

Rochester Institute of Technology
B. Thomas Golisano College Of
Computing and Information Sciences

Master of Science in Information Sciences and Technologies

~ Project Approval Form ~

Student Name: Rima Kandalgaonkar

Project Title: Automatic Tag Suggestion For Stackoverflow Questions Using
Machine Learning

Project Area(s):	Application Dev.	Database	Website Dev.
(√ primary area)	Game Design	HCI	eLearning
	Networking	Project Mngt.	Software Dev.
	Multimedia	System Admin.	<input checked="" type="checkbox"/> Informatics
	Geospatial	Visual Analytics	

~ MS Project Committee ~

Name	Signature	Date
------	-----------	------

Erik Golen

Chair

John-Paul Takats

Committee Member

Automatic Tag Suggestion For Stackoverflow Questions Using Machine Learning

By: Rima Kandalgaonkar

Project submitted in partial fulfillment of the requirements for the
degree of Master of Science in **Information Sciences and
Technologies**

Rochester Institute of Technology

B. Thomas Golisano College of

Computing and Information Sciences

Department of Information Sciences and Technologies

07-Nov-2020

Table of Content

List of Figures	4
List of Tables	4
Abstract	5
1. Introduction	6
2. Literature Review	8
3. Data Exploration and Preparation	9
3.1 Dataset	9
3.2 Dataset Exploration	10
3.3 Data Preprocessing and Cleaning	14
3.4 Feature Extraction	15
4. Solution Approach and Results	15
4.1 LDA	15
4.1.1 Data Preprocessing for LDA	16
4.1.2 LDA Implementation	16
4.1.3 LDA Evaluation	17
4.1.4 LDA Optimization	19
4.2 Classification	24
4.2.1 Data Preprocessing for Classification	24
4.2.2 ML Algorithm Implementation	25
4.2.3 ML Model Evaluation	26
4.2.4 Performance Tuning	27
5. Conclusion	32
Reference	33
Appendix	34

List of Figures

- Figure 1. The form to enter question in stackoverflow (taken from stackoverflow.com)
- Figure 2. World cloud for the most used tags for tags.csv file
- Figure 3. Ratio of missing values per column from merged dataset
- Figure 4. Most common tags vs count
- Figure 5. Number of scores vs number of questions
- Figure 6. Number of tags per question
- Figure 7. Number of answers per question
- Figure 8. Time vs number of questions reported
- Figure 9. Gensim LDA generated output
- Figure 10. Interactive chart for the topics extracted using LDA
- Figure 11. Number of topics vs coherence score
- Figure 12. LDA model with optimised parameters
- Figure 13. LDA output topics with word probabilities
- Figure 14. Actual vs predicted Tags for optimized LDA model with unseen input text
- Figure 15. Parameter settings for TfidfVectorizer
- Figure 16. GridSearchCV parameter tuning output
- Figure 17. Best estimated parameters for linearSVC using GridSearchCV
- Figure 18. GridSearchCV results for LinearSVC model
- Figure 19. AUC-ROC curve for LinearSVC model
- Figure 20. Number of questions vs Tags for test and predicted data
- Figure 21. Number of questions vs Number of Tags for test and predicted data

List of Tables

- Table 1. The data fields in Question.csv
- Table 2. The data fields in Answers.csv
- Table 3. The data fields in Tags.csv
- Table 4. Manually recognised topics and labeled accordingly
- Table 5. Coherence score for different values of eta and alpha
- Table 6. Topics obtained from LDA after parameter tuning
- Table 7. Performance metrics for ML algorithms
- Table 8. Performance metrics for ML algorithms with ClassifierChain
- Table 9. Performance metrics for LinearSVC after parameter tuning

Abstract

The stack overflow requires users to manually determine relevant tags for the entered question. It is a time-consuming process and requires a wide knowledge of the topic to identify correct tags to direct the questions to the experts for quick resolution. The goal of this study is to develop a model that will suggest appropriate tags for the entered context in the question for achieving better classification of the Tags on stackoverflow. Two approaches are proposed to develop automatic tag suggestion systems. In the first approach, an unsupervised learning algorithm-Latent Dirichlet Allocation (LDA) is implemented to generate topics from stackoverflow question dataset. The second approach is supervised classification model implementation for tag suggestion. Data exploration, analysis and preprocessing are performed to understand the dataset better and produce good quality data for modelling. The LDA model is evaluated and optimized based on hyperparameter and coherence score which represents interpretability of the generated model. For classification, logistic regression, random forest, Support Vector Machine (SVM) and Linear Support Vector Classification (SVC) are used to discover the best suited model. The model evaluation is performed using accuracy, precision, recall, F-1 score, hamming loss and Area Under Curve-Receiver Operating Characteristics (AUC-ROC) curve. The optimization and parameter tuning of LinearSVC generated based results for tag prediction using classification.

Keywords: topic modelling, classification, multilable, tag prediction, LDA, Logistic Regression, Random Forest, SVM, LinearSVC

1. Introduction

Question and answer (Q&A) platforms play a very important role in everyone's life. These platforms are the places over the internet where people can gain knowledge about everything by posting questions, retrieving relevant information, answering the posted questions etc. There are thousands of websites which provide Q&A platforms to gain answers and knowledge to the world. There are various technical websites/ forums which contain millions of Q&A such as Quora, Instructables, stackoverflow. These are few examples of platforms that are helpful for hobbyists, coders and developers to get help and find valuable resources. The categorization of Q&A is one of the most important elements to retrieve or ask for relevant information in an efficient way. The coding Q&A platforms such as stackoverflow uses tags to categorize questions and related information [1]. Figure 1. Shows form to ask questions for the relevant solution from stackoverflow. As shown in Figure 1. tag represents the type or the domain of the question which has to be entered accurately to get an answer to the question.

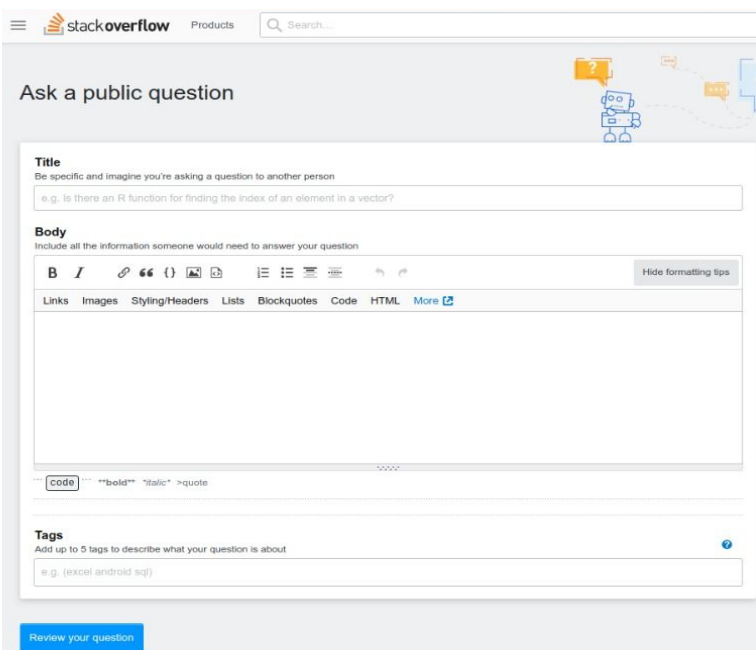
The image shows the 'Ask a public question' form on the Stack Overflow website. At the top, there is a navigation bar with the Stack Overflow logo, 'Products', and a search bar. Below the navigation bar, the page title is 'Ask a public question'. The form itself is divided into three main sections: 'Title', 'Body', and 'Tags'. The 'Title' section has a text input field with a placeholder example: 'e.g. Is there an R function for finding the index of an element in a vector?'. The 'Body' section includes a rich text editor with various formatting options like bold, italic, link, image, and code. Below the editor, there are links for 'Links', 'Images', 'Styling/Headers', 'Lists', 'Blockquotes', 'Code', and 'HTML'. The 'Tags' section has a text input field with a placeholder example: 'e.g. (excel android sql)'. At the bottom of the form, there is a blue button labeled 'Review your question'.

Figure 1. The form to enter question in stackoverflow (taken from stackoverflow.com)

The tags have a huge importance in the Q&A platforms as it states topics in the information or question and properly defined tags are easy to direct questions to the expert and active users which ensures quick resolution. Tags can also be useful to find similar types of information that is relevant to the topic. Currently, the process of entering Q&A involves manual tagging, and the user has to determine proper tags for the content which is getting entered. It is time-consuming and requires a wide knowledge of the topics to identify correct tags before entering. The auto

tagging approach can help users to solve mentioned problems by providing various tags related to the entered context. To make this process more friendly and improve the efficiency of the tagging process, a tag suggestion system is implemented using machine learning (ML) algorithms for the dataset of stackoverflow.

This document consists of various details of implementation of the auto-tagging system. It covers details such as Literature Review (existing solutions and their findings), Data Exploration and preparation, Solution approach and conclusion.

Problem Statement: Manual tagging while entering questions and related information on stackoverflow is time-consuming and needs wide knowledge of the topic for quick and relevant resolution. The stackoverflow is a valuable resource for coders and software engineers and the information related to Q&A is categorized using various tags to make it easily accessible. Currently, while entering Q&A users have to determine relevant tags for the entered question and the process is manual which takes more time and requires domain knowledge. The auto tagging approach can help users to solve mentioned problems by providing various tags related to the entered context. The stackoverflow uses defined tags to direct questions to expert and active users which ensures quick resolution [1].

Project Objectives: The goal of this project is to develop a model that will suggest appropriate tags for the entered context in the question for achieving better classification of the content on stackoverflow.

- Data Extraction, exploration and data cleaning to understand the data, the quality of questions and the Tags using various visualizations. Organization of data in consistent format, feature selection for ML algorithm implementation.
- Implementing machine learning approaches such as topic modelling and classification and identifying the best suitable algorithms to handle user input questions on stack overflow for the Tag suggestion system.
- Performance evaluation through standard metrics and optimize performance by parameter tuning for the trained models for efficient tag recommendation.

The report is organised as follows, section 2 contains literature review which elaborates prior work done in the area of tagging and data analysis using various ML techniques. Section 3 describes the data exploration and preparation including data cleaning, text pre processing and feature extraction. Section 4 focuses on two ML approaches, topic modeling using LDA and classification using Logistics Regression, Random Forest, Support vector Machine, Linear Support Vector Classifier. It includes details regarding its implementation, evaluation, results and performance optimization. Section 6 includes results and conclusions derived from this project.

2. Literature Review

This section highlights prior work done in the area of tagging and analysis of the data using various ML techniques. The approaches mentioned implement various methods to handle textual information, its preprocessing and to improve performance of the system.

