Intelligent Tagging and Recommendation System for StackOverflow Posts

Group Members: Sixuan Li, Wenyang Cao, Haoran Yang, Wenling Zhou, Jake Xiao

Github Repo:
https://github.com/educated-fool/stack-overflow-intelligent-tagging

Background & Problem Definition

Background:

- Stack Overflow: Major Q&A platform for programmers
- 18+ million questions on diverse topics
- Efficient tagging crucial for organization & discovery
- Manual tagging: Inconsistent & time-consuming.

Problem Definition:

- Automate tagging process
- Enhance post discoverability with accurate tag prediction
- Improve user experience with tag suggestions & similar posts.

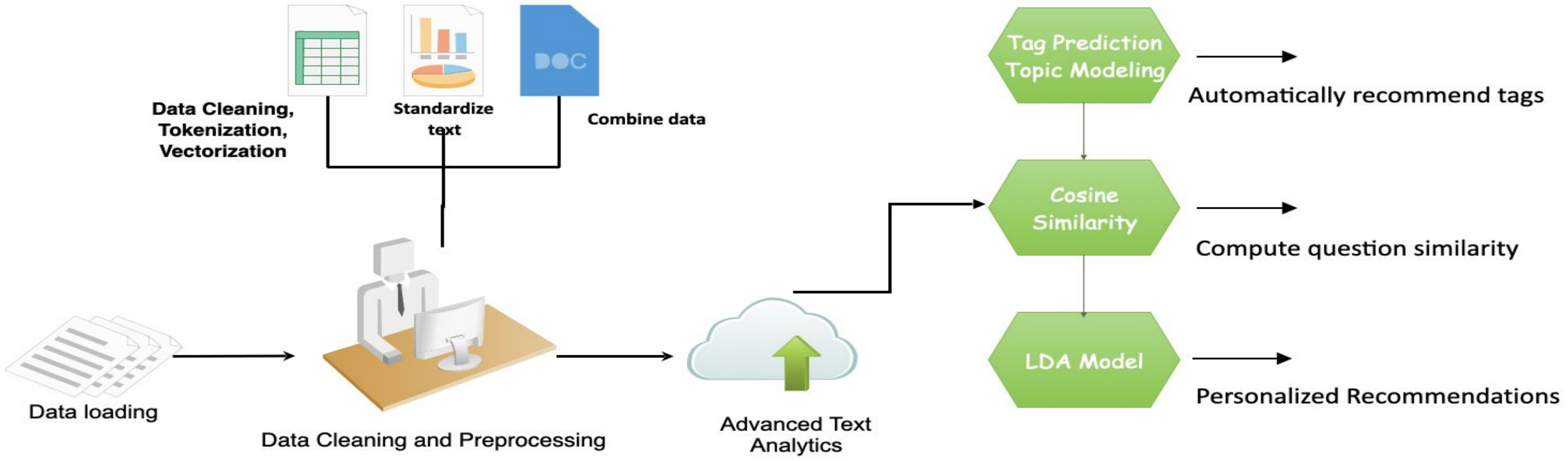
Data Source Details

Overview:

- StackSample: 10% of Stack Overflow programming Q&A website
- 1.26 million questions (Aug 2008 - Oct 2016)
- 37,000+ unique tags
- Total size: 3.6 GB

I. **Questions.csv**: Includes title, body, creation date, closed date, score, and owner ID for questions.
II. **Answers.csv**: Includes body, creation date, score, and owner ID for answers, linked to questions via ParentId.
III. **Tags.csv**: Contains pairs of IDs and Tags, with each question associated with one or more tags.

Design Structure


Data Cleaning,
Tokenization,
Vectorization
Standardize text
DOC
Combine data
Data loading
Data Cleaning and Preprocessing
Advanced Text Analytics
Tag Prediction Topic Modeling
Automatically recommend tags
Cosine Similarity
Compute question similarity
LDA Model
Personalized Recommendations

These three techniques work together to improve our system from different angles. First, we preprocess and vectorize the text, converting it into high-dimensional feature vectors. Then, we use the **Tag Prediction Model** to automatically generate tags. Using **Cosine Similarity**, we compute the similarity between the new question and existing questions to recommend similar ones. Finally, we apply **LDA topic modeling** to analyze the thematic distribution of questions, further enhancing the quality of tag recommendations. Combined, these techniques provide a more intelligent, accurate, and comprehensive tag and question recommendation system, improving user experience and overall system performance.

