

# NLP-based recommender system for stand-up comedians



Beautifulsoup



## EXECUTIVE SUMMARY

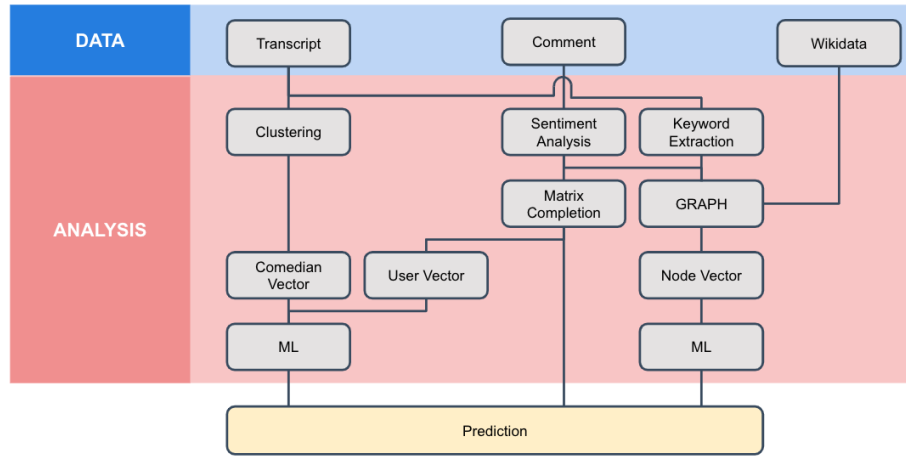


Fig 1. Roadmap of the project

### Problem Statement

Recommendation systems have been used widely these days, but ironically, in most of the cases, the performance of both the Collaborative Filtering (CF) and Content-Based Filtering (CBF) are not ideal. We found that the metric for evaluating the recommendation has long been neglected, constituting a bad experience on what we observed in YouTube recommendations: the recommended videos fail to capture our tastes. To test and improve the established methods (CF and CBF), we incorporate the transcripts and the YouTube comments to our recommender system, set up an objective evaluation metric, and come up with two more variations, hoping to suggest some comedians that fit our appetite well.

### Proposed Solutions

Three main methods to predict which comedians the users may like:

1. Vectorize comedians and users and use machine learning for prediction.
2. Practice classic CF.
3. Vectorize nodes in graphs and use machine learning for prediction.

### Key Findings

The classic CF only has 64% accuracy, meaning that there are rooms to improve and this approach has been overly accentuated. However, after encompassing other details into our recommender system, the best outcome of our proposed variations is no better than 70% accuracy, indicating that many details that we put into are noise. Therefore, further improvements could be made by studying more on how to extract more meaningful information from text and other resources.

## PROPOSED APPROACH

### CLASSIC COLLABORATIVE FILTERING


In the second branch of our roadmap, the collaborative filtering is conducted. First, the user comments are scraped from the comedian's video from YouTube. Secondly, The user comments are transformed into the polarity value by using the sentiment analysis with TextBlob. After having all the comments transformed, a sparse user-comedian matrix is formed as shown below. Afterwards, the soft singular value decomposition (SVD) is used for matrix completion. The original SVD can be decomposed into form:

$$A_{n \times p} = U_{n \times n} \cdot S_{n \times p} \cdot V_{p \times p}^T$$

The soft SVD is used to specify the number of latent spaces. The formulation of soft SVD:

$$A_{n \times p} = U \cdot \text{diag}[\sigma_1, \dots, \sigma_m] \cdot V^T$$

After training the factorized matrix, the user-comedian matrix can be completed.



