INTRODUCTION

OVER VIEW:

A system that supports autonomous tagging can improve the user experience by clustering information into discrete common topics. The other benefit is that the user can be recommended queries related to his own problem which could help him find the answer in an efficient and effective manner.

PURPOSE:

Stack Overflow is a question-and-answer website for programmers. It is the flagship site of the Stack Exchange Network. It was created in 2008 by Jeff Atwood and Joel Spolsky. It features questions and answers on certain computer programming topics.

LITERATURE SURVEY

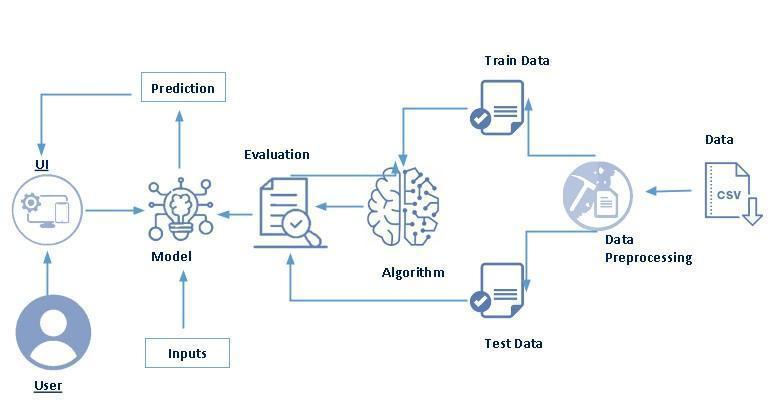
Existing problem:

The existing problem of this analysis is the normal analysis approach from the given data set of the stack overflow.

Proposed solution:

From the given stack overflow data the analysis is done using Python, Data science, Machine Learning, Web Framework like flask or Django, IBM Watson Studio. Here we aim to develop an autonomous system that can accurately and efficiently assign relevant tags to Stack Overflow questions without manual intervention.The system should take into account the question text and other contextual information to identify the most relevant tags The autonomous tagging system will be able to accurately predict relevant tags for Stack Overflow questions. It will enhance the organization and searchability of questions, facilitating a smoother user experience for developers seeking information on the platform.

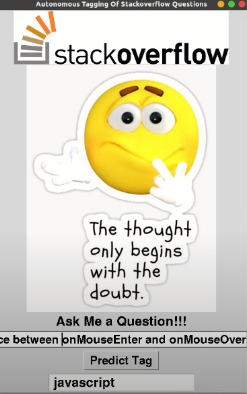
FLOWCHART:



EXPERIMENTAL INVESTIGATIONS:

Tags play a crucial role in organizing and categorizing questions, enabling users to search and find relevant information effectively.Currently, tags on Stack Overflow are assigned manually by users, which can be time-consuming and inconsistent. Moreover, as the platform grows, the number of questions increases rapidly, making it challenging to keep up with the tagging process.The objective is to create an automated solution using machine learning to accurately predict and assign appropriate tags to Stack Overflow questions.

RESULT:



ADVANTAGES:

It provides quick problem-solving, access to a vast knowledge base, improved code quality, networking opportunities, and career development.

APPLICATIONS:

A tag is a word or phrase that describes the topic of the question and helps categorize the question with other, similar questions. Tags help connect experts with questions they will be able to answer, by sorting questions into specific, well-defined categories.

* Identify a language.
* Draw attention from a particular department.
* Label a product.

CONCLUSION:

autonomous tagging has the potential to offer significant advantages in terms of efficiency and consistency on platforms like Stack Overflow. However, its successful implementation should be carefully planned, taking into account user feedback and the need for ongoing model refinement.

APPENDIX:

SOURCE CODE:

Loading data:

**import** pandas **as** pd

df**=**pd**.**read\_csv('stackoverflowtags.csv')

df**.**head(2)

df**.**title**.**value\_counts() *# counts the number of occurances in the data*

*draping duplicates:*

df**.**title**=**df**.**title**.**drop\_duplicates(keep**=**'first') *# drops the duplicates*

df**.**title**.**value\_counts()

Text cleaning:

**import** re *# importing regular expressions used for cleaning texts*

*# importing natural language toolkit*

*# that helps in cleaning texts by using*

*# stopwords, SnowballStemmer, WordNetLemmatizer libraries*

**from** nltk.corpus **import** stopwords

**from** nltk.stem.snowball **import** SnowballStemmer

**from** nltk.stem.wordnet **import** WordNetLemmatizer

*# function to remove html tags and other*

*# unwanted stuff in the question asked by the user*

**def** clean(s):

s**=**str(s)

s**=**s**.**lower()

html**=**re**.**compile('<.\*?>') *#removing html tags*

cleaned **=** re**.**sub(html,' ',s)

fil**=**[]