In [2], Smrithi et al. have proposed a hybrid auto-tagging methodology to develop an auto-tagging system. Their system is capable of detecting programming languages and performing classification for questions. The code snippets are used to detect programming languages and various data features such as tokenization and pattern matching are used to train Multinomial Naive Bayes Classifiers for identification of the programming languages. The linear Support Vector Machine (SVM) classifier model is used to predict the tags from the content of the question. Various feature vectors built from training examples are used to train SVM classifiers. By combining these two approaches they have achieved 72% of accuracy for the auto-tagging system.

In [3], Alrashedy et al. have implemented a programming language prediction system using Natural Language Processing. For implementation, they have used two ML algorithms Random Forest Classifier and the XGBoost in the sense of achieving better accuracy. In this paper authors have focused on various questions to predict the programming languages by using a combination of two input parameters textual information (questions) and code snippets. They have achieved 91.1% accuracy by combining textual information and code snippet to train and test the classifier. Whereas for training and testing the classification model by making use of one of the input parameters from textual information or code snippets leads to the system accuracy of 81.1% and 77.7% respectively.

Logan et al. [4] have proposed tag recommendation models for stackoverflow by introducing an algorithm named NetTagCombine. The NetTagCombine is an extension to the currently available ML algorithms for Q&A tag recommendation systems. They have added a new component called Network-Based Ranking to the existing algorithms. The Network-Based Ranking consists of the structure of the networks in the stackoverflow data which is used to obtain additional information related to the post. The addition of this new parameter to the existing tag recommendation algorithm has achieved better tag recommendation accuracy. Their study reveals that network information and analysis can be beneficial to obtain data for the resemblance present in the content of websites.

Miltiadis et al. [5] have proposed a method to analyze stackoverflow questions to find topic, type and code related to the textual information. They have used topic modeling to find the topics. The Latent Dirichlet Allocation (LDA) with MALLET is used to train the textual information

and find text topics. Various natural language processing techniques are used for data processing before applying it to the model. By using LDA authors have answered various questions, such as variation in the question types and its dependency on programming languages, question type dependency on programming constructs and identifiers etc.

In [6], Michael Byrne has implemented a hashtag predicting system using Bayesian probabilistic models. The model which is used for hashtag prediction is based on ACT-R's declarative memory retrieval. In the implementation of methodology, the model is trained using $\frac{2}{3}$ of the dataset to calculate the strength of dependency between post information and tags. Logistic regression is used for calibration of the parameters and for performance optimization. The classification accuracy of 65% is achieved with this implementation.

In [7], Z. Yue et al. have proposed a recommendation system called End-to-end Tag-based Recommendation System (ETRS) for verbal reasoning question dataset. In this paper, Natural Language Processing (NLP) tools are used to perform analysis on textual information and extract the keywords as tags. To improve user satisfaction about using this system they have addressed three important issues, new items, new users and new systems. They have suggested an accuracy matching approach to add tags for users and improve the overall performance of the system. The paper presents results and effectiveness of this system as compared to item based or user based recommendation systems as it does not require huge data for better performance.

For this study both supervised and unsupervised approaches are explored and considered for implementation and analyze the quality of tags suggested. All the aspects of the dataset such as Title, questions body and Code snippets are considered for the study as each element has its own unique weight which tells about the Tags whereas many researches focus on only code snippets to detect programming languages. This study not only focuses on implementation of various ML algorithms, but also emphasises on performance evaluation and optimization in depth.

3. Data Exploration and Preparation

3.1 Dataset

For the implementation of automatic tag suggestion system, dataset is considered with specific type and format which should include details such as specific Id for each question entered, textual information in the form of Q&A, predefined tags and the timestamps. Each detail has its own importance in the dataset such as predefined tags with the body (textual information such as Q&A is important for training different ML models, time stamps are important for data exploration etc. The dataset considered for this implementation consists of stackoverflow Q&A.

The data is collected from Kaggle.com[8]. It includes three CSV formatted files: Questions.csv, Answers.csv and Tags.csv.

Table 1. The data fields in Question.csv

Field Name	Description	Details
Id	A unique id assigned to each question	A numerical value
OwnerUserId	A unique id assigned to each user	A numerical value
CreationDate	Question creation date	The data between 1Aug2008 to 19Oct2016
ClosedDate	Question closed date	The data between 1Aug2008 to 19Oct2016
Score	Usefulness of question	Score in the range -42 to 4718
Title	Question title	Text data
Body	Question body	Text data

Table 2. The data fields in Answers.csv

Field Name	Details	Variable Type
Id	A unique id assigned to each question	A numerical value
OwnerUserId	A unique id assigned to each user	468798 unique values
CreationDate	Question creation date	The data between 1Aug2008 to 19Oct2016
ParentId	Parent question id	A numerical value
Score	Usefulness of answer	Score in the range -42 to 4718
Body	Question body	Text data

Table 3. The data fields in Tags.csv

Field Name	Details	Variable Type
Id	A unique id assigned to each question	A numerical value
Tag	Tags assigned to each question	14883 unique tags

The features used for implementation of automatic tag suggestions are Title, Body, Tags, and Score. The dataset contains three files and the fields are mentioned in Table 1, 2 and 3 Data contains 1264204 questions from 630908 users with 2014375 number of answers. The file Tags.csv includes 37034 number of tags with assigned question Id.

3.2 Data Exploration

Data exploration is a crucial step in any data analytics project. It is an initial investigation over data and it helps to understand it in detail. Data visualization is an effective way to present and understand data in a graphical format by choosing the right visuals such as graphs, pie charts, maps etc. Figure 2 represents word cloud for most used 50 tags.

- Figure 4. shows top 50 tags and its frequency to get an idea about the tags from the dataset. The C# tag frequency is ~ 7000 and XML tag frequency is ~ 500. The most of the questions which are available in the dataset are related to C#, java tags.

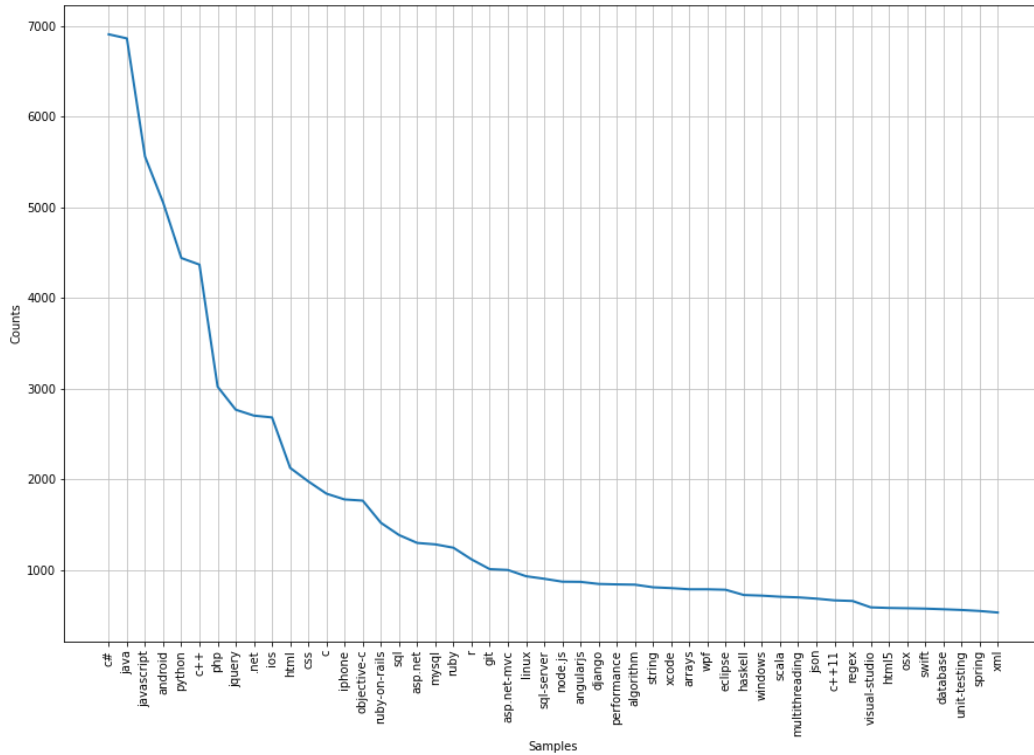


Figure 4. Most common tags vs count

- The score column can be used to extract good quality questions and answers from the dataset. The figure 5. shows the number of questions assigned with particular score values. For example, more than 500000 questions have a score of 0.

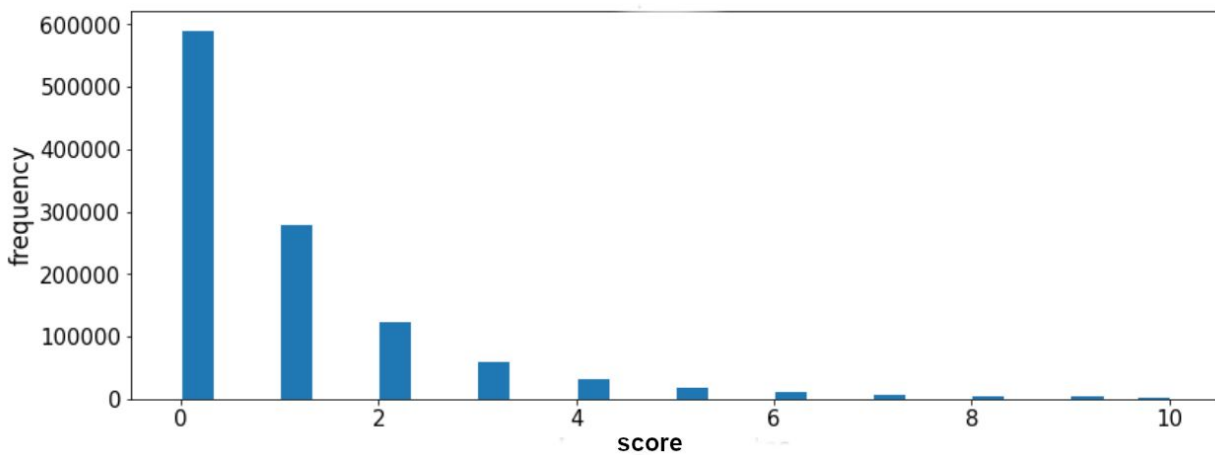


Figure 5. Number of scores vs number of questions

- There are multiple tags assigned to each question. In figure 6. We can see there are more than 350000 questions which have 3 tags assigned to them. We can use this data to find tags vs questions dependency and how particular a tag represents to that question.

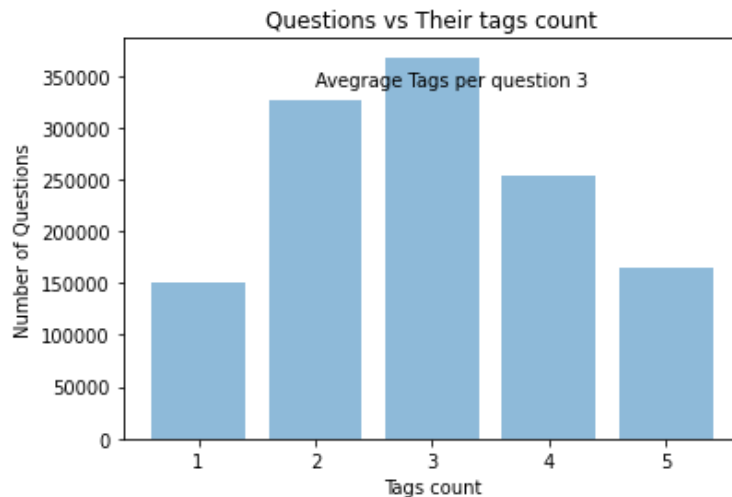


Figure 6. Number of tags per question

- The figure 7. shows relation between questions and answers and displays question vs answer count. On average each question has 2 answers.

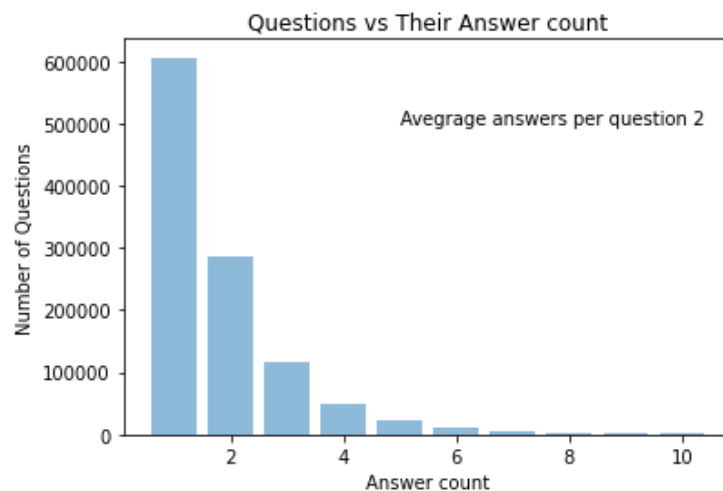


Figure 7. Number of answers per question

- Timing analysis is also important to understand the number of questions reported vs time. This analysis can also help us to understand the specific tags which are used over the time, increase or decrease in the use of specific tags. Figure 8. shows the number of questions/ Title reported over the time.



Figure 8. Time vs number of questions reported

3.3 Data Preprocessing and Cleaning

The dataset contains three files and the fields are mentioned in Table 1,2,3. Data contains 1264204 questions from 630908 users with 2014375 number of answers. The file Tags.csv includes 37034 number of tags with assigned question Id. For implementation of various ML algorithms, the raw data has to go through various data cleaning and preprocessing processes. Question body and title has various unwanted characters such as <p>, </p>\n\n<p> etc. Those are html tags and are removed for further data processing with the help of python library. The text in the Body and Title column are standardized by converting text into lower case, converting short form to full forms (I've > I have), removing scripting related commands (\n, \s etc.), removing blank spaces and other special characters. Performed Tokenization on the Body column data. Tokenization helped to separate each word and can be used to perform further cleaning individually. Furthermore all the punctuations such as (!"#\$\$%&\'()*+,-./:;<=>?@[\\]^_`{|}~) are removed from the Body column using python libraries.

In the preprocessing include stop word removal, stemming, and part-of-speech tagging tasks. The stop word removal process is the most commonly used preprocessing process to remove words such as a, the, an, as, and, etc. Stop words carry little semantic value and are less useful for information retrieval and therefore removing helps in querying better and saves processing time while searching. Stop words are removed using python NLTK (natural language toolkit) library.

Stemming is used to convert various terms to their root form before indexing. It is a technique used to reduce feature numbers [9]. For eg. words such as organize, organizes, organizing are reduced to base form organiz. Therefore, chances of retrieval of correct information increased, so

whenever the recall needs to be improved stemming is useful. Stemming is performed using Porter's algorithm in python.

3.4 Feature Extraction

The questions title and body are unstructured data in the form text and have to be converted into matrix representation before applying any ML model. The term frequency- inverse document frequency (TF-IDF) vectorization is used for this purpose. The TF-IDF is used to extract features and represent it in the matrix format. The vectors are used to represent a document where each dimension represents its feature. The term frequency represents the number of times the term occurs in the document, and the document frequency is the number of the document in datasets that contain a specific term. TF-IDF representation is performed by using the `tfidfVectorizer` library in python. The questions title and body are converted to vectors with each term assigned a `tf-idf` score. The minimum document frequency and maximum document frequency parameters are used to limit the number of features to extract the relevant terms and remove the very frequent terms which are less significant.

4. Solution Approach and Results

The tag suggestion can be implemented in various ways. The first approach proposed is LDA topic modeling technique [11]. The topic modeling is an unsupervised ML algorithm that uses the Natural Language Processing (NLP) technique which extracts the meaning of the text data in the form of topics. The topic analysis techniques give insight about huge text by giving output as topics that are used most frequently in the data. Topic modeling enables topics extraction from user questions and understands each question content and assigns it appropriate tags.

Another approach is to implement classification models using various ML algorithms. Classification is supervised data mining technique used to assign each point in the dataset a specific target or class. The classification model for tag suggestions will be a multi-label classification as each question can be assigned to more than one tag. The multi-label classification can be implemented using One-vs-Rest technique and Classifier Chain technique [12]. This method is complex and requires high processing power. From the study of existing methodologies, few ML algorithms are implemented which are best suitable for multi-label classification purposes.

4.1 LDA (Latent Dirichlet Allocation)

The topic modeling is the first approach used for tag prediction. With the topic modeling we can discover topical patterns from large datasets, differentiate the set of data according to extracted topics and the dataset can be summarized and organized accordingly. The Latent Dirichlet

Allocation (LDA) is the most commonly used topic modeling technique. LDA provides text classification in a dataset to a particular topic[11]. LDA is a generative probabilistic model that represents documents as a mixture of topics composed of words with different probabilities. LDA is based on a generative process that assumes each document is made up of randomly chosen topic distribution. Each word in the document is picked up by a randomly selected topic over the vocabulary.

4.1.1 Data Preprocessing for LDA

All adjectives, verbs are completely removed using Part of Speech (POS) tags and only nouns which carry important information are passed as an input to LDA to improve the quality of topics. The PoS tags are used to describe a lexical group of words such as verbs, adjectives, and nouns [10]. The PoS tagging can extract nouns as nouns tend to be more significant in deriving topics than adjectives or any other parts of speech. POS can be implemented using NTKL pos tagger library in python.

4.1.2 LDA Implementation

The GenSim library is used to apply LDA in Python. The text in the Body and Title column is converted into dictionaries and corpus with word mappings to their ids for LDA. The LDA is implemented using default parameters with a number of topics as 25. The number of topic parameter (num_topics) must be predetermined and specified while running LDA. LDA provides topics, where each topic has a combination of words. The words under each topic have a certain weight which represents the importance of that word in particular topic.

```

                Probability Word      probability Word
Topic No 2 → [(2, '0.174*array" + 0.103*index" + 0.063*php" +
0.057*byte" + 0.027*transit" + 0.026*categori" +
0.023*length" + 0.019*hash" + 0.016*end" + 0.014*user"),
Topic No 15 → (15, '0.122*color" + 0.068*session" + 0.054*background" +
0.040*none" + 0.031*constraint" + 0.029*person" + 0.027*ms"
+ 0.023*profil" + 0.022*cooki" + 0.018*hibern"),
Topic No 14 → (14, '0.169*int" + 0.073*bar" + 0.070*char" + 0.060*foo" +
0.046*def" + 0.045*float" + 0.044*foo" + 0.039*product" +
0.028*val" + 0.016*self'),
Topic No 8 → (8, '0.115*except" + 0.079*void" + 0.058*string" + 0.050*e"
+ 0.039*catch" + 0.039*throw" + 0.033*intent" + 0.027*tag"
+ 0.024*tri" + 0.024*context'),
Topic No 6 → (6, '0.066*app" + 0.062*io" + 0.032*video" + 0.028*git" +
0.027*push" + 0.026*notif" + 0.026*branch" + 0.023*play" +
0.022*xcode" + 0.020*chrome'),

```

Figure 9. Gensim LDA generated output

Figure 9. Shows part of the output generated by gensim LDA implementation. It displays each topic with its topic number and top 10 words in each topic with its word-probability distribution. All the topics as per the number of topic (num_topics) parameter are displayed. The number of words to be displayed per topic can be adjusted as per requirement. Table 4. shows the results for extracted topics in tabular format for GenSim topic modeling implementation showing top 20 words for each topic. The topics are identified manually from the word distribution and labeled them accordingly, the identified labels are shown in red text below each topic-word list. For eg. Topic #06 has words such as git, push, branch, master, github etc and therefore labeled as version control. Similarly Topic #17 has words such as row, column, queri, databas, select, table etc which is related to Database/SQL. The limitation is categorising different programming languages as they have many similarities such as C, C++, java, python.

Table 4. Manually recognised topics and labeled accordingly

Topic # 05	Topic # 06	Topic # 10	Topic # 11	Topic # 14	Topic # 16	Topic # 17	Topic # 18	Topic # 20	Topic # 21	Topic # 22	Topic # 23	Topic # 24
messag	app	request	file	int	class	row	function	version	list	line	view	control
devic	io	connect	date	bar	name	column	div	project	number	import	button	user
log	video	server	path	char	method	queri	var	applic	item	print	event	form
info	git	servic	directori	foo	type	databas	script	sourc	option	python	item	page
node	push	client	td	def	properti	data	html	packag	group	command	menu	model
stack	notif	respons	tr	float	call	tabl	style	depend	count	modul	activ	post
port	branch	address	folder	foo	instanc	id	text	configur	element	output	cell	json
root	play	password	gem	product	object	record	id	studio	order	pip	icon	view
phone	xcode	http	content	val	paramet	sql	javascript	compil	loop	script	posit	action
bootstrap	chrome	header	download	self	interfac	select	input	load	case	hello	click	data
call	alloc	user	format	function	implemen	mysql	element	build	rang	call	show	field
callback	iphon	access	xml	price	refer	db	page	use	collect	def	label	end
debug	chang	li	pdf	end	declar	insert	src	instal	iter	world	void	cach
report	migrat	secur	datetim	return	constructo	creat	content	spring	break	exit	textview	statu
app	master	login	upload	translat	base	store	jqueryi	support	join	run	style	valid
method	player	authent	marker	x2	field	end	js	app	sort	cat	scroll	rout
crash	repositori	host	export	x1	void	result	consolelog	develop	select	msg	action	scope
eandroidr												
untim	canva	email	asset	mask	pass	transact	css	java	sum	perl	fragment	rail
trace	simul	web	header	tpl	creat	server	bodi	librari	rule	end	navig	compon
driver	github	send	ioexcept	po	attribut	field	type	target	sequenc	execut	context	mvc
Android, iOS, App	Version Control	Server-Client	XML, HTML	Programming	Programming	Database, SQL	HTML	Programming Tool	SQL	CLI, Python	HTML, CSS	JavaScript

4.1.3 LDA Evaluation

The evaluation of topic modelling is very challenging as it is an unsupervised technique and does not have a standard comparable matrix. One of the measures used to find the effectiveness of the lda model is coherence score. The model evaluation is done using coherence score which

indicates performance of selected topic modeling techniques. Topic coherence score is a measure of human interpretability of topic models. It represents relative distance(similarity) between the high probability words within a topic. It helps to identify and distinguish the topic semantic using the cosine similarity. The C_v is a gensim library parameter which is used to calculate the coherence score. Its value lies between 0 and 1. The LDA model does not consider the labels (tags) while training the models. It only considers documents (questions) to generate the models and find topic distribution across the documents. The obtained coherence score for the base LDA model with default parameters is.