	Ramy Youssef	Andy Woodhull	Amy Schumer	Arsenio Hall	Adi Assaf	Bert Kreischer	Bill Burr	Chris D'Elia	Chris Rock	Dave Chappelle	Ellie Simmonds	Paul Giamatti	Ramy Wood	Russell Peters	Sebastian Maniscalco
A.B	0.181429	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN
ADRIOTUN AKANDE	NAN	0.875000	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	-0.15	NAN	NAN
Anton Nym	NAN	0.680980	NAN	NAN	NAN	0.000000	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN
Lipika Wasekar- Morgan	NAN	-0.237500	NAN	NAN	NAN	0.480000	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN
Jennifer Bates	NAN	0.671675	NAN	NAN	0.0	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN
Cam'ronDestiny	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN
Jay Jay	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN
Cynthia Copland	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN
Comedy Central Stand-Up Series is Comedy Central Stand-Up	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN
Michelle McFarland	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN	NAN

	Ramy Youssef	Andy Woodhull	Amy Schumer	Arsenio Hall	Adi Assaf	Bert Kreischer	Bill Burr	Chris D'Elia	Chris Rock	Dave Chappelle	Ellie Simmonds	Paul Giamatti	Ramy Wood	Russell Peters	Sebastian Maniscalco
A.B	0.181429	0.000046	0.000085	-0.000044	0.000006	-0.000102	-0.000005	0.000076	0.000117	0.000185	...	-0.000053	0.000029	0.000	...
ADRIOTUN AKANDE	0.000031	0.875000	0.000083	-0.180061	0.000470	-0.000019	-0.000011	0.216019	0.000055	0.140067	...	-0.000022	0.000066	0.140	...
Anton Nym	0.000088	0.680980	0.000088	-0.000000	0.000180	0.000000	-0.000000	0.000000	0.000000	0.000000	...	-0.000000	0.000000	0.000000	...
Lipika Wasekar- Morgan	0.000000	-0.237500	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	...
Jennifer Bates	0.000000	0.671675	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	...
Cam'ronDestiny	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	...
Jay Jay	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	...
Cynthia Copland	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	...
Comedy Central Stand-Up Series is Comedy Central Stand-Up	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	...
Michelle McFarland	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	...

### HYBRID APPROACH

- Preprocess for the comedian vectors

Before we start to do the GMM clustering for the comedian vectors, the scraped transcripts need to have some preprocessed as shown in **Fig 2**. Firstly, we remove the special characters such as punctuation and emoji for the scraped transcript by regular expression and NLTK. Afterwards, we convert the clean data to TF-IDF matrix, and since it is a really huge matrix, we reduce the dimension of the matrix by PCA and ensure that it still maintains 90% of the information. Then, we have all the document vectors to do the clustering.

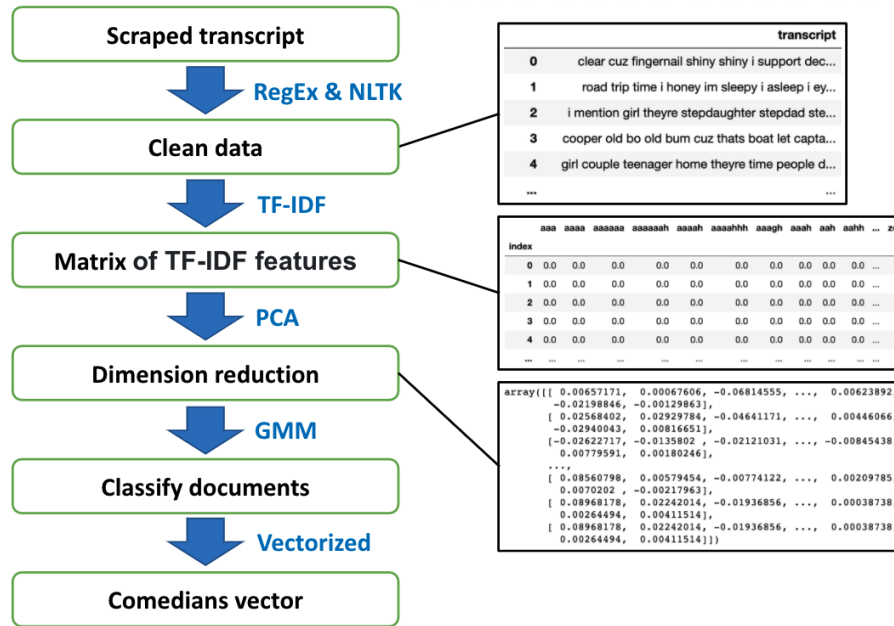


Fig 2. Workflow of vectorizing comedian vectors

Since it's an unsupervised learning problem. To find the optimal number of clusters, we apply AIC which is an information criteria that can determine which number of clusters is optimal. We can see that when the number of clusters is 4, the AIC is the lowest. So, based on this, we classify all the documents by GMM into 4 styles. Then we can group the documents by which comedian they belong to and calculate the probabilities that comedian might be for each style. Therefore, we can get the comedian vectors as shown in Fig 3.