**for** i **in** cleaned**.**split(): *# splits the text and repalces the unwanted characters with ''*

**if** i**!=**'c++':

cleaned**=**re**.**sub('[^A-Za-z]', '', i) *#search the pattern !(A-Z & a-z) and replace with ''*

fil**.**append(cleaned)

**else**:

fil**.**append(i)

**return** fil *# returns the splitted text with removed stopwords and html tags*

stop**=**set(stopwords**.**words('english')) *#loading stopwords in english to compare and remove*

sno**=**SnowballStemmer('english')

clean(df**.**title[0]) *# function call to clean the text*

Stemming:

*# function to stem the data*

*# stemming means grouping the words*

*# after cleaning the data for further processing*

**def** stem(s):

fil**=**[]

**for** i **in** s:

**if** i **not** **in** stop:

s**=**(sno**.**stem(i)**.**encode('utf8')) *# encoding the data into a clean file*

fil**.**append(s)

s**=**b' '**.**join(fil)

**return** s

*# creating the new columns*

*# cleaned questions and cleaned tags*

*# after cleaning the data*

ques**=**[]

**for** j **in** df**.**title:

ques**.**append(stem(clean(j)))

df['cleanQues'] **=** ques

**import** re

ctags**=**[]

**for** i **in** df**.**tags:

ctags**.**append(re**.**sub('[^A-Za-z#+-]', ' ', i)) *# search the pattern !(A-Z & a-z) and replace with ''*

df['cleanTags']**=**ctags

After cleaning and stemming:

df**.**head(10)

*# creating the new dataset that only*

*# consists cleaned questions and tags*

d**=**pd**.**DataFrame()

d['text']**=**df**.**cleanQues

d['tags']**=**df**.**cleanTags

d**.**to\_csv('datafinal',index**=False**)

df **=** pd**.**read\_csv('datafinal')

df**.**head()

Splitting dataset:

*# splitting the dataset to train and test in 80% and 20%*

**from** sklearn.model\_selection **import** train\_test\_split

x\_train,x\_test,y\_train,y\_test**=**train\_test\_split(df**.**text, df**.**tags, test\_size**=**0.2, random\_state**=**9)

Converting text and tags to vectors:

*# Used ti-idf , bow*

*# importing TfidfVectorizer , CountVectorizer from sklearn.feature\_extraction*

*# to convert the text and tags to vectors*

*# so that we can train and test the dataset*

**from** sklearn.feature\_extraction.text **import** TfidfVectorizer,CountVectorizer

tfvectorizer **=** TfidfVectorizer(min\_df**=**0.00009, max\_features**=**200000, smooth\_idf**=True**, norm**=**"l2",

tokenizer **=** **lambda** x: x**.**split(), sublinear\_tf**=False**, ngram\_range**=**(1,3))

x\_train\_multilabel **=** tfvectorizer**.**fit\_transform(x\_train)

x\_test\_multilabel **=** tfvectorizer**.**transform(x\_test)

vectorizer **=** CountVectorizer(tokenizer **=** **lambda** x: x**.**split(), binary**=**'true')

y\_train\_multilabel **=** vectorizer**.**fit\_transform(y\_train)

y\_test\_multilabel **=** vectorizer**.**transform(y\_test)

Training using one vs rest

*# using multi-class classification to classify the tags*

*# classifiers like One-Vs-Rest classifier and Stochastic Gradient Descent Classifier*

**from** sklearn.multiclass **import** OneVsRestClassifier

**from** sklearn.linear\_model **import** SGDClassifier

**from** sklearn **import** metrics

**from** sklearn.metrics **import** f1\_score,precision\_score,recall\_score

classifier **=** OneVsRestClassifier(SGDClassifier(loss**=**'log', max\_iter **=** 5, tol **=** **None**, alpha**=**0.00001, penalty**=**'l1'), n\_jobs**=-**1)

classifier**.**fit(x\_train\_multilabel, y\_train\_multilabel)

predictions **=** classifier**.**predict(x\_test\_multilabel)

print("accuracy :",metrics**.**accuracy\_score(y\_test\_multilabel,predictions))

print("macro f1 score :",metrics**.**f1\_score(y\_test\_multilabel, predictions, average **=** 'macro'))

print("micro f1 scoore :",metrics**.**f1\_score(y\_test\_multilabel, predictions, average **=** 'micro'))

print("hamming loss :",metrics**.**hamming\_loss(y\_test\_multilabel,predictions))