COLUMBIA SPS

Design Choices and Rationale

Tag Prediction Modeling

- ❑ Text Preprocessing
- ❑ High-Dimensional Feature Vectorization
- ❑ Evaluated Multiple Models

Cosine Similarity

- ❑ Measure of Textual Similarity
- ❑ Independence from Vector Magnitude
- ❑ Efficient Computation in High-Dimensional Spaces

LDA Topic Modeling

- ❑ Effective Preprocessing with CountVectorizer
- ❑ Efficient LDA Training with Gensim
- ❑ Interactive Visualization with pyLDAvis

Text Processing

Step 1: Input

- ❖ Load data from CSV files (questions, answers, tags)
- ❖ Merge relevant columns from questions and tags dataframes

Step 2: Tokenization

- ❖ Convert text to lowercase
- ❖ Tokenize text into words
- ❖ Handle multi-word expressions and remove non-alphabetic characters (except specific symbols)

Step 3: Stop Words & Lemmatization

- ❖ Remove common stop words
- ❖ Apply lemmatization to reduce words to their base form
- ❖ Filter tags to retain only the top 4,000 most common tags

Step 4: Vectorization

- ❖ Apply TF-IDF vectorization to titles and bodies
- ❖ Use separate vectorizers for titles and bodies

Tag Prediction Modeling

1. Multi-Model Approach

   Evaluated three models: Logistic Regression, XGBoost, and SGDClassifier.

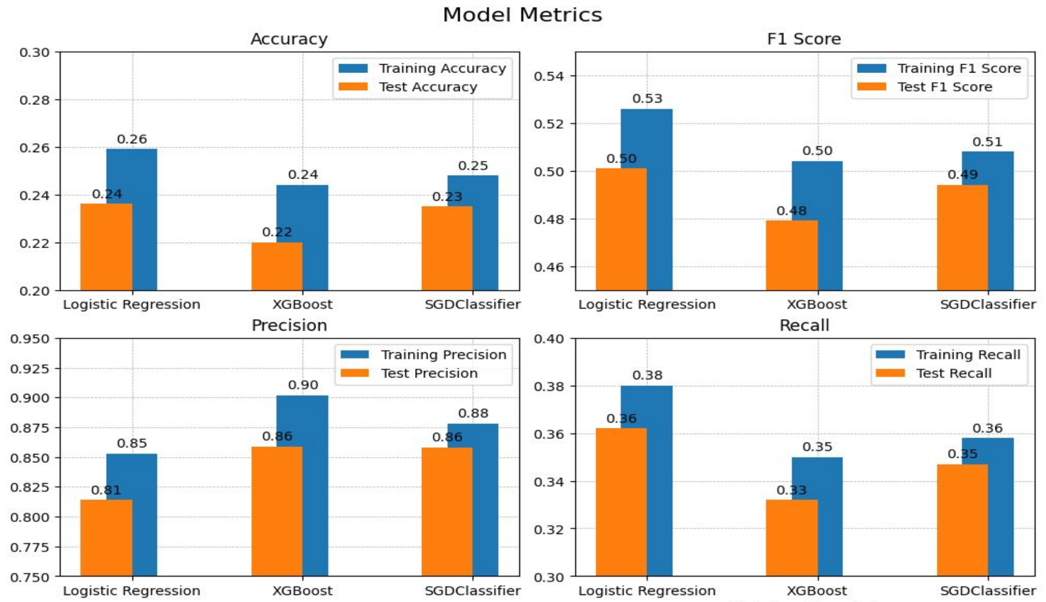
2. Model Comparison

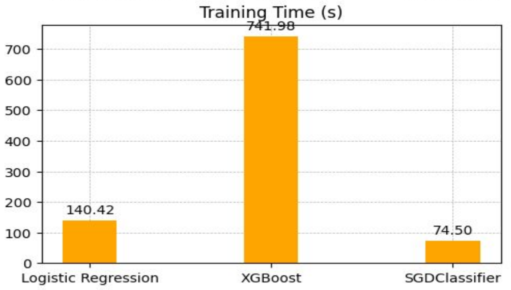
   Logistic Regression: Balanced metrics, moderate training time.
   XGBoost: High precision, lower recall, long training time.
   SGDClassifier: Competitive performance, fastest training time.