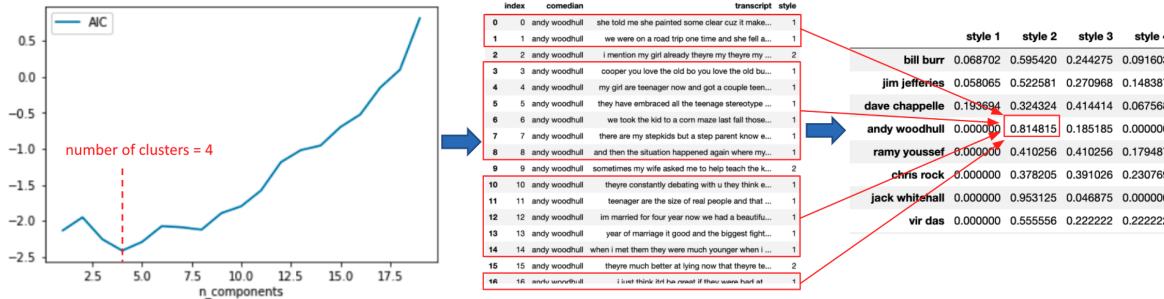


Fig 3. Result of GMM clustering and vectorized comedian vectors

- Training set of user vectors and comedian vectors

By combining the completed user-comedian matrix from CF and the comedian styles above, we can understand users' preference. The user vector is simply the percentage of the number of liked comedians (guess from completed matrix) in a certain style. For example, according to the completed matrix, user 0 likes no comedian in style 1 and thus has 0 for its prefer1 column.

	prefer1	prefer2	prefer3	prefer4	style1	style2	style3	style4	polarity
0	0.0	0.398008	0.407957	0.392965	0.0	0.300000	0.300000	0.400000	1.000000

Fig 4. Combining the user preference vectors and the comedian vectors..

## GRAPH APPROACH

- Keyword Extraction: See Appendix.
- Setup



**Fig 5.** (left) Schematic graph of train and test graph. (right) Snapshot of Neo4j setup.

To evaluate the result, the graph is built only with the training set and will make predictions on the potential link in the test set.

- Scenarios

Scenario 1 (left) contains the basic structure of the graph with unweighted links, while in scenario 2 (right), the link between users and comedians are weighted by the number of links. The weight of the link is represented as the number of the links between users and comedians. The number of the link =  $\text{Round}(\text{Rating}) + 1$ .

Based on scenario 2, scenario 3 clustered the users into 6 groups (explained in the next section) and added links from users to their belonging groups.

- Training set

y	Citizenship_Canada	Citizenship_India	Citizenship_New Zealand	...	u2	u3	c0	c1	c2	c3	k0	k1	k2	k3
0	0	0	0	0 ...	-0.648848	0.441313	0.169194	0.179598	0.158009	-0.084910	0.065133	-0.042955	0.039214	-0.039833
1	1	0	0	0 ...	0.110672	-0.014171	0.169194	0.179598	0.158009	-0.084910	0.065133	-0.042955	0.039214	-0.039833
2	0	0	0	0 ...	-0.884607	-0.132899	0.169194	0.179598	0.158009	-0.084910	0.065133	-0.042955	0.039214	-0.039833
3	0	0	0	0 ...	-0.573020	0.455453	0.169194	0.179598	0.158009	-0.084910	0.065133	-0.042955	0.039214	-0.039833
4	1	0	0	0 ...	-0.524527	-0.118715	0.169194	0.179598	0.158009	-0.084910	0.065133	-0.042955	0.039214	-0.039833
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7252	1	1	0	0 ...	0.247709	-0.615871	0.175699	-0.451615	-0.307100	-0.369318	-0.001913	-0.107575	-0.042244	-0.001469
7253	0	1	0	0 ...	0.495883	0.448396	0.175699	-0.451615	-0.307100	-0.369318	-0.001913	-0.107575	-0.042244	-0.001469
7254	0	1	0	0 ...	0.750049	0.147722	0.175699	-0.451615	-0.307100	-0.369318	-0.001913	-0.107575	-0.042244	-0.001469
7255	0	1	0	0 ...	0.906057	-0.235264	0.175699	-0.451615	-0.307100	-0.369318	-0.001913	-0.107575	-0.042244	-0.001469
7256	0	1	0	0 ...	-0.409075	0.076214	0.175699	-0.451615	-0.307100	-0.369318	-0.001913	-0.107575	-0.042244	-0.001469