Classification report

*# classification report*

print("Precision recall report :\n",metrics**.**classification\_report(y\_test\_multilabel, predictions))

*# using dill saving the classification, tfvectorization nad vectorization*

*# in three files and using them to predict the tags*

**import** dill

model\_data **=** 'model\_data.sav'

tfidf\_data **=** 'tfidf\_data.sav'

bow\_data **=** 'bow\_data.sav'

dill**.**dump(classifier, open(model\_data, 'wb'))

dill**.**dump(tfvectorizer, open(tfidf\_data, 'wb'))

dill**.**dump(vectorizer, open(bow\_data, 'wb'))

**import** pandas **as** pd

**import** dill

**import** re

*# opening the saved files*

*# which are used to predict the tags*

*# to the questions asked by the user*

classifier **=** dill**.**load(open('model\_data.sav', 'rb'))

tfvectorizer **=** dill**.**load(open('tfidf\_data.sav', 'rb'))

vectorizer **=** dill**.**load(open('bow\_data.sav', 'rb'))

**import** re *# importing regular expressions used for cleaning texts*

*# importing natural language toolkit*

*# that helps in cleaning texts by using*

*# stopwords, SnowballStemmer, WordNetLemmatizer libraries*

**from** nltk.corpus **import** stopwords

**from** nltk.stem.snowball **import** SnowballStemmer

**from** nltk.stem.wordnet **import** WordNetLemmatizer

**def** clean(s):

s**=**str(s)

s**=**s**.**lower()

html**=**re**.**compile('<.\*?>') *#removing html tags*

cleaned **=** re**.**sub(html,' ',s)

fil**=**[]

**for** i **in** cleaned**.**split(): *# splits the text and repalces the unwanted characters with ''*

**if** i**!=**'c++':

cleaned**=**re**.**sub('[^A-Za-z]', '', i) *#search the pattern !(A-Z & a-z) and replace with ''*

fil**.**append(cleaned)

**else**:

fil**.**append(i)

**return** fil *# returns the splitted text with removed stopwords and html tags*

stop**=**set(stopwords**.**words('english')) *#loading stopwords in english to compare and remove*

sno**=**SnowballStemmer('english')

*# function to stem the data*

*# stemming means grouping the words*

*# after cleaning the data for further processing*

**def** stem(s):

fil**=**[]

**for** \_ **in** s:

**if** \_ **not** **in** stop:

s**=**(sno**.**stem(\_)**.**encode('utf8')) *# encoding the data into a clean file*

fil**.**append(s)

s**=**b' '**.**join(fil)

**return** s

**from** tkinter **import** **\***

**from** PIL **import** ImageTk, Image

**import** os

root **=** Tk()

*##### Title*

root**.**title("Autonomous Tagging Of Stackoverflow Questions")

*##### Image of stacoverflow logo*

img **=** ImageTk**.**PhotoImage(Image**.**open("index.png"))

panel **=** Label(root, image **=** img)

panel**.**image **=** img

panel**.**grid(row **=** 0, column **=** 0)

img1 **=** ImageTk**.**PhotoImage(Image**.**open("st.jpg"))

label1 **=** Label(root,image **=** img1)

label1**.**image **=** img1

label1**.**grid(row **=** 1, column **=** 0)

label2 **=** Label(root, text **=** "Ask Me a Question!!!",font **=** "Arial 20 bold", fg **=** 'black')

label2**.**grid(row **=** 2, column **=** 0)

*##### Text box to enter the question*

ques **=** StringVar()

quesEntered **=** Entry(root, width **=** 40, textvariable **=** ques,font**=**"Arial 18 bold")

quesEntered**.**grid(column **=** 0, row **=** 3, padx **=** 3, pady **=** 3)

*# function that takes the question from user*

*# predicts the tags to the questions*

ans **=** StringVar()

**def** pred():

t **=** ques**.**get()

l**=**[]

l**.**append(stem(clean(t)))

x**=**tfvectorizer**.**transform(l)

t**=**classifier**.**predict(x)

k**=**vectorizer**.**inverse\_transform(t)

res **=** re**.**sub('[^A-Za-z#+-]+', ' ', str(k[0]))

ans**.**set(res)

label3 **=** Entry(root,textvariable **=** ans, font **=** "Arial 20 bold")

label3**.**grid(column **=**0, row **=** 5, padx **=** 3, pady **=** 3)

*##### Button that predicts the tags by calling the function predict*

button **=** Button(root,text **=** "Predict Tag", font **=** "Arial 18 bold", command **=** pred)

button**.**grid(column**=** 0, row **=** 4, padx **=** 3, pady **=** 3)

root**.**mainloop()

In [ ]: