deeper neural network to classify the texts

-Use LDA, NNMF and other advanced NLP techniques to improve scores.