7257 rows × 56 columns

**Fig 7.** Snapshot of final training data

## EXPERIMENTAL RESULTS

In all experiments, when the polarity value  $\geq (<)$  0.5 changes it to 1(0) means the user may like (dislike) this comedian.

### CLASSIC COLLABORATIVE FILTERING APPROACH

Since SVD can be used to fill in missed or unlabeled datas, by training the hyperparameters on training data, and the results on testing data are shown below. The precision is 0.707, recall is 0.846, and f1 is 0.771. Also, we have **64.793 % accuracy**.

**Tabel 2.** Result by using user matrix

	Actual true	Actual false
Predict true	1073	444
Predict false	195	103

### HYBRID APPROACH

- Models

```
('RD', DummyClassifier(strategy='uniform'))
('LR', make_pipeline(StandardScaler(), LogisticRegression(solver='liblinear'))))
('LDA', LinearDiscriminantAnalysis()))
('KNN', KNeighborsClassifier()))
('DT', DecisionTreeClassifier()))
('RF', RandomForestClassifier()))
('NB', GaussianNB()))
('SVM', make_pipeline(StandardScaler(), SVC(kernel='rbf', gamma='auto'))))
('XGB', make_pipeline(StandardScaler(), XGBClassifier()))
('MLP', MLPClassifier(hidden_layer_sizes=[100,100], activation='logistic'))
```

**Fig 8.** The Machine Learning models

- \* Dummy Classifier randomly predicts the result of 1 or 0 with equal probability.
- \* LR, SVM, and XGB need to standardize the data first.
- \* MLP is essentially a neural network. Here we use 2 hidden layers with 100 perceptrons in both layers and the activations are all using logistic function.
- The results:  
This cross validation is done under k-fold = 20.

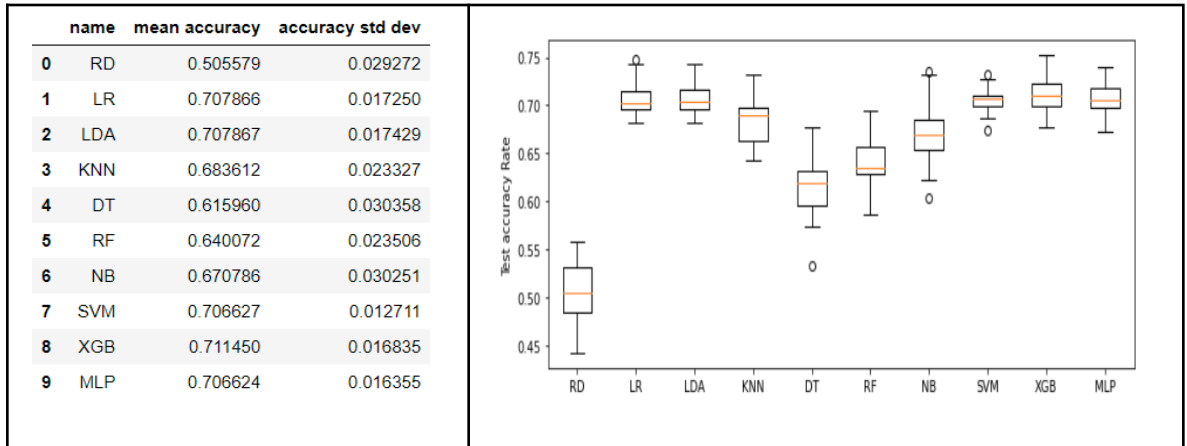


Fig 9. Results of all models

The results all suggest that XGB performs the best (highest mean accuracy and highest mode). Given this outcome, the hypothesis test is performed to see whether the differences between XGB and other models are statistically significant:

#### - Hypothesis Test

Null Hypothesis: XGB is not significantly different from others. Due to the fact that XGB is the best classifier, we can do some hyperparameter tuning and the result is not bad.

XGB vs. RD: p-value = 32.831192121883134	Precision = 0.712
XGB vs. LR: p-value = 0.800619410493323	Recall = 0.920
XGB vs. LDA: p-value = 0.7961202854491364	Accuracy = 0.6837465564738292
XGB vs. KNN: p-value = 5.211061704287136	
XGB vs. DT: p-value = 14.813421698326604	
XGB vs. RF: p-value = 13.294115115922919	
XGB vs. NB: p-value = 6.325253514648788	
XGB vs. SVM: p-value = 1.2311196224902137	
XGB vs. MLP: p-value = 1.1072653136814767	

	0	1
0	75	472
1	102	1166

Fig 10. (left)T-Test result (right) prediction result after tuning hyperparameter

## GRAPH-BASED MACHINE LEARNING APPROACH

#### - Models

Same as the hybrid approach.

#### - Cross Validation

This cross validation is done under k-fold = 20.

**Table 3.** The result of machine learning model on three scenarios

1

	name	mean accuracy	accuracy std dev
0	RD	0.501716	0.026720
1	LR	0.529140	0.163092
2	LDA	0.526109	0.160540
3	KNN	0.508921	0.138710
4	DT	0.485240	0.137126
5	RF	0.485653	0.144876
6	NB	0.551433	0.170227
7	SVM	0.532630	0.166122
8	XGB	0.553010	0.146412
9	MLP	0.601350	0.134692

2

	name	mean accuracy	accuracy std dev
0	RD	0.505038	0.024287
1	LR	0.537589	0.161160
2	LDA	0.517200	0.166518
3	KNN	0.518984	0.139476
4	DT	0.481926	0.129872
5	RF	0.487160	0.139302
6	NB	0.554605	0.168834
7	SVM	0.537037	0.161083
8	XGB	0.557698	0.148899
9	MLP	0.615681	0.119389

3

	name	mean accuracy	accuracy std dev
0	RD	0.496633	0.025212
1	LR	0.523221	0.161561
2	LDA	0.516070	0.169914
3	KNN	0.515949	0.134471
4	DT	0.489062	0.121776
5	RF	0.491289	0.142464
6	NB	0.556538	0.169146
7	SVM	0.539654	0.158534
8	XGB	0.565544	0.138323
9	MLP	0.622299	0.119662

The results all suggest that MLP performs the best (highest mean accuracy and highest mode). Given this outcome, the hypothesis test is performed to see whether the differences between MLP and other models are statistically significant.

#### - Hypothesis Test

Null Hypothesis: MLP is not significantly different from others.

According to **Table 4**, all three scenarios reject the null hypothesis. Therefore, we choose MLP for further hyperparameter tuning.



**Tabel 4.** Significant test of different machine learning models on three scenarios

MLP vs. RD: p-value = 3.162718363289611	MLP vs. RD: p-value = 3.958537504016811	MLP vs. RD: p-value = 4.479228845676908
MLP vs. LR: p-value = 1.488070388300277	MLP vs. LR: p-value = 1.6971934510500473	MLP vs. LR: p-value = 2.1480848450171974
MLP vs. LDA: p-value = 1.5650310797035645	MLP vs. LDA: p-value = 2.095083283977791	MLP vs. LDA: p-value = 2.2280678511660343
MLP vs. KNN: p-value = 2.083783112680877	MLP vs. KNN: p-value = 2.295772455383708	MLP vs. KNN: p-value = 2.5752972295512544
MLP vs. DT: p-value = 2.633100597316842	MLP vs. DT: p-value = 3.3049490580739964	MLP vs. DT: p-value = 3.4016617841633776
MLP vs. RF: p-value = 2.549408949768981	MLP vs. RF: p-value = 3.053533169434377	MLP vs. RF: p-value = 3.0693563473561385
MLP vs. NB: p-value = 1.0023538795134523	MLP vs. NB: p-value = 1.287483841817824	MLP vs. NB: p-value = 1.3834488048009166
MLP vs. SVM: p-value = 1.4006224277982722	MLP vs. SVM: p-value = 1.7097093731216204	MLP vs. SVM: p-value = 1.8136531540000236
MLP vs. XGB: p-value = 1.0591333973156518	MLP vs. XGB: p-value = 1.3242904308653385	MLP vs. XGB: p-value = 1.352598011417183

- Hyperparameter tuning (MLP)

The outcome of scenario 2 and 3 are similar, the comparison is made in the next section.

**Tabel 5.** Results of hyperparameter in three different scenarios

	Scenario 1	Scenario 2	Scenario 3																											
Train	<p>Precision = 0.780 Recall = 0.907 Accuracy = 0.758026732809701</p> <table><tr><td></td><td>0</td><td>1</td></tr><tr><td>0</td><td>935</td><td>1286</td></tr><tr><td>1</td><td>470</td><td>4566</td></tr></table>		0	1	0	935	1286	1	470	4566	<p>Precision = 0.697 Recall = 0.994 Accuracy = 0.6964310321069312</p> <table><tr><td></td><td>0</td><td>1</td></tr><tr><td>0</td><td>47</td><td>2174</td></tr><tr><td>1</td><td>29</td><td>5007</td></tr></table>		0	1	0	47	2174	1	29	5007	<p>Precision = 0.696 Recall = 0.996 Accuracy = 0.6954664461898856</p> <table><tr><td></td><td>0</td><td>1</td></tr><tr><td>0</td><td>29</td><td>2192</td></tr><tr><td>1</td><td>18</td><td>5018</td></tr></table>		0	1	0	29	2192	1	18	5018
	0	1																												
0	935	1286																												
1	470	4566																												
	0	1																												
0	47	2174																												
1	29	5007																												
	0	1																												
0	29	2192																												
1	18	5018																												
Test	<p>Precision = 0.714 Recall = 0.821 Accuracy = 0.6451790633608815</p> <table><tr><td></td><td>0</td><td>1</td></tr><tr><td>0</td><td>130</td><td>417</td></tr><tr><td>1</td><td>227</td><td>1041</td></tr></table>		0	1	0	130	417	1	227	1041	<p>Precision = 0.704 Recall = 0.994 Accuracy = 0.7035812672176308</p> <table><tr><td></td><td>0</td><td>1</td></tr><tr><td>0</td><td>17</td><td>530</td></tr><tr><td>1</td><td>8</td><td>1260</td></tr></table>		0	1	0	17	530	1	8	1260	<p>Precision = 0.702 Recall = 0.995 Accuracy = 0.7013774104683196</p> <table><tr><td></td><td>0</td><td>1</td></tr><tr><td>0</td><td>11</td><td>536</td></tr><tr><td>1</td><td>6</td><td>1262</td></tr></table>		0	1	0	11	536	1	6	1262
	0	1																												
0	130	417																												
1	227	1041																												
	0	1																												
0	17	530																												
1	8	1260																												
	0	1																												
0	11	536																												
1	6	1262																												

## **DISCUSSION**

### **METHOD COMPARISON**

The way we compare our results is simply calculating the ratio of different predicted values. Since “graph-scenario 2” has the best outcome, it will be the reference of the comparison.

	Percentage of different prediction on test set in comparison to” graph - scenario 2 “
Classic CF	15.64%
Hybrid	8.98%
Graph - Scenario 1	6.1%
Graph - Scenario 3	0.17%

The difference of predicted value between “graph - scenario 2” and “graph -scenario 3” is less than 1%, we thereby conclude that there is no significant difference between these two scenarios, while the result of “graph - scenario 2” is significantly different from others. **Therefore, the best outcome we could achieve in this project is around 70%.**

We could apply Monte-Carlo sampling on all methods to collect more accuracy results and assess the difference more accurately by conducting the hypothesis test, but it will take too much time to run.

## **CONCLUSIONS**

The best accuracy of the recommender system built in this project is about 70%, higher than the classic CF by some 6%. The result shows that the most important feature is user clustering, which goes hand in hand with the underlying assumption of CF: the rank of the user-comedian matrix should be low assuming that users can be categorized to limited groups. However, the graph result suggests that including the user groups explicitly in the graph does not affect the prediction, indicating that the vectorized nodes have already captured the nature of the user clusters, which actually is exactly what we want to observe.

## **FUTURE WORKS**

If the user data can be collected more in the future, we believe that the accuracy may be improved. In addition, since we focus on enhancing the accuracy of prediction, we only do some simple preprocess and use a textblob library to do sentiment analysis for the transcripts and comments. Therefore, if we can improve the methods of data cleansing and sentiment analysis in the future, the prediction accuracy is believed to be increased.

## **LESSON LEARNED**

More information will lead to more noise, so a recommendation system is not that easy as we thought in the first place. We tried our best to use what we have learned in class (such as embedding technique and knowledge graph) into practice and also apply some modeling techniques like FastMap, GMM and matrix completion. However, we can only improve the accuracy by 6%, which means there is still lots of noise. After finishing this project, we realized how important cleaning the data to extract useful information is, otherwise, the analysis would reduce to a garbage-in-garbage-out result.

Nonetheless, this project still gives us many chances to practice something we learned. We try to mine the data from Youtube, gather information from WikiData, build knowledge graphs to simulate relations and try multiple uncharted python packages to deal with different tasks. Those tools are core abilities when we get into future work or research.

## TEXT PREPROCESSING

All properprocessing that we have done:

```
# Lowercase every words
text = text.lower()
# Remove every words with [blah blah blah] format
text = re.sub('\[.*?\]', '', text)
# Remove every words with {blah blah blah} format
text = re.sub('\{.*?\}', '', text)
# Get rid of the punctuations
text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
# Get rid of all the numbers or words that contain numbers
text = re.sub('\w*\d\w*', '', text)
# Get rid of these specific punctuations
text = re.sub('[<img alt="three wavy lines" data-bbox="385 615 415 645"/>...]', '', text)
# Get rid of '\n'
text = re.sub('\n', '', text)

# Tokenizes and Lemmetizes (or stems) them
tokenized = word_tokenize(text)
stemmed = [porter_stemmer.stem(t) for t in tokenized]
```

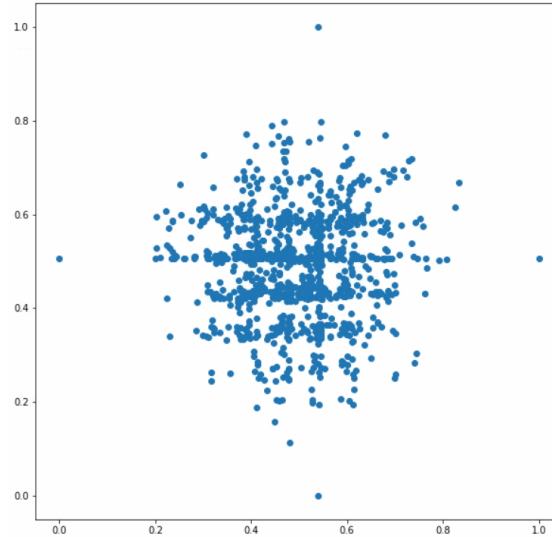
To extract the keywords of a comedian, we first append all the corresponding transcripts of that comedian. Then, after preprocessing the transcripts, we encode the transcripts to a data-term matrix. From this data-term matrix, we could find more common words that are used among comedians. We further remove those words that is used by more than 7 comedians (those words are mostly profanities or different forms of verbs) then use the “summa” library to extract keywords.

To extract the keywords of a comedian, we first append all the corresponding transcripts of that comedian. Then, after preprocessing the transcripts, we encode the transcripts to a data-term matrix. From this data-term matrix, we could find more common words that are used among comedians. We further remove those words that is used by more than 7 comedians (those words are mostly profanities or different forms of verbs) then use the “summa” library to extract keywords.

	Comedian	Keywords
0	amy schumer	grow, grows, arrested, schumers, schumer, buttholes, butthole, growing comforting, uti, popcorn, bleeding, bleed, actress, showed, hyperemesis, sh...
1	andy woodhull	toast, maze, punishing, punishment, ordered punish, dressing ate, parenting, dish, floss, tradition crane, rinsing, stepparent, sourced, source, d...
2	arsenio hall	wesley, marching, tiger, grow, plus, bernie, ambien, booger, ashy, tyson, dust, peed, nevada, semmi, kim jongun, hoe jose, offends, changing, arse
3	aziz ansari	randy, randys, harris, texted, writing, writes, ja, darwish, perform, performed, performer, performing, performance, exist, existing, existence, p...
4	bert kreischer	marshmallow, leeanan, ilas, pajama, glock, leeanas alexa, ralphs, stayed, owl, panicking, pouring whiskey, hardy, ralph sampson, deodorant, ringwor...
5	bill burr	questa, io, mai, po, tutto, quando, sei, tutti, nel, cose, ed os, giusto, persone colore, quello cui, penso ne, testa lei, freaked, nessuno, visto...
6	chris d'elia	anika, anikas, laid, gayer, inspirational, inspiring, inspired, staying peach, stayed, changing delia, invite, invitation, invited, dallas, grow, ...
7	chris rock	lm, lts, kobe, lt, pork, cooked, cook, witness, witnessing, tupac, cheated, bout, bully, bullying, loved custody, wealth, realised, toss, tossed, ...
8	dave chappelle	esos, unos, mujer, mujeres, blanco, esa, muchos, uno este, dio, dios, ellas, familia estaba, algo, mismo saben amigo, blancos sincero, ir, ese mom...
9	eric andre	salvia, bukkake, calvin, calvinism, eric, jessies, licked, licking, pus, jessie texted, snoop, atm, brian, tasted, texting, slap, arrested, scrub,...
10	george lopez	cabr pendejo, ay culo chinga rale, caca, latinas, latina, deported, deport, pendejos, tambi, viejo pedo culos, pedos, anderson, cacas chingado, en...
11	hannah gadsby	painted, douglas, fanny, nanette, hormonal, hormone, laughter, homosexuality, homosexual, picasso cubism ruin, feedback, unfortunate painting diag...
12	hasan minhaj	minhaj, jbs, prom, bhai, nia, nias, bethany, corrupt, guajajara, aisha, brazilian, slap, slapped, corruption bannon, correspondent, fox, crushing...
13	iliza shlesinger	mermaid, bachelorette, stacys, stacy, reject, rejected, rejecting, hollywood, planned, fried, fry, snootch, hunting, hunt, liner, witching, veil, ...
14	jack whitehall	chad, shouting, shouted, shout, royal, traveling, traveled, travel, trollfully, rolled, swimming, nearly, disney princess troll, introduced, intr...
15	jerry seinfeld	superman, poptart, aware, doin, looge, pushing, bout, kellogg poptarts, shaving, shave, compete push, meaningless, beep, glue, complicated, respon...
16	jim jefferies	disabled, disability, dans, vibrating, andrew, pushed, pushing, push, lived, pi, vomiting, vomit, vomited, prostitute, prostitution, involve, brot...
17	jimmy o. yang	representing, represent, soap, tai, haunt, confused, acting, organic, lamborghini, foam, foaming, nephew hong, julian, frustrated, frustrates, pin...

## USER CLUSTERING

Since the similarity of ratings between two users can be captured by the Cosine Similarity, and the ratio of commonly rated comedians of them can be captured by the Jaccard Similarity, **the user distance is defined as the Cosine Similarity times the Jaccard Similarity**. After defining the distance, a distance matrix is created and the FastMap algorithm is applied to encode the user vector.



**Fig.** The result of encoded user vectors.

Later on, KNN is used to find the clusters of the data. The best number of clusters is 6, according to the Silhouette Coefficient.



**Fig.** Silhouette Coefficient vs. number of clusters.

## FEATURE IMPORTANCE

To study the importance of each feature, we run the Random Forest Classifier on user grouping and all other comedian properties. It turns out that “Group” (the result of the “user clustering” section) is the most important feature of all. As for comedian features, comedians in different ages have different importance on prediction, as well as their marital status.