Coherence Score: 0.475267

The interactive chart of topics is created using the pyLDavis package in python. Figure 10. shows a screenshot of the interactive chart for the topics extracted from the LDA.

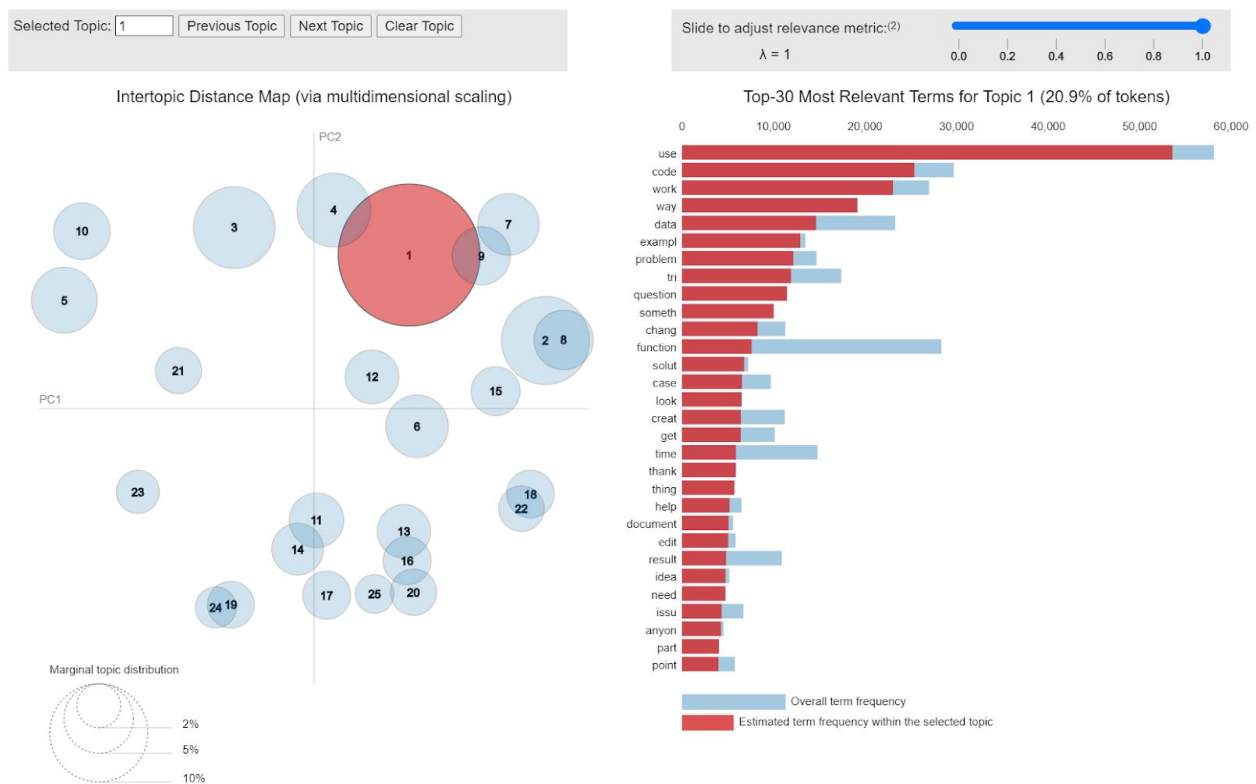


Figure 10. Interactive chart for the topics extracted using LDA

The bubbles in the visualized chart represent individual topics. The overlapped bubbles represent the words which are common in two topics. In figure 10. The bubble number 5, 12 and 21 etc. are unique topics which are not overlapping or very less number of words are overlapped with any other topic. The generated chart is interactive, so hovering over the topic gives statistics such as overall term frequency and estimated term frequency within the selected topic.

4.1.4 LDA Optimization

To optimize the LDA model performance and get more recognizable topics it is important to increase sample size and reduce Tag numbers along with parameter tuning. The methods followed for optimization are:

- **Increasing Data Sample and reduce number of Tags**

Increased the sample size by filtering more questions in preprocessing. To improve the quality of topic extraction 20 Tags were selected. The tags selected are : Java, Python, SQL, mysql, html, css, git, asp.net, vb.net, bash, excel, eclipse, mangodb, andriod, ios, iphone, matlab, api, ruby, ruby-on-rails.

- **Finding an optimum number of topics**

LDA optimization includes various steps such as finding an optimum number of topics for a given dataset and performing hyperparameter tuning. LDA analysis was performed using gensim LDA mallet to find coherence scores for different numbers of topics. It is a time-consuming process as the LDA model needs to be implemented multiple times to find coherence scores for a different number of topics by changing the number of topics from 2 to 30 with a fixed step size of 4. Figure 11. Shows the obtained number of topics vs respective coherence score plot. The maximum coherence score is obtained for 18 numbers of topics. For further fine tuning 18 topics were selected.

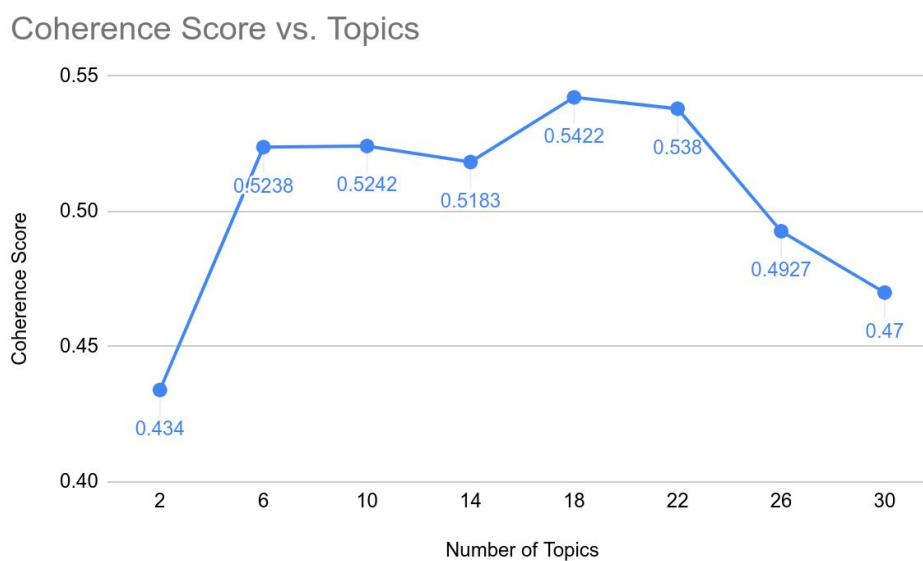


Figure 11. Number of topics vs coherence score

- **LDA parameter tuning**

In LDA each document (questions) is made up of different topics and each topic consists of various words having different weights. The Gensim LDA has two parameters, alpha and eta which can be tuned to obtain the best results. These parameters can be used to control the number of topics in a document (Document-topics density) and the number of words in a topic (Topic-word density). The alpha controls document-topic density i.e. the combination of topics for any given document. If alpha is lower, the documents will likely have less number of topic combinations. If alpha is more, the documents will likely have more number topic combinations. The eta controls topic-word density i.e. the distribution of words per topic. The lower value of eta indicates that the topics will likely have less number of word combinations. The higher eta indicates that the topics will likely have more words.

For finding optimum value for alpha and eta for 18 topics, the LDA is implemented on 75% of the corpus to save processing time with multiple combinations of alpha and eta. Table 5. shows the coherence score for 18 topics with different alpha and eta values. The maximum value of the coherence score is obtained at alpha= 0.01 and eta= 0.9. LDA is implemented again with an optimum value for parameters: number of topics = 18, alpha = 0.01, eta =0.9.

Table 5. Coherence score for different values of eta and alpha

alpha \ eta	0.01	0.31	0.61	0.9
0.01	0.4570	0.5915	0.5967	0.6275
0.31	0.4840	0.5762	0.5752	0.5953
0.61	0.5121	0.5442	0.5600	0.5793
0.9	0.5398	0.5560	0.5847	0.5876

Figure 12. Shows the LDA model with optimum parameters using gensim LDA. Figure 13. Shows the output of gensim LDA with top 5 topics distribution having top 5 words and probability assigned to each word.

```

lda = ldamodel.LdaModel(corpus=corpus,
                        id2word=id2word,
                        num_topics=18,
                        random_state=100,
                        passes=10,
                        per_word_topics=True,
                        alpha=0.01,
                        eta=0.9099)

print(lda.print_topics())
doc_lda = lda[corpus]

```

Figure 12. LDA model with optimised parameters

```

[(0, '0.073*end" + 0.049*user" + 0.045*model" + 0.029*def" +
0.029*rail"'),
(1, '0.230*name" + 0.132*valu" + 0.065*field" + 0.064*type" +
0.035*id"'),
(2, '0.175*view" + 0.059*locat" + 0.047*map" + 0.044>tag" +
0.038*posit" + 0.034*parent" + 0.030*tab" + 0.022*custom" +
0.018*gridview" + 0.015*countri"'),
(3, '0.079*imag" + 0.069*matlab" + 0.031*size" + 0.029*plot" +
0.025*point"'),
(4, '0.166*file" + 0.061*line" + 0.040*script" + 0.037*command" +
0.023*bash"'),
(5, '0.037*time" + 0.037*event" + 0.027*call" + 0.027*block" +
0.026*task"'),

```

Figure 13. LDA output topics with word probabilities

The obtained coherence score for the LDA model with optimum value for parameters with 100% corpus is.

Coherence Score: 0.60158

The coherence score is improved from 0.4752 to 0.6016 (26.6% increase) after parameter tuning.

Table 6. shows the results for extracted topics after performing parameter tuning and displays top 20 words for each topic. The topics are identified from the word distribution and labeled them accordingly, the identified labels are shown in red text. For eg. Topic #12 has words such as git, push, branch, master, github etc and therefore labeled as version control, git. Similarly Topic #07 has words such as row, column, queri, databas, select, table etc which is related to Database/SQL. The topics obtained are better to interpret and categorize with different labels (Tags). The topics generated after fine tuning are better to interpret than the base model.

Table 6. Topics obtained from LDA after parameter tuning

	Topic # 01	Topic # 02	Topic # 03	Topic # 04	Topic # 05	Topic # 06	Topic # 07	Topic # 08	Topic # 09	Topic # 10	Topic # 11	Topic # 12	Topic # 13	Topic # 14	Topic # 15	Topic # 16	Topic # 17
name	user	page	imag	file	time	end	column	way	date	class	android	gem	import	valu	error	cell	server
field	post	class	size	line	process	rang	mongodb	applic	time	method	view	rubi	int	data	eclips	sheet	api
model	action	id	plot	command	task	number	queri	code	day	function	activ	rail	void	matlab	project	button	request
id	password	div	matrix	script	call	amp	databas	web	month	var	intent	repositori	string	array	version	event	messag
proporti	session	form	point	path	memori	dim	data	test	year	view	id	branch	return	function	sourc	color	client
class	email	asp	vector	bash	state	code	row	service	hour	return	textView	commit	except	list	plugin	text	servic
locat	rail	runat	video	directori	block	function	id	connect	format	control	button	git	class	item	info	bar	connect
valu	account	html	posit	output	job	sub	record	app	report	object	gridview	master	catch	way	depend	chang	data
type	access	td	dimens	error	devic	def	group	document	price	code	posit	web1	auto	index	configur	box	error
descript	valid	type	coordin	folder	second	case	tabl	work	timestamp	call	savadinsta	chang	tag	exampl	resourc	click	respons
categori	comment	content	anim	shell	statu	count	product	api	week	instanc	context	version	tri	collect	java	label	json
titl	authent	style	pictur	excel	start	worksheet	mongo	someth	tue	self	bundl	heroku	context	result	packag	tab	web
person	param	bodi	frame	program	applic	valu	select	control	ad	foo	fragment	command	code	search	compil	menu	http
parent	api	text	view	copi	thread	formula	db	googl	subscript	alloc	ntim	push	null	word	instal	item	code
base	login	script	code	code	rate	loop	valu	time	minut	error	listview	requir	byte	document	modul	background	applic
attribut	permiss	tr	map	echo	int	vba	order	exampl	sun	let	imageView	environ	result	element	build	display	rest
book	page	tag	figur	filenam	run	macro	field	solut	amount	init	void	merg	boolean	code	test	code	host
map	end	input	photo	way	level	error	result	site	today	tableview	matchpar	instal	error	match	applic	way	post
compani	filter	control	float	work	event	integ	creat	anyon	pm	sender	wrapcont	develop	data	type	debug	row	access
address	rout	block	area	window	cach	return	count	thing	card	interfac	item	root	method	someth	target	style	network
Programm	security,	html, css	image,	shell	programm	vba, excel	SQL,	api, web	time	Object	Android,	version	C, Java	Programm	IDE,	excel,	client-serv
ing, java,	authentica	animation	script,	script,	ing, java,	python	database	services	frame/	oriented,	os	control,	eclipse	ing	spreadshe	er,	networkin
python	tion	bash	bash	bash	python				format	MVC		git			et	g	

- **Predicting Tags from input text**

The 10 new questions from stack overflow website are manually tested on trained LDA model for tag prediction to observe the performance of the model for unseen data. Below are the questions and user provided tags from Stackoverflow along with the Tags generated by the LDA model with their probabilities. The results for predicted Tags from the input text for one question is shown below and the results for rest of the questions are attached in appendix.

Input question:

(<https://stackoverflow.com/questions/2334712/how-do-i-update-from-a-select-in-sql-server>)

Title: How do I UPDATE from a SELECT in SQL Server?

Body: In SQL Server, it is possible to insert rows into a table with an INSERT.. SELECT statement:
 INSERT INTO Table (col1, col2, col3) SELECT col1, col2, col3 FROM other_table WHERE sql = 'cool'
 Is it also possible to update a table with SELECT? I have a temporary table containing the values and would like to update another table using those values. Perhaps something like this:UPDATE Table SET col1, col2 SELECT col1, col2 FROM other_table WHERE sql = 'cool' WHERE Table.id = other_table.id

Actual Tags: sql, sql-server, tsql, select

Predicted Tags: [(7, 0.9959697)]

Topic # 07	SQL, database
-------------------	----------------------

For all 10 questions the trained LDA model is able to predict most of the primary tags and the most dominant Tag (with maximum probability) matches with the Tags provided by the user. There are three topics which are assigned as programming topics because programming terms and concepts are very common in most of the programming languages such as Java, C, Python etc. Which leads to a general programming Tag for programming related questions.

Along with the manual testing, 2500 more questions from stack overflow dataset are tested for tag suggestion. The dataset for optimised LDA model testing is taken from kaggle (<https://www.kaggle.com/vipulgote4/stackoverflow-dataset/data?select=valid.csv>) Figure 14. shows the results for tags suggested for 2500 questions for tags including sql, git, excel, shell etc. Overall ~67% questions have the correct tags recommended. The overall accuracy is affected because of the ‘android’ tag as it is difficult to distinguish between programming languages.

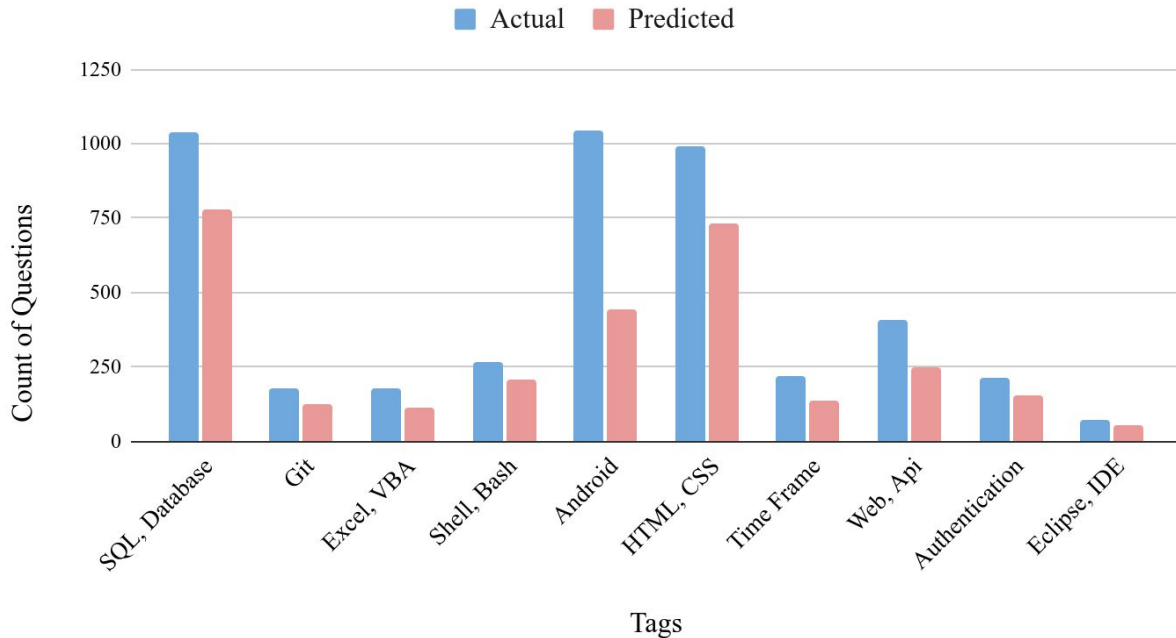


Figure 14. Actual vs predicted Tags for optimized LDA model with unseen input text

4.2 Classification

The classification model for tag suggestions is a multi-label classification as each question can be assigned to more than one tag. The multi-label classification is implemented using One-vs-Rest technique and Classifier Chain technique.

4.2.1 Data Preprocessing for Classification

The initially cleaned and preprocessed data is used for implementation. The Tags column has multiple values for each question and is categorical value, therefore it has to be converted to vector representation. The multilabel binarizer is used for encoding tags column into numeric/vector form. Multilabel binarizer converts the Tags column (which has upto 5 tags per question) into a binary matrix in multilabel format which represents the presence of a particular Tag. It adds columns, one for each Tag to the dataset with binary values 0 or 1. Then TF-IDF is applied to extract and encode text features from the dataset. Then the data is split into training and testing using `train_test_split` method with 80% data as training and 20% as a test set. The model would be built on training data and tested on remaining data.

4.2.2 ML Algorithm Implementation

The following ML algorithms are used for building classification models.

- **Logistic Regression(LR)** : The Logistic Regression is a supervised regression algorithm which predicts the target variables and explains the relation between the dependent variable and one or more independent variables. The dependent variable for Logistic regression is categorical in nature. It is a statistical model which calculates the probability of a result which can have a discrete set of classes and predicts the occurrence of the event using logit function. Logistic regression is easy to implement, interpret and does not require high computation power. Formula (1). Shows the mathematical representation of predicted output for Logistic Regression.

$$y_i = \frac{e^{(\alpha + \beta x_i)}}{1 + e^{(\alpha + \beta x_i)}} \quad i = 1, \dots, n \quad \dots (1)$$

Where,

y = predicted output

α = the bias term

β = coefficient for single input value of x

- **Random Forest(RF)** : The Random Forest model fits decision tree classifiers on the subsamples of the dataset and averages for better accuracy to provide a final prediction model. It is a supervised learning algorithm with an ensemble of Decision Tree along with Bagging method and can be used for both classification and regression. The advantage of random forest is, it gives equal importance to all features in the data to avoid any bias in the model. The independent trained random results in robust models. It handles large samples of data and thousands of input features efficiently.
- **Support Vector Machine(SVM)** : The SVM is a supervised ML algorithm. It is used for both classification and regression. Stochastic Gradient Descent(SGD) is an efficient classifier for selective learning of linear classifiers such as SVM and Logistic Regression. The SVM separates two classes by building hyperplane between two classes. The support vector is defined as the closest sample from classes to the hyperplane and the separation between two classes is identified by the distance between the closest sample to the hyperplane between two classes. The goal of linear SVM is to maximize the margin and have more distinct classes.

- **Linear Support Vector Classification(SVC)** : It is similar to SVM with linear kernel. It has more flexibility in the parameters such as penalties, loss functions etc. to scale large numbers of samples.. Linear SVC classifiers are faster and easy to implement.

The multi-label classification is implemented using One-vs-Rest technique and Classifier Chain technique to transform into single label problem. One-vs-Rest technique is considered as building multiple independent binary classifiers for each class(tag). The One-vs-Rest classification technique is simple and easily interpretable but does not consider the correlations between different classes. Classifier Chain technique takes into account the multiple label correlations. This method is complex and requires high processing power. Classifier Chain technique takes into account the multiple label correlations.

4.2.3 ML Model Evaluation

Performance of models based on Accuracy, Precision, Recall and F-measure, Hamming loss.

- **Accuracy:** The accuracy is the ratio of correctly predicted instances to the total instances. The accuracy can be expressed as follows,

$$\text{Accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN}) \quad (2)$$

- **Precision:** Precision is a measure which is represented as the ratio of correctly predicted positive instances to the total predicted positive instances.

$$\text{Precision} = (\text{TP}) / (\text{TP} + \text{FP}) \quad (3)$$

- **Recall:** Recall is defined as the ratio of correctly predicted positive instances to the all instances in the actual class - Positive or '1'.

$$\text{Recall} = (\text{TP}) / (\text{TP} + \text{FN}) \quad (4)$$

- **F-measure:** The F-measure is a most commonly used measure to evaluate the performance of classification model The F-measure is a weighted average of the precision and recall of the system and it can be given as,

$$\text{F-measure} = 2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall}) \quad (5)$$

- **Hamming Loss:** Hamming-Loss is the fraction of labels that are incorrectly predicted, i.e., the fraction of the wrong labels to the total number of labels. Its value lies between 0 and 1, lower being better.

The following results are obtained with default parameters of the models.

Table 7. Performance metrics for ML algorithms with OneVsRest

One vs Rest	Accuracy	Precision	Recall	F-1 Score	Hamming loss
SVM(SGD)	48.82%	90.15%	54.34%	66.75%	0.0264
LR	54.62%	84.46%	63.64%	72.88%	0.0236
LSVC	56.64%	81.94%	69.13%	74.84%	0.0235
RF	44.87%	89.18%	49.99%	61.95%	0.0286

Table 8. Performance metrics for ML algorithms with ClassifierChain

Classifier Chain	Accuracy	Precision	Recall	F-1 Score	Hamming loss
SVM(SGD)	52.54%	89.49%	58.15%	69.45%	0.0249
LR	58.02%	85.69%	66.85%	74.63%	0.0226
LSVC	58.88%	80.40%	71.60%	75.62%	0.0235
RF	46.42%	90.28%	53.78%	63.15%	0.0278

LinearSVC has shown overall better performance in both the multilabel classification techniques. The LinearSVC is selected as a core algorithm for further improvement in the performance metrics as it has flexible parameters for optimization.

4.2.4 Performance Tuning

To improve performance metrics of the finalized model limited Tags are considered. The top 15 Tags are considered for further analysis while performing performance tuning. Due to limitations on time and resources (processing power) and also the need of running multiple models for parameter tuning, top 15 Tags are selected. Previously TF-IDF was applied to extract and encode text features from the dataset. The parameters 'max_features', 'min_df', 'max_df' are varied to ignore terms which are less significant while building the vocabulary based on the term frequency across the corpus. Having more features in the data while building models can increase processing time and calculation levels in the model. Without limitation on feature size total 12,87,101 features were present after Tfidf vectorization. With the help of mentioned parameters of TfidfVectorizer it is reduced to 1,00,000. The tuning parameters and its values are given in Figure 15.

```
vectorizer_X = TfidfVectorizer(analyzer = 'word',
                              min_df=0.0005,
                              max_df = 1.0,
                              strip_accents = None,
                              encoding = 'utf-8',
                              ngram_range = (1, 1),
                              preprocessor=None,
                              token_pattern=r"(?u)\S\S+",
                              max_features=100000)
```

Figure 15. Parameter settings for TfidfVectorizer

- **Tuning LinearSVC:**

LinearSVC has the following parameter which can be tuned to obtain improvement in the performance[13].

Penalty: The dataset containing a large number of features tends to create more complex models and cause the problem of overfitting. The regularization technique adds penalty in order to create less complex models so that model generalises and will not overfit. The L1 regularization shrinks the coefficient of less important features to zero by eliminating some features. It works as feature selection for large feature scenarios. L2 regularization reduces the coefficient of by adding penalty which leads to less complex and unbiased models.

Regularization parameter: The strength of the regularization is inversely proportional to regularization parameter.

The GridSearchCV is a python function used to estimate the best parameters by cross validation over various parameters specified to be optimized. It makes hyperparameter tuning easy and finds the best optimal parameter values for a given model. The cross validation parameter enables k fold cross validation which splits the data and runs multiple iterations while generating best models. Figure 16. shows the part of terminal output by GridSearchCV iteration with different estimators and accuracy. Figure 17. shows best estimated parameters for linearSVC generated as a result of GridSearchCV parameter estimation.

```

[CV] estimator_C=1.2915496650148839, estimator_penalty=l1 .....
[CV] estimator_C=1.2915496650148839, estimator_penalty=l1, score=0.759, total= 4.4min
[CV] estimator_C=1.2915496650148839, estimator_penalty=l1 .....
[CV] estimator_C=1.2915496650148839, estimator_penalty=l1, score=0.759, total= 4.5min
[CV] estimator_C=1.2915496650148839, estimator_penalty=l2 .....
[CV] estimator_C=1.2915496650148839, estimator_penalty=l2, score=0.753, total= 24.9s
[CV] estimator_C=1.2915496650148839, estimator_penalty=l2 .....
[CV] estimator_C=1.2915496650148839, estimator_penalty=l2, score=0.751, total= 23.7s
[CV] estimator_C=1.2915496650148839, estimator_penalty=l2 .....
[CV] estimator_C=1.2915496650148839, estimator_penalty=l2, score=0.753, total= 25.2s
[CV] estimator_C=1.2915496650148839, estimator_penalty=l2 .....
[CV] estimator_C=1.2915496650148839, estimator_penalty=l2, score=0.755, total= 24.9s
[CV] estimator_C=1.2915496650148839, estimator_penalty=l2 .....
[CV] estimator_C=1.2915496650148839, estimator_penalty=l2, score=0.756, total= 25.1s
[CV] estimator_C=1.6681005372000588, estimator_penalty=l1 .....
[CV] estimator_C=1.6681005372000588, estimator_penalty=l1, score=0.753, total= 5.5min
[CV] estimator_C=1.6681005372000588, estimator_penalty=l1 .....
[CV] estimator_C=1.6681005372000588, estimator_penalty=l1, score=0.750, total= 5.5min

```

Figure 16. GridSearchCV parameter tuning output

```

estimator=OneVsRestClassifier(estimator=LinearSVC(C=1.0,
class_weight=None,
dual=False,
fit_intercept=True,
intercept_scaling=1,
loss='squared_hinge',
max_iter=1000,
multi_class='ovr',
penalty='l2',
random_state=1,
tol=0.0001,
verbose=0),
n_jobs=None),
iid='deprecated', n_jobs=None,
param_grid={'estimator_C': array([ 1.          , 1.29154967, 1.66810054, 2.15443469, 2.7825594 ,
3.59381366, 4.64158883, 5.9948425 , 7.74263683, 10.          ]),
'estimator_penalty': ['l1', 'l2']},
pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
scoring=make_scorer(accuracy_score), verbose=5)

```

Figure 17. Best estimated parameters for linearSVC using GridSearchCV

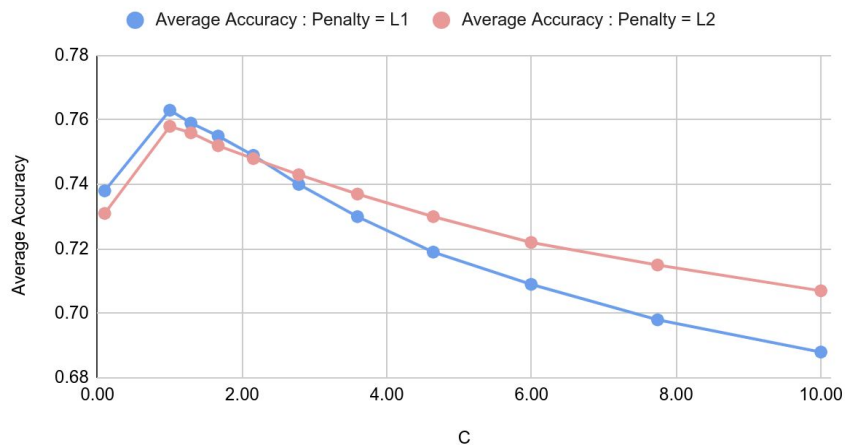


Figure 18. GridSearchCV results for LinearSVC model

The GridSearchCV results shown in Figure 18. Represents average accuracy for different Penalty types (L1/ L2) over different Regularization parameter (C). The regularization parameter is varied in the range of 0 to 10 to find the best accuracy score. The value of accuracy initially

increases upto $C = 1$ and then gradually decreases. The maximum average accuracy of 0.763 is obtained at $C = 1$ by performing L1 regularization whereas maximum average accuracy of 0.758 is obtained at $C = 1$ by performing L2 regularization with LinearSVC model.

AUC ROC Curve:

AUC(Area under curve) - ROC (Receiver Operating Characteristics) is one of the metrics of performance evaluation of classification models[14]. ROC represents a probability curve which plots true positive rate (TPR) against false positive rate (FPR). AUC represents degree or measure of separability between the classes. Higher AUC represents better the model with greater chance at predicting 0s as 0s and 1s as 1s. An excellent model has AUC equal to 1.

$$\text{TPR} = \text{TP} / (\text{TP} + \text{FN})$$

$$\text{FPR} = \text{FP} / (\text{TN} + \text{FP})$$

Figure 19. shows the AUC-ROC curve with AUC for multiple labels (15 Tags) and averaged AUC for linear SVC. The averaged AUC is ~0.92 which indicates that the trained classification model has good distinction between the labels.

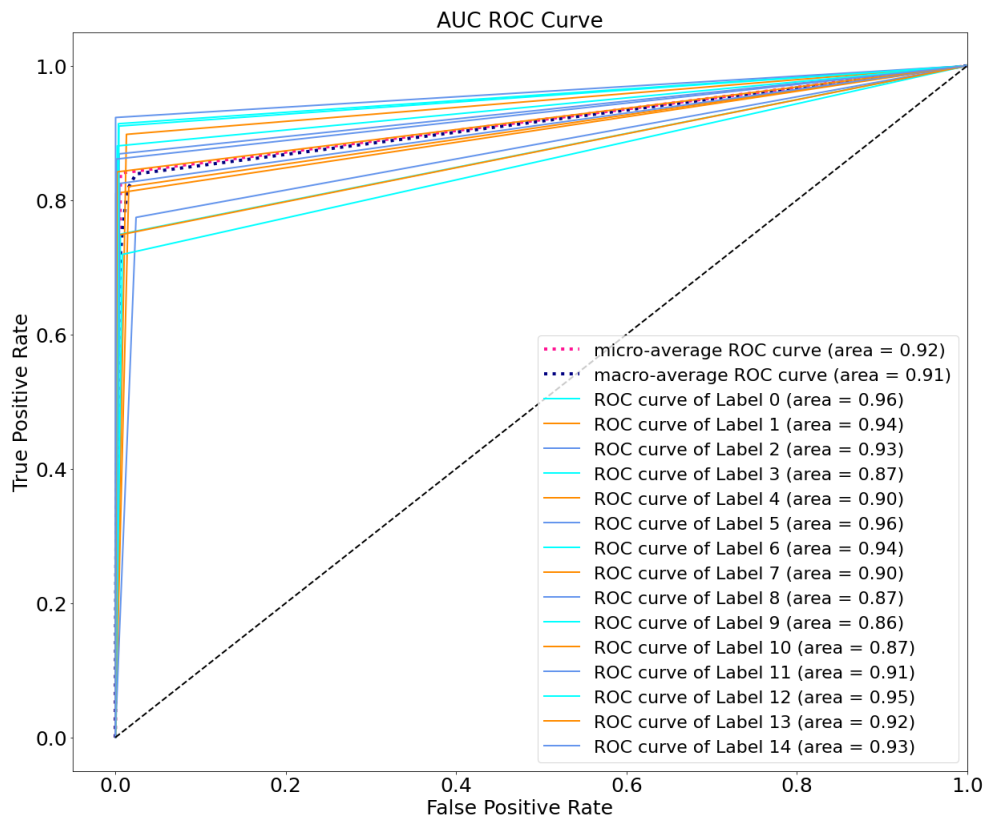


Figure 19. AUC-ROC curve for LinearSVC model

Figure 20. represents each tag with its number of questions in actual(test) and correctly predicted(pred) instances. For C#, 5714 out of 5892 questions are correctly labeled.

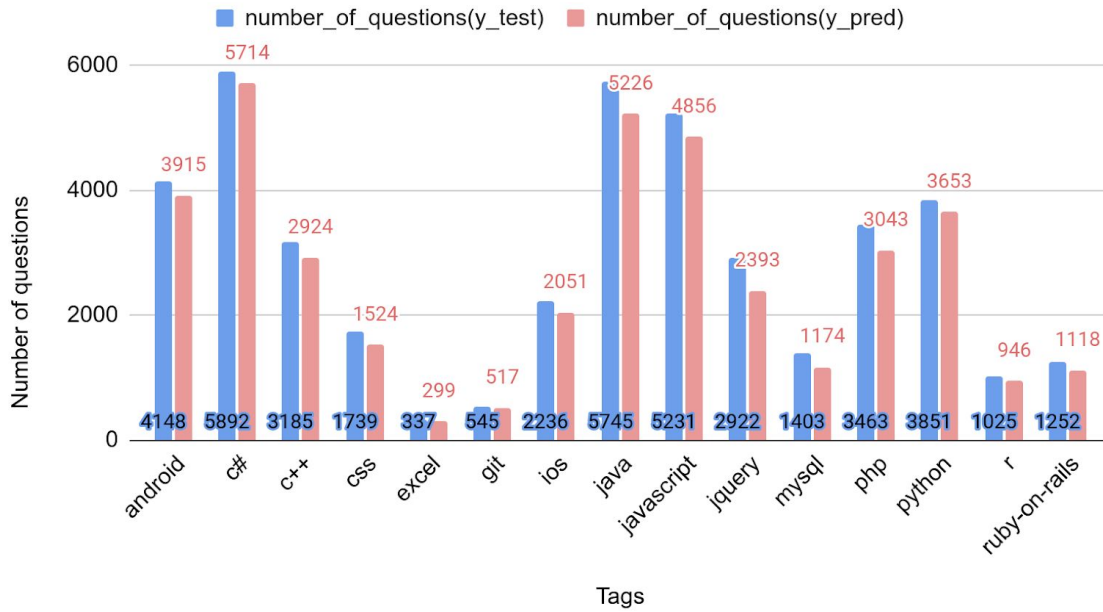


Figure 20. Number of questions vs Tags for test and predicted data

Figure 21. represents number of questions vs number of actual and predicted tag predictions. 34107 questions in the test dataset have only one tag associated with it. The classification model has predicted 33210 questions with one tag. Similarly, 3738 questions in the test dataset have two tags associated with it. The classification model has predicted 2956 questions with two tags.

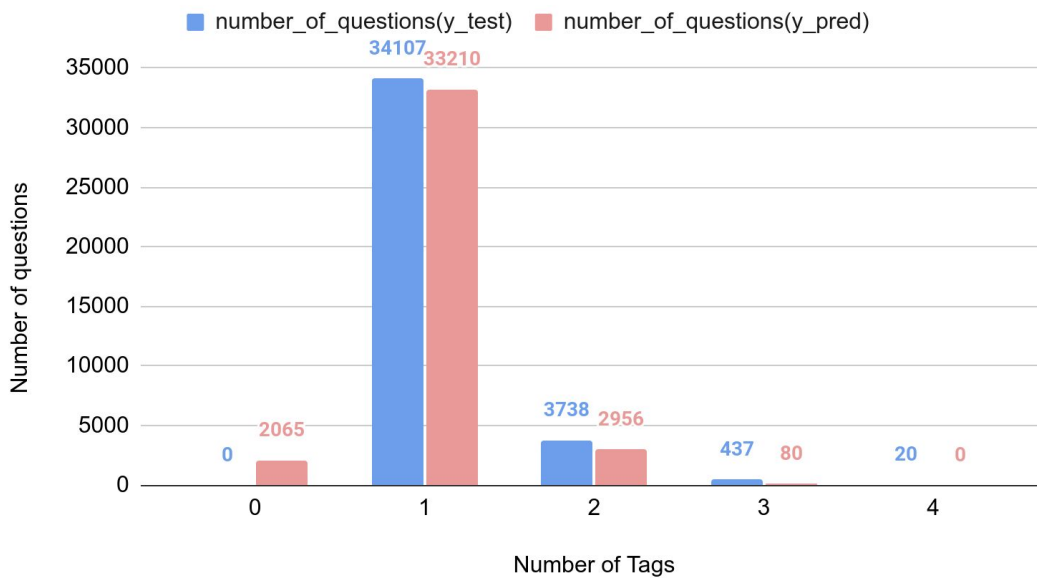


Figure 21. Number of questions vs Number of Tags for test and predicted data

Table 9. shows the improved performance metrics values after parameter tuning of penalty,regularization for Linear SVC with classifier chain technique. Compared to previous results from Table 7, 8, Accuracy is improved by 36.9%, Recall is improved by 18.7%, precision is increased by 13.2% and F1-score is improved by 16.3%.

Table 9. Performance metrics for LinearSVC after parameter tuning

Metric	Accuracy	Recall	Precision	F1 Score	Hamming loss
LinearSVC with L1 Penalty	80.55%	84.96%	91.08%	87.90%	0.01747
LinearSVC with L2 Penalty	80.61%	85.01%	91.01%	87.91%	0.01750

5. Conclusion

In the development of automatic Tag suggestion system for StackOverflow various ML techniques have been used. The unsupervised topic modeling approach - LDA is the one approach proposed for auto Tag suggestion in this study. The baseline LDA model from gensim is applied to the Question, Title and Body content of the selected dataset to generate various topics with coherence score of 0.4752. Further feature extraction and hyperparameter tuning indicated, model built with 18 topics yields maximum coherence score. The fine tuned model with 18 topics, alpha (document-topic density) = 0.01 and eta (topic-word density) = 0.9 parameters generated coherence score of 0.6015. The ~26% of improvement of coherence score led to more interpretable and distinct topics. The final model correctly predicted the primary Tags with maximum probability for the unseen questions. The second approach proposed for this study is performing classification using various ML algorithms. For baseline performance measurement logistic regression, random forest, SVM and LinearSVC are used to discover the best suited model. The model evaluation is performed using accuracy, precision, recall and hamming loss. The fine tuning of the core algorithm, LinearSVC resulted in accuracy improvement of ~36% and f-1 score improvement of ~16%. The LinearSVC with Penalty (L2), Regularization parameter (C) =1 performed best with accuracy of 80.61%, f-1 score of 87.91% and AUC-ROC of ~0.92.

References

- [1] Anon. What are tags, and how should I use them? - Help Center. Retrieved February 10, 2020 from <https://stackoverflow.com/help/tagging>
- [2] Adinarayanan, Smrithi Rekha & N, Divya & Sivakumar, P. (2014). A Hybrid Auto-tagging System for StackOverflow Forum Questions. Retrieved February 10, 2020, 10.1145/2660859.2660970.
- [3] Alrashedy, Kamel & Dharmaretnam, Dhanush & German, Daniel & Srinivasan, Venkatesh & Gulliver, T. Aaron. (2018). Predicting the Programming Language of Questions and Snippets of StackOverflow Using Natural Language Processing.
- [4] Short L, Wong C, Zeng D. Tag recommendations in stackoverflow. San Francisco: Stanford University, 2014, Google Scholar.
- [5] Allamanis, Miltiadis & Sutton, Charles. (2013). Why, when, and what: Analyzing Stack Overflow questions by topic, type, and code. Retrieved February 10, 2020, 53-56. 10.1109/MSR.2013.6624004.
- [6] Stanley, Clayton, and Michael D. Byrne. "Predicting Tags for StackOverflow Posts." Proceedings of ICCM. 2013.
- [7] Z. Yue, Y. Jiang, D. Pan, and Z. Luo. 2017. An End-to-end Tag-based Recommendation System for Verbal Reasoning Questions. In Proceedings of the 10th EAI International Conference on Simulation Tools and Techniques (SIMUTOOLS '17). Association for Computing Machinery, New York, NY, USA, 131–135. DOI:<https://doi.org/10.1145/3173519.3173530>
- [8] Stack Overflow. 2019. StackSample: 10% of Stack Overflow Q&A. (October 2019). Retrieved February 10, 2020 from <https://www.kaggle.com/stackoverflow/stacksample/data>
- [9] Anon. 2009. Stemming and lemmatization. (April 2009). Retrieved February 10, 2020 From <https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization>
- [10] Anon. 2019. 5. Categorizing and Tagging Words. (September 2019). Retrieved February 10, 2020 from <https://www.nltk.org/book/ch05.html>
- [11] Blei, David & Ng, Andrew & Jordan, Michael. (2001). Latent Dirichlet Allocation. The Journal of Machine Learning Research. 3. 601-608.
- [12] Anon. 2007. 1.12. Multiclass and multilabel algorithms. (2007). Retrieved February 10, 2020 from <https://scikit-learn.org/stable/modules/multiclass.html>
- [13] Hsu, C. & Chang, C. & Lin, C.. (2008). A practical guide to support vector classification. BJU International. 101. 1396-1400.