3. Limitation

   Costs: XGBoost training time is high, impractical for large datasets.

   Dataset: Limited to text and tags; excluding body content.





```
# Initialize and train the XGBoost
modelxgb_clf = xgb.XGBClassifier(
        n_estimators=100,
        learning_rate=0.1,
        max_depth=5,
        n_jobs=-1)
model = OneVsRestClassifier(xgb_clf)
```

Cosine Similarity

```python
df = spark.createDataFrame(questions_answers_df[['text']])

regex_tokenizer = RegexTokenizer(inputCol="text",
                                 outputCol="tokens", pattern="\\W")
df = regex_tokenizer.transform(df)

stopwords_remover = StopWordsRemover(inputCol="tokens",
                                     outputCol="tokens_sw_removed")
df = stopwords_remover.transform(df)

word2vec = Word2Vec(vectorSize=100, minCount=1,
                    inputCol="tokens_sw_removed", outputCol="wordvectors")
model = word2vec.fit(df)
df = model.transform(df)
```

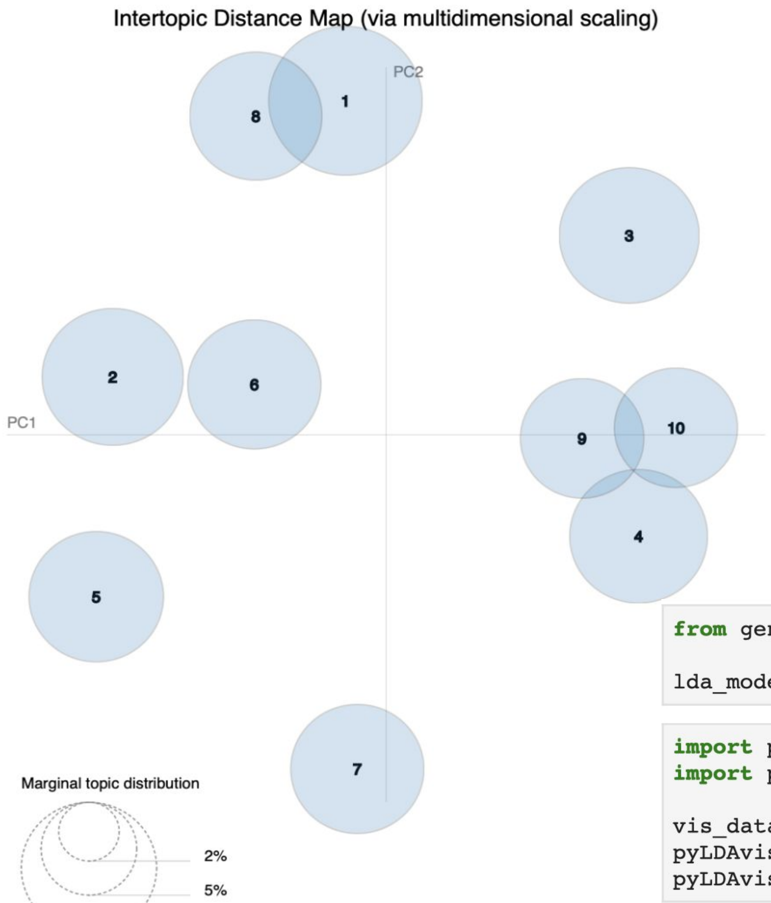
```python
def find_similar_articles(user_query):
    """
    input:
    user_query (str):

    output:
    None
    """

    query_df = spark.createDataFrame([(0, user_query)], ["id", "text"])
    query_df = regex_tokenizer.transform(query_df)
    query_df = stopwords_remover.transform(query_df)
    query_df = model.transform(query_df)

    query_vector = query_df.select("wordvectors").collect()[0][0]

    #cosine similarity
    def cos_sim(a, b):
        return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))

    # Calculate similarity scores
    articles = df.select("text", "wordvectors").collect()
    similarities = [
        (article["text"], article["text"], cos_sim(query_vector, article["wordvectors"]))
        for article in articles
    ]

    # Sort and print the top 5 similar articles
    similarities = sorted(similarities, key=lambda x: x[2], reverse=True) # rank
    for i, (text, full_text, similarity) in enumerate(similarities[:5]): #top 5
        print(f"Top {i + 1}:")
        print(f"text: {full_text}")
        print(f"similarity: {similarity:.4f}")
        print("\n" + "-"*50 + "\n")
```

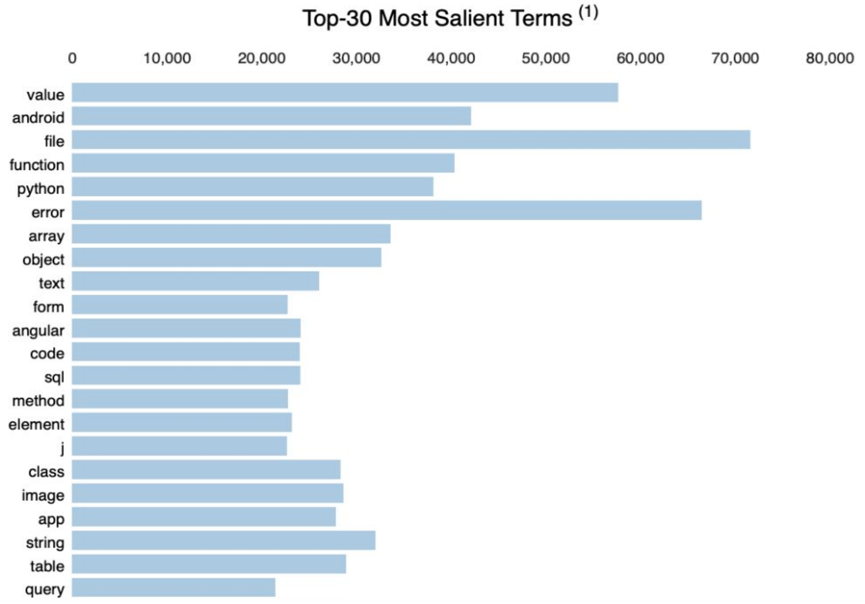
```python
find_similar_articles("python")
```

```
Top 1:
text: Is there an online interpreter for python 3?  Possible Duplicate  Python 3 online interpreter   shell   Where
can I find an online interpreter for Python 3   I m learning Python but can t install it at work where I d like to
do some practice  Thanks  Sorry to repeat the question  I can t bump earlier posts and was just hoping there is one
out there now   I don t know of a Python 3  and presumably you know about the browser app based on Python 2 5     B
ut if you re unable to install Python on your computer  I can point you to an interpreter configured to run from US
B keys   Portable Python  supports python 3
similarity: 0.8039

--------------------------------------------------


Top 2:
text: Import Errors with Python script run in R I have a Python program  which searches for an anomaly  First train
then test   Now I need to start this Python program from RStudio  I have read about system  python myfirstpythonfil
e py    but when I launch my Python program in this way I have import errors with numpy  scipy  etc  How can I laun
ch my Python program from RStudio   Having problems importing numpy or scipy suggests that your script is not runni
ng in the correct Python environment  It is possible to install multiple versions of Python on a computer  and whic
h one is run when you type python is determined by the PATH setting  It may be that when RStudio executes your scri
pt  via python myfirstpythonfile py  it is launching the wrong Python Ã¢   a version of Python on your computer tha
t does not have the numpy packages installed  You can test if this is the case by running the following on the comm
and line and seeing what it outputs  python  c  import sys  print sys executable    You can try the same from withi
n RStudio  system  python  c  import sys  print sys executable      If it gives a different result  you can pass th
e result of the first as an absolute path to python  changing  path to python for the correct value for your system
system   path to python myfirstpythonfile py    As you mention in the comments that you are actually trying to use
Python3  then you may be able to simply do the following from within RStudio  system  python3 myfirstpythonfile py
This will run your script using your installed Python3 and the associated packages libraries
similarity: 0.7941

--------------------------------------------------


Top 3:
text: Python 2 or Python 3 as the student's first language Which is more suited as the platform for a first course
in computing  Python 2 or Python 3   Reason for asking your opinion  Python 2 is used in the vast majority of insta
llations worlwide  but Python 3 is the coming thing   Python 2  Unfortunately library support for python 3 is disma
l
similarity: 0.7934

--------------------------------------------------
```

COLUMBIA SPS

LDA Topic Modeling



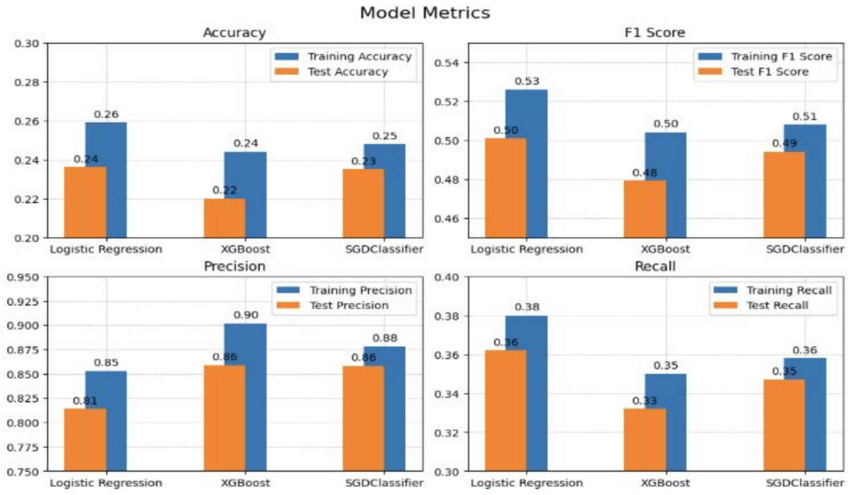


```
from gensim.models import ldamodel

lda_model = ldamodel.LdaModel(corpora, num_topics=10, id2word = dictionary, passes=50)
```

```
import pyLDAvis
import pyLDAvis.gensim_models as gensimvis

vis_data = gensimvis.prepare(lda_model, corpora, dictionary)
pyLDAvis.enable_notebook()
pyLDAvis.display(vis_data)
```

Evaluation Metrics

Tag Prediction Modeling :



Logistic Regression: Best for achieving a balance between precision and recall.
XGBoost: Ideal if precision is the most important metric, despite slightly lower recall.
SGDClassifier: Offers a good balance with high precision and reasonable recall, and is faster to train.

Cosine Similarity

Training Time: Efficient, with quick handling of large datasets due to PySpark's distributed processing.
Accuracy: Effectively captures semantic similarities between texts.
Precision and Recall: Not applicable; focus is on vector representation quality.
Cosine Similarity: High scores indicate strong performance in identifying similar texts.

LDA Topic Modeling :

Perplexity: -8.221812828255267

Coherence Score: 0.2888966970234389

This LDA model shows strong performance, with a ow Perplexity and a moderate Coherence Score, effectively capturing essential themes and producing meaningful topics.

COLUMBIA SPS

Data Collection and Preprocessing ● Load dataset: Retrieve data from Kaggle and read into DataFrames. ● Clean data: Remove HTML tags, special characters, handle missing values. ● Merge and preprocess: Combine columns, tokenize, lemmatize, handle multi-word expressions.	**Wenling Zhou, Sixuan Li**
Feature Engineering ● TF-IDF embeddings: Create TF-IDF matrix using TfidfVectorizer. ● Word2Vec embeddings: Train Word2Vec model for contextual relationships. ● Vectorize data: Prepare TF-IDF and Word2Vec vectors.	**Wenling Zhou, Haoran Yang**
Tag Prediction Model ● Split data: Divide into training and test sets, binarize tags. ● Train classifiers: Use Logistic Regression, Random Forest, SVM, XGBoost, SGDClassifier. ● Optimize and evaluate: Hyperparameter tuning, handle class imbalance, evaluate metrics.	**Haoran Yang, Sixuan Li**
Cosine Similarity for Similar Posts ● Compute similarity: Calculate cosine similarity between vectors. ● Recommend posts: Suggest top similar posts based on user queries. ● Develop system: Implement functions for input and output of similar posts.	**Jake Xiao**
Topic Modeling for Tag Recommendations ● LDA modeling: Generate topic clusters with LDA using Gensim. ● Visualize topics: Use pyLDAvis for topic interpretation. ● Recommend tags: Identify themes and suggest relevant tags.	**Wenyang Cao**