Research Paper Recommendation Project

Data Science Applications DTI5125[EG]

Group name: DSA_202101_ 8

Abdallah Medhat Mohamed Rashed

Ahmed Fares Saad Eldin Khalifa

Amira EssamEldin Ibrahim Ibrahim Elgebaly

Amr Ashraf Mahmoud Elsherbiny

1. Objectives

The overall objective is to build a conversational recommender system that is able to recommend papers for a user based on similarity with a query paper. Moreover, it is also able to recommend recent publications in fields defined by the user. We use Google Dialog flow to build the chatbot interface.

2. Requirements

Using a dataset of arXiv dataset provided by Cornell university, build a recommender system that is able to search in this huge amount of data, and provide a recommendation for the user in an interactive manner. The user should provide data to help with recommendation, and any further needed information should be asked for by the chatbot. The results should be presented quickly because there is a time limit for answering in Google Dialog flow.

3. Methodology

3.1) Data Loading and Exploration

We start by loading dataset from Kaggle. The dataset contains more than 1.7 million publications, belonging to many specific research areas. In order to reduce our search space and avoid very rare labels, we restrict ourselves to only publications of math, physics, statistics, and computer science that were published after 2010. Note that publications can belong to one or many classes. Figure 1 shows a sample of the dataset.

	title	id	doi	categories	year	abstract	cs	math	phy	stat
0	Quantum Fluctuations Contribution to the Rando...	1001.0342	NaN	quant-ph	2010	It is shown, by considering the case of the ...	0	0	1	0
1	Counting irreducible polynomials over finite f...	1001.0409	NaN	math.HO math.NT	2010	C. F. Gauss discovered a beautiful formula f...	0	1	0	0
2	Adiabatic Creation of Atomic Squeezing in Dark...	1001.0445	10.1103/PhysRevA.82.012112	quant-ph	2010	We study the multipartite correlations of th...	0	0	1	0

Figure Sample from arXiv dataset

After loading the dataset and grouping the subcategories into the four aforementioned categories, we visualize the distribution of the data to better understand it as shown in figure 2. Note that according to arXiv category taxonomy, physics category contains the most number of subcategories, therefore after grouping all subcategories to the main category, we can see that physics has the most number of publications in the dataset.

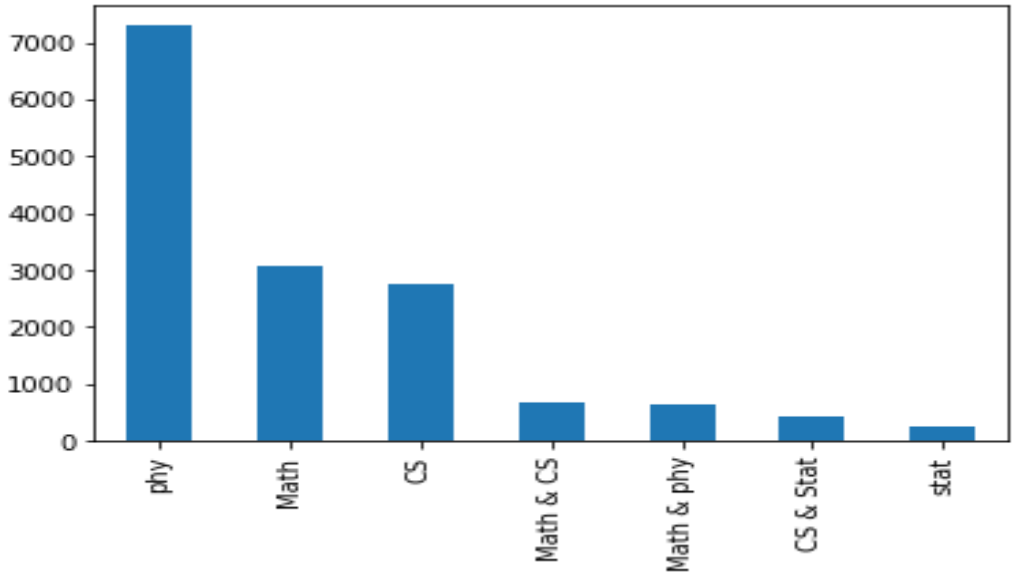


Figure 2: Dataset distribution

3.2) Data Preprocessing and Feature Extraction

In order to provide a recommendation, we need to extract meaningful feature from out dataset that enable us to clearly discriminate between different publications. To do so, we start by concatenating the title and abstract of each publication, then we remove stop words and perform stemming, and perform then extract TF-IDF and LDA features and concatenate them to get a single feature vector for each record. The resulting feature vector is 527 long. We also extract Doc2Vec features from the text and use it to compare with the aforementioned feature vector.

3.3) Modeling

After extracting the features, now comes the time for modeling. Our goal is to provide a recommendation as quickly as possible given a query abstract. An exhaustive search would be very time consuming, and it is impossible to incorporate this into a chatbot. Therefore, our first step towards reducing the search time is to perform classification and use the predicted labels to search only in publications with matching labels. Since the publications can have one or many classes, this is a multilabel classification problem. To tackle this problem, we try different strategies, discussed below. In order to evaluate our models, we use accuracy and hamming loss, because hamming loss gives partial score to partially correct predictions. Below, we discuss how we used 3 different models for classification.

3.3.1) Random Forest

Random forests are inherently able to solve multilabel classification problem since there is no restriction on the shape of the predicted vector. Therefore, we use it as our first strategy. Performing hyperparameter tuning did not have a strong impact on our random forest accuracy, therefore we stuck with the default parameters. The maximum accuracy that we were able to get was 82.5% with hamming loss of 0.06, with TF-IDF LDA combination.

3.3.2) Multi-output Classifier

A common approach to tackle multilabel classification is multi-output classifiers. It is used for models that do not support this operation by default, and the technique is very simple, it fits one classifier for every target label. We use it to train a K-nearest-neighbors model. After performing grid search to get the best K, and with Doc2Vec features, we got an accuracy of 67%.

3.3.2) Classifier Chains

A final approach we try is classifier chains. In classifier chains, multiple classifiers are trained to predict each label. However, unlike binary models, classifier chains incorporate knowledge from each other's predictions, in order to account for correlations and dependencies between labels. To train classifier chains, we use a logistic regression model. Logistic regression is significantly faster to train than random forest and has few parameters to tune. With this simple model and TFIDF LDA features, we got an accuracy of 83.2% and a hamming loss of 0.05. This was our top performing model.

3.3.4) Clustering

While our classifier chain with logistic regression gave reasonable accuracy given the present class imbalance, it still required a lot of time to present results. Dialog flow requires answers to be given within 5 seconds or the conversation will be terminated. Therefore, we need another reduction in our search space. To do so, we use K means clustering to cluster our dataset into 4 clusters using TFIDF LDA features. The idea is to predict the cluster id for every query abstract, and also perform classification on it. This differs from the previous approach in that we will only search within a cluster for publications with matching predicted label, therefore possibly reducing our search space. Figure 3 shows a visualization of our data when grouped into 4 clusters, after performing PCA dimensionality reduction. Figures 4-7 show the distribution of our labels within each cluster. To better understand the strategy, let's look at cluster 0 (fig 7), with majority of publications belonging to computer science. If we get a query abstract that has a predicted cluster of 1, and a predicted label of physics, we will only search in the physics publications within this cluster, which are less than 400 records, giving us a huge speedup. The increase may not be huge if our predicted class was computer science, but the overhead of predicting the cluster is minimal so we will end up with a similar performance to only classification.

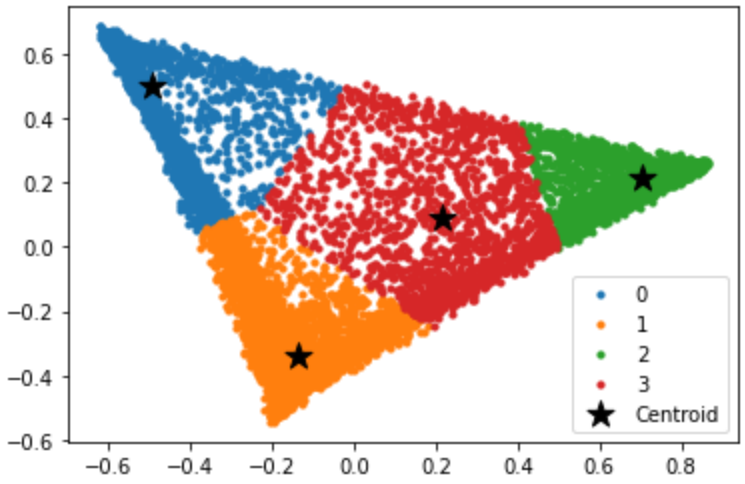
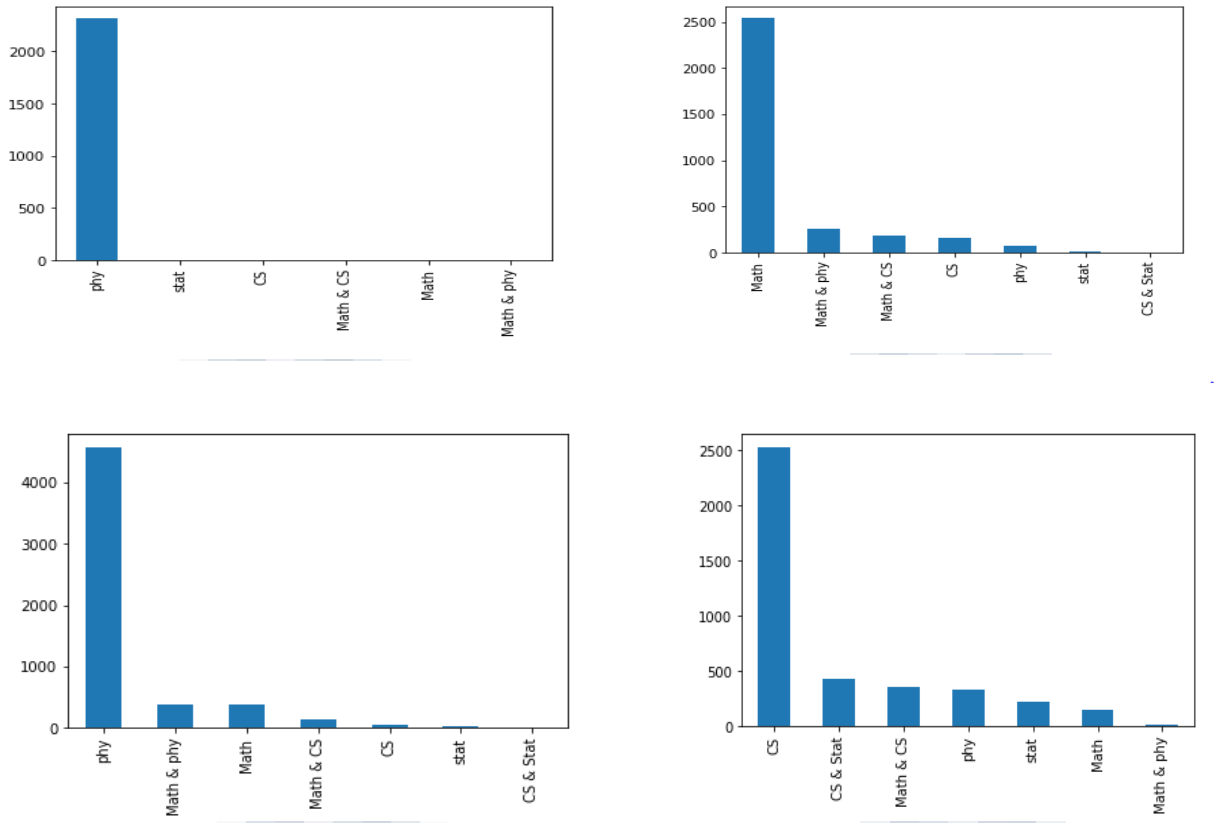


Figure 3 Data after K Means clustering



3.3.5) Champion Model

We found that combining TF-IDF and LDA actually helped improve the prediction capabilities, and chain classifier with logistic regression scores an accuracy of 83% on test data and hamming loss 0.058. We therefore choose the Chain classifier with logistic regression with LDA+TF-IDF to

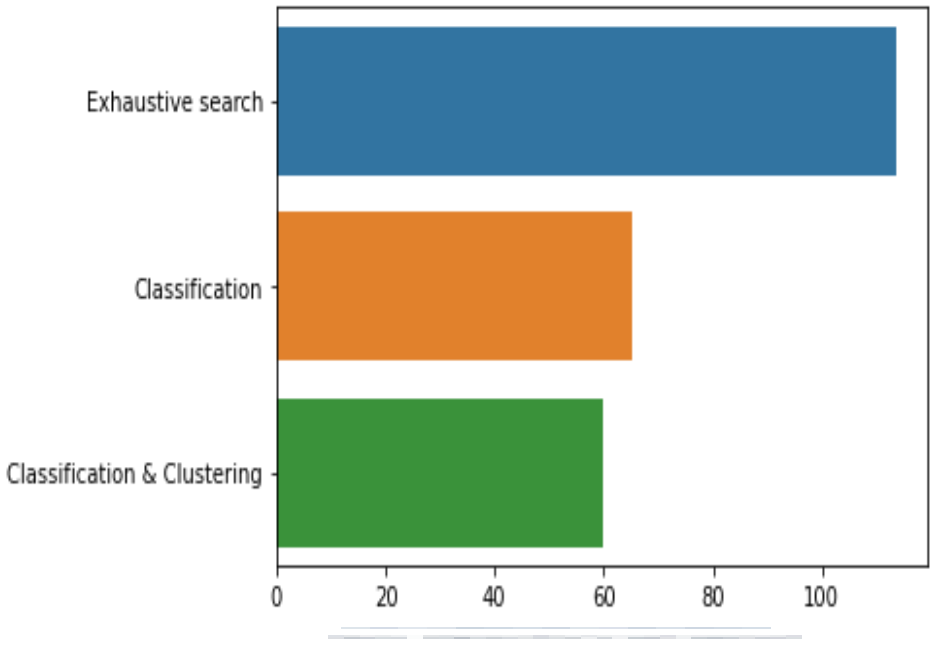
be out champion model. To summarize our results, table 1 contains the performances after classification and clustering.

	Random Forest (LDA TF-IDF)	Multi-output classifier KNN (Doc2Vec)	Chain classifier Logistic Regression (LDF TF-IDF)	k-means
Test accuracy	82.5%	67%	83.2%	83.2%
Hamming loss	0.06	0.12	0.05	0.057

Table 1: Summary of performances

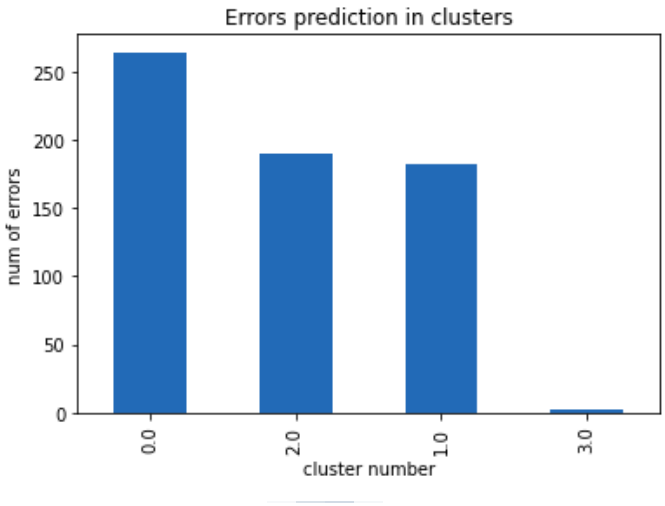
4. Results and Error Analysis

In this section, we discuss the performance of our recommendation system. The first thing that matters here, is how fast our recommendations are, and whether our classification, or classification and clustering helped improve the performance. To see this, we randomly sample 10 records from our dataset to avoid random lucky/unlucky single samples and use 3 different approaches to provide recommendations. In the first one, we use exhaustive search where we check similarity between the query and all our records. Obviously, this is extremely time consuming. In the second approach, we perform only classification on the query, and search in the publications that match the predicted label. Finally, we perform classification and clustering, and search only in the records that match the cluster id and the predicted label.

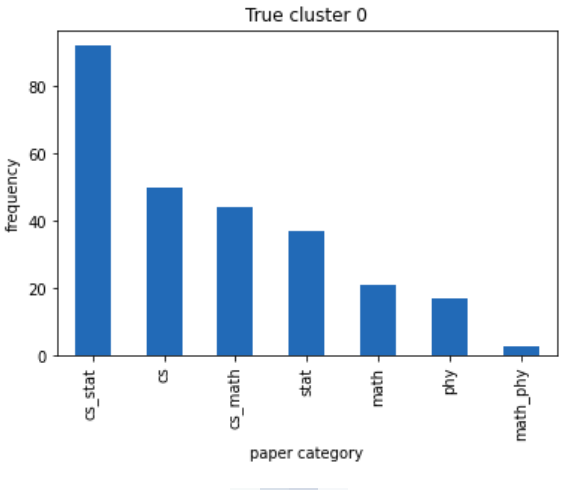


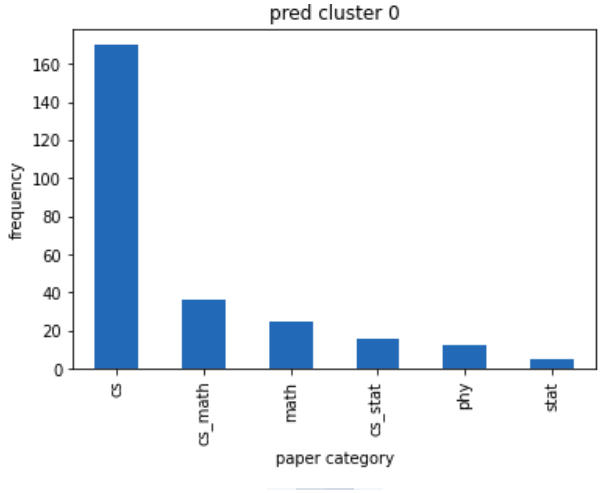
We can see above that classification and clustering take the least time, averaging 5.9 seconds per recommendation. It is followed by classification only, which averages 6.5 seconds per recommendation.

In this paragraph, we try to figure out the reasons for misclassification of some papers. First we understand error distribution as illustrated in table 1. By plotting each cluster against frequency of error examples, we found that most errors belong to cluster 0 in contrast with cluster 3 which has the lowest number of errors [figure 9].



To deeply understand the nature of cluster 0, we plot the category distributions of cluster 0 with respect to true categories as illustrated in figure 10 and predicted categories as illustrated in figure 11. We found that although CS_stat represents the majority of true categories, CS represents the majority in predicted categories which leads us to inquire about the model ability to distinguish between CS_stat papers and CS papers.





2 shows each original category and corresponding prediction from which we found that about 82 cs_stat papers,44 cs_math ,14 math, 17 physics and 22 stats are classified as CS. This means

that model faces difficulties in distinguishing between CS and other categories. It may return to the nature of the CS field as it is a disciplinary field that intersects with multiple fields, and this explains being the major category in prediction.

cat	pred_cat	
cs	cs_math	29
	cs_stat	9
	math	79
	math_phy	1
	phy	17
cs_math	cs	44
	math	48
	phy	5
cs_stat	cs	82
	cs_math	7
	math	5
	stat	5
math	cs	14
	cs_math	12
	math_phy	22
	phy	32

cat	pred_cat	
math_phy	cs_math	2
	math	49
	phy	83
phy	cs	17
	cs_math	2
	math	16
	math_phy	16
stat	cs	22
	cs_math	1
	cs_stat	7
	math	6
	phy	7

Table 2

For a more general overview, The distribution of each cluster with respect to true categories is provided in figure 12 and predicted categories are provided in figure 13.

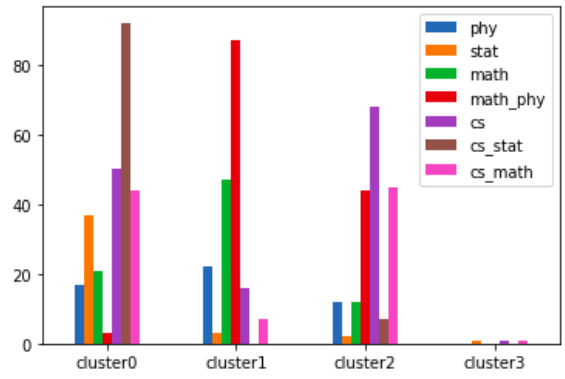


Figure 12

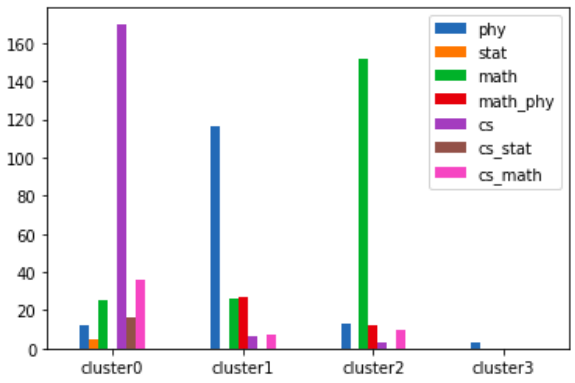
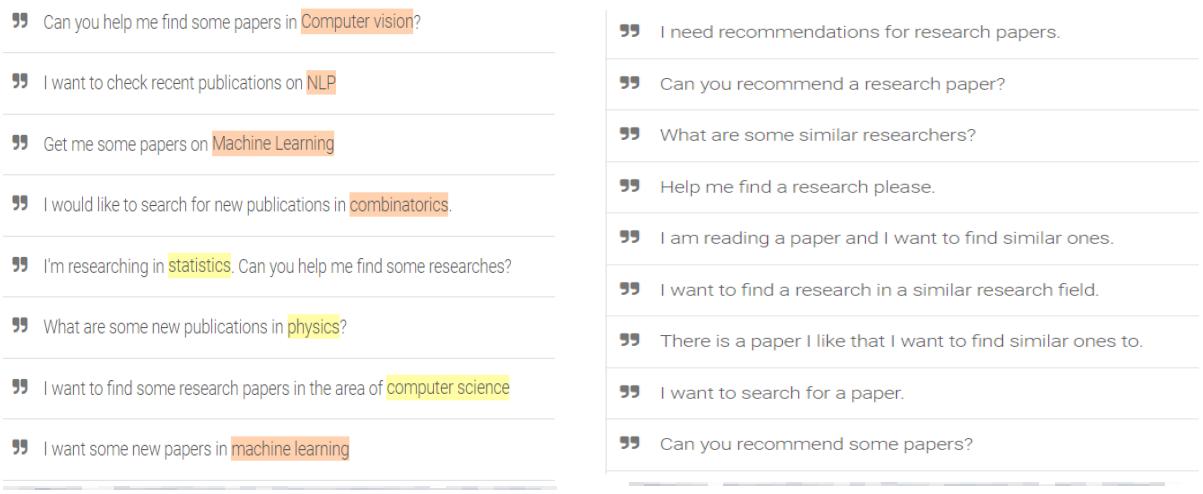


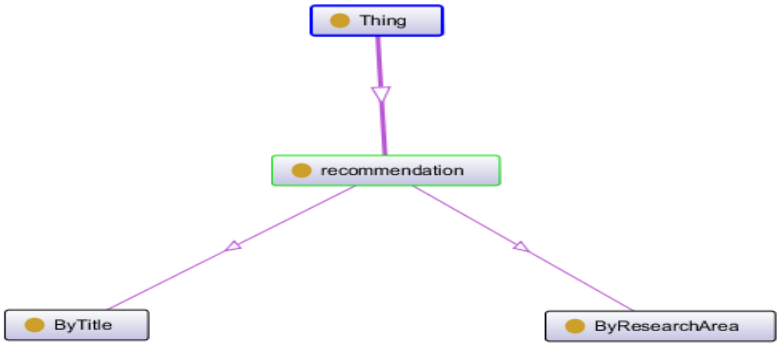
Figure 13

We observe that the math_phy represents the majority in cluster1 while phy represents the major category in cluster 1 predictions which indicates that the model is confused by phy and math_phy and shows that 83 math_phy is classified as phy. As a result, the model faces some problems recognizing papers with multi-label and most errors return to classify paper as one category and neglect the other category. For improving the model performance, we may use deep learning methods to be able to capture the detailed differences.

5. Chatbot Development

After building the backend of our recommendation system, now comes the time for the front end, or the chatbot interface. We develop a chatbot using Google Dialog flow, with two main intents and several follow up intents. The two main intents are find_research_cat for searching by category, and find_research_sim for searching by a similar paper, where the user will be asked to provide a title. Using this title, we will search for the publication in our dataset, use its pre-extracted features, and perform our search and recommendation. We define two main entities for our chatbot, namely research-category and research-sub-category, where we provide the chatbot with examples of main research areas and subfields so that it can be able to extract it from the text. Figure 9 shows a sample of questions in find_research_cat intent, and figure 10 shows questions from find_research_sim intent. Finally, figure 10 shows the ontology of our chatbot.





Finally, for deployment of the chatbot, we develop a server using flask to enable the chatbot to connect with the developed backend models. We enable fulfillment for our intents, and our requests are sent to the flask server which uses the models and the dataset to provide recommendation.