[14] Markham, K. (2020, February 03). ROC curves and Area Under the Curve explained (video). Retrieved November 08, 2020, from <https://www.dataschool.io/roc-curves-and-auc-explained/>

Appendix

Input Text 1:

<https://stackoverflow.com/questions/64528613/pivot-table-filter-not-changing-to-each-value-in-loop>

Title: Pivot Table filter not Changing to each value in Loop

Body: I've seen a lot of similar posts but none of them have been able to fix my issue with changing a pivot table filter field. I'm trying to filter through a loop that takes each value in the list on ws2 and pastes changes the FilterID to that value. However in all the different methods I've tried, setting it to a value, setting it to a string, using "CurrentPage", none of them have worked or resulted in a 1004 error. My latest effort is below. How can I get the "MatchFilter" to change based on the value in my loop below ?

The method below just gives a 1004 Application or object not defined error.

I also tried to add MatchFilter.Orientation = xlPageField but no success.

```
Sub Filter_Test()
```

```
Dim ws1 As Worksheet
```

```
Set ws1 = Sheets("Order_Groupings")
```

```
'The match list is the list of unique Match IDs to filter through, pasting each in the filter
```

```
Dim ws2 As Worksheet
```

```
Set ws2 = Sheets("Match_List")
```

```
'Here we will paste the results from each optimizer run
```

```
Dim ws3 As Worksheet
```

```
Set ws3 = Sheets("Optimizer_Results")
```

```
Dim pt As PivotTable
```

```
Set pt = ws1.PivotTables("Order_Groupings")
```

```
Dim FilterID As String
```

```
Dim MatchFilter As PivotField
```

```
Set MatchFilter = pt.PivotFields("Match_ID")
```

```
Dim numIDs As Integer
```

```
'This is the number of different match IDs
```

```
'Match count is set to J4 right now in the Match List tab
```

```
numIDs = ws2.Range("match_count").Value
```

```
'For loop to cycle through each Match ID
```

```
For i = 1 To numIDs
```

```
    FilterID = ws2.Range("A4").Offset(i, 0).Value
```

```
    'Trying to set the MatchFilter to the new value in FilterID
```

```
    With MatchFilter
```

```
        .ClearAllFilters
```

```
        .CurrentPageName = FilterID
```

End With
Next i
End Sub

Tags: excel, vba, pivot-table

- Predicted Tags after applying trained LDA model
[(1, 0.08263205), (6, 0.3817161), (7, 0.08705216), (11, 0.017354155), (14, 0.34400588), (16, 0.08579699)]

Topic # 01	security, authentication
Topic # 06	vba, excel
Topic # 07	SQL, database
Topic # 11	Android, os
Topic # 14	programming
Topic # 16	excel, spreadsheet

Input Text 2:

<https://stackoverflow.com/questions/64527950/not-able-to-put-a-condition-in-shell-script>

Title: Not able to put a condition in shell script

Body: I am new to shell scripting I have been trying to create a shell script which runs on every month end. But the script doesn't run on Sundays and Mondays. Only Tues-Sat. So if the month end falls on sun or mon, it will run 28th or 29th. Eg- if a month is of 30 days then ideally script should run on 30. But if 30th falls on Sunday then the script should run on 29th.

Tags: bash, shell, sh

- Predicted Tags after applying trained LDA model
[(4, 0.43657923), (6, 0.15265396), (9, 0.402516)]

Topic # 04	shell script, bash
Topic # 06	vba, excel
Topic # 09	time frame/ format

Input Text 3:

<https://stackoverflow.com/questions/63192324/djoser-permission-classes-issue>

Title: Djoser Permission Classes Issue

Body: I am trying make an app with djoser as my third party package. I have made a bunch of @api_views along with it and i wanted to apply some permissions as well. I have already done IsAuthenticated but am unable to update custom permission for djoser:

'user': ['djoser.permissions.CurrentUserOrAdminOrReadOnly']

I have used PERMISSION instead of DEFAULT_PERMISSION_CLASSES in djoser list.

I would like to be able to get a token and only able to change that users data and not someone elses but till now it hasnt worked. i can update or post any type of data of different user with a valid token.

Tags: django, authentication, permissions, customization

- Predicted Tags after applying trained LDA model

[(1, 0.5727031), (2, 0.062216956), (8, 0.15459637), (14, 0.1453297), (15, 0.059292734)]

Topic # 01	security, authentication
Topic # 02	html, css
Topic # 08	api, web services
Topic # 14	programming
Topic # 15	IDE, eclipse

Input Text 4:

<https://stackoverflow.com/questions/64402280/how-do-i-differentiate-enabled-and-disabled-tracks-with-spotify-web-api>

Title: How do i differentiate enabled and disabled tracks with Spotify Web Api?

Body: I'm working with the Spotify Web Api in C# and I want to find out which tracks of my playlists are disabled. I know how to get all the FullTrack-Objects from my playlists, but I don't know which property tells me whether the track is disabled or enabled. Here's the link to the FullTrack-Object documentation: <https://developer.spotify.com/documentation/web-api/reference/tracks/get-track/>

In the following picture you see what i mean with disabled/enabled tracks: Disabled/enabled Track example

The disabled track 'Get Up' is greyed out and I can't play it. The enabled track 'Glitch Gang' is not greyed out and playable. Note: Both tracks are not local!

A FullTrack-Object has the property is_playable... But that seems to be always null no matter what.

Then I tried to identify disabled tracks according to the available_markets property. I thought those would be null or empty when the track is disabled, but that wasn't the case. Tracks that were enabled sometimes also had an empty array of available_markets.

So I'm stuck here... I don't know how or if it's even possible to differentiate disabled/enabled tracks with the Spotify Web Api. Does Anyone have a answer to that?

Tags: api, web, spotify

- Predicted Tags after applying trained LDA model

[(0, 0.16001536), (5, 0.10698558), (8, 0.62986875), (14, 0.036733307), (17, 0.06364164)]

Topic # 00	java, python
Topic # 05	java, python
Topic # 08	api, web services
Topic # 14	programming
Topic # 17	client-server, networking

Input Text 5:

<https://stackoverflow.com/questions/64530635/get-resource-id-from-multiple-imageviews-and-assign-drawable-to-them>

Title: Get Resource Id from multiple ImageViews and assign drawable to them

Body: My app has a GridLayout and 60 ImageViews on it. I want to assign a drawable to 25 cells of grid layout. for that purpose i have an array with 25 random numbers.

then i defined a hashmap that each item is one of the image views:

```
private HashMap<Integer, Integer> imageViews = new HashMap<>();
```

```
imageViews.put(0, R.id.imageView1);
```

```
imageViews.put(59, R.id.imageView60);
```

and put 60 imageviews in hashmap ... 0 is id for first imageview. and set drawable to cells with this method:

```
private void setBombDrawable(){
    HashMap<Integer, Integer> map = mImageViewsMap.getImageViews();
    for (int tag : randomBombArray) {
        int id = map.get(tag);
        ImageView imageView = findViewById(id);
        imageView.setImageResource(R.drawable.exploded);
    }
}
```

code runs without problem, but if there is any simpler way for getting imageviews id or tag from grid layout instead of 60 lines of code?

Tags: android, hashmap, imageview, drawable, grid-layout

- Predicted Tags after applying trained LDA model

[(3, 0.07239184), (6, 0.13976413), (11, 0.47108233), (13, 0.12926328), (14, 0.1026138), (16, 0.08239396)]

Topic # 03	image, animation
Topic # 06	vba, excel
Topic # 11	Android, os
Topic # 13	C, Java

Topic # 14	programming
Topic # 16	excel, spreadsheet

Input Text 6:

<https://stackoverflow.com/questions/309424/how-do-i-read-convert-an-inputstream-into-a-string-in-java>

Title: 'How do I read / convert an InputStream into a String in Java?

Body: If you have a java.io.InputStream object, how should you process that object and produce a String? Suppose I have an InputStream that contains text data, and I want to convert it to a String, so for example I can write that to a log file. What is the easiest way to take the InputStream and convert it to a String?

Tags: Java, string, io, stream, inputstream

- **Predicted Tags after applying trained LDA model**

[(5, 0.15902096), (13, 0.3767766), (14, 0.45493168)]

Topic # 05	java, python
Topic # 13	C, Java
Topic # 14	programming

Input text 7:

<https://stackoverflow.com/questions/2610497/change-an-html5-inputs-placeholder-color-with-css>

Title: Change an HTML5 input's placeholder color with CSS

Body: Chrome supports the placeholder attribute on input[type=text] elements (others probably do too). But the following CSS doesn't do anything to the placeholder's value:

```
input[placeholder], [placeholder], *[placeholder] {
    color: red !important; }
```

```
<input type="text" placeholder="Value">
```

Expand snippet

Value will still remain grey instead of red.

Is there a way to change the color of the placeholder text?

Tags: css, html, placeholder, html-input

- **Predicted Tags after applying trained LDA model**

[(2, 0.5579981), (14, 0.19046639), (16, 0.2470147)]

Topic # 02	html, css
Topic # 14	programming
Topic # 16	excel, spreadsheet

Input Text 8:

<https://stackoverflow.com/questions/927358/how-do-i-undo-the-most-recent-local-commits-in-git>

Title: How do I undo the most recent local commits in Git?

Body: I accidentally committed the wrong files to Git, but I haven't pushed the commit to the server yet. How can I undo those commits from the local repository?

Tags: git, version-control, git-commit, undo

- Predicted Tags after applying trained LDA model
[(4, 0.08792023), (12, 0.89251983)]

Topic # 04	shell script, bash
Topic # 12	version control, git

Input Text 9:

<https://stackoverflow.com/questions/1274057/how-to-make-git-forget-about-a-file-that-was-tracked-but-is-now-in-gitignore>

Title: How to make Git “forget” about a file that was tracked but is now in .gitignore?

Body: There is a file that was being tracked by git, but now the file is on the .gitignore list. However, that file keeps showing up in git status after it's edited. How do you force git to completely forget about it?

Tags: git, gitignore, git-rm

- Predicted Tags after applying trained LDA model
[(4, 0.28245988), (12, 0.7044038)]

Topic # 04	shell script, bash
Topic # 12	version control, git