AUTOMATIC QUESTIONS TAGGING SYSTEM

2023

—

Natural language processing

Project stages

1 – read datasets from question table and tags table

2- merge two tables by id

3- filter data by checking the score of the questions

4-perform preprocessing steps on each text in the [questions, title, tags] columns.

5- perform feature extraction methods upon data to extract features by represent it in numerical form that computer can deal with

6- split dataset to train part and test part

7- generate classification models with train dataset and test the performance of the model through test data

8- represent results in plots

Project idea

Sites that are specifically designed to have questions and answers for their users like Quora and Stack overflow often request their users to submit five words along with the question so that they can be categorized easily. But sometimes users provide wrong tags which makes it difficult for other users to navigate through. Thus, they require an automatic question tagging system that can automatically identify correct and relevant tags for a question submitted by the user. So the project's main purpose is to generate classification models that can be used to classify the tags users enter to the site with questions and the given title so we will know if the tags are relevant or not. In this project we don't have to predict the answers, we only classify tags given.

Techniques explanation.

Project steps

1) Read data:

- read file “Tags.csv” to get tags table which consists of two columns[“id”, “Tag”].
- read file “Questions.csv” to get questions table which consists of 7
columns[“id” , “OwnerUserId”, “CreationDate” , “ClosedDate” , “Score” , “Title” , “Body”].

2)check missing values:

- check missing values in data .in case,there are any we Drop rows that contain any missing values ,but in our Dataset, there are 0 missing values in all rows.

3) Tags table organizing:

- each tag has givin id number(the unique identifier for each question) but it doesn’t have to be unique in tags table, so we join tags with same id in one one group according to its id .so that the resulted tags will be linked later to the question with its unique id.

4) merging tables:

- merge two tables by id of the question and tags’ group.

5) drop columns:

- we shall drop the unnecessery columns in the final table and these columns are [“OwnerUserId”, “CreationDate” , “ClosedDate”] as they won’t make difference on classification of the tags.

6) minimize the size of the data:

- we will minimize the size of data to increase the speed of models and avoid long time processing in models training and testing and perform text preprocessing. We do this by removing the the guestions with popularity(“score”) less than 5. We use low-quality or poorely-received questions.

…… now we go to text preprocessing part

text preprocessing:

preparation of features:

- We should modify texts to be ready for feature selection and modeling.

- ***Techniques used in questions and title columns "features":***

We have two features to prepare ---> title column , body column .

1- *Techniques applied on Title column*

- Remove special characters: remove punctuation at the text such as ['@', '!' , '?', '$' , '/', ...etc].
- Word tokenization: Apply tokenization to deal with smaller pieces (each single word in text)
- Remove stop words: remove unwanted words like 'in' , 'is' from the text as it's not adding any meaning.
- Apply POS to remove prepositions and unnecessary words

  PRP: Personal pronoun, PRP$: Possessive pronoun.
- Get the root:
  We can apply one of two methods for this step:
  - Stemming: text processing task in which you reduce words to their root.
  - Lemmtization: reduces words to their core meaning, but it will give you a complete English word that makes sense on its own.

Stemming gives more accuracy score than lemmatization in our models.

2- Techniques applied on Body column

- We will add one extra step for body texts then We apply all previous steps.
- We will apply htmltags completely from each text on body column ---> that is the extra step performed.

Preparation of label:

- We need to form classes of most popular tags to classify the questions through them later in model training.

So to do this we followed these steps:

1- Create single list containing all the possible tags in tags column

2- Get the most frequent tags by using freqdist function that creates a matrix of tags with frequency number that presents the times they appeared in column.

3- Create list that contains specific number of common tags which have the biggest appearance (frequent number) in the whole tags.

4- Filter the tags by this list so that any tag doesn't exist in the common tags list it will be removed.

5- The last stage of label preparation is encodeing as we have to convert label column from cateogrical to numerical to be easy to train models with it.
We used multilabelbinarizer that represents y-labels in binary matrix.it's one hot encoding method used in multi-label problems.

Since now label column is ready and data is completely preprocessed,we go the step of preparing features.

Features extraction stage:

We tried three feature extraction methods in our project

- Tf-idf
- bow "bag of words"
- Topic modeling

1- tf-idf "term frequency-inverse document frequency":

Captures importance of a word to a document. To calculate TF-IDF score for each word in document (questions and title) we calculate TF and IDF by these equations.

TF = term count in document/total terms in document

$$IDF = \log\left(\frac{total\ documents + 1}{documents\ containing\ the\ term + 1}\right)$$

TF-IDF score = TF X IDF

pros of its usage:

- It's suitable to the concept as we are trying to follow the algorithms of TF-IDF to achieve as we are trying to classify the questions by measuring the similarity between the words on text of the question with the specific tag.
- Simple and easy to use and calculate.
- It doesn't take too much time to process.
- It achieves the best accuracy score in testing models.

This method uses two parameters than can be manipulated to improve the accuracy score and these parameters are:

- **Max_df:** the maximum document frequency that a word can have before it is considered a stop word and removed.
- **Max_features:** the maximum features that can be created by the algorithm, so it's used to limit the number of features (words)

2- count vector: known as bag of words

- It tells us how many times each word can occur in questions and title.

Drawbacks of its usage in our models...

- it's taking very long time in processing the models
- it's giving accuracy less than tf-idf
- it cauuses some iterative models to not converge which means it's working accurately.

3- topic modeling:

- Topic modeling can be used to identify the most important words or phrases in a set of questions that are related to a particular topic. These features can then be used to train a machine learning model to automatically tag new questions based on their content.
  Steps to perform it .....

  - Apply tf-idf to the data of body and title with the same steps shown above.we do that to reduce the impact of words that occur frequently across all documents but may not carry much meaning.
  - Second step: the NMF "Non-Negative Matrix Factorization "algorithm is trained on TF-IDF matrix. NMF algorithm is matrix factorization technique that factorizes a non-negative matrix into two non-negative matrices, one representing the topics (out of words) and the other representing the distribution of topics in each document. The NMF algorithm is commonly used for topic "tags for our problem" modeling because it produces interpretable topics and can handle large datasets efficiently.

It has one hyperparameter that varies with the use case and it's num_topics.
Pros: it's good for the modeling and the dataset size .
Cons:the accuracy score of its usage is still less than TF-IDF.

---

models:

- we will use **Multiclass classification models:** classification task with more than two classes. Each sample can only be labelled as one class. So, the multiclass models we used are:

1- LabelPowerset classifier
2- Chains classifier
3- Binary rerlevence
4- One-vs-rest

We display five scores:

- **Accuracy_score:** computes the accuracy, either the fraction (default) or the count (normalize=False) of correct predictions.

- **Recall_score:** The recall is the ratio **tp / (tp + fn)** ----> tp (the number of true positives) and fn (the number of false negatives). The recall is intuitively the ability of the classifier to find all the positive samples.

- **Precision_score**: The precision is the ratio **tp / (tp+fp)** where tp (the number of true positives) and fp (the number of false positives). The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.

- **F1_score**: The F1 score can be interpreted as a **weighted average of the precision and recall**, where an F1 score reaches its best value at 1 and worst score at 0.

$$F1 = 2 * (precision * recall) / (precision + recall)$$

The average parameter in the recall_score, precision_score and f1_score functions is set to **'weighted'**, which means that the calculation is performed for each label and then averaged by the number of samples in each class. This is useful when there is an **imbalance** in the number of samples per class.

- **Hamming loss**: calculates the fraction of labels that are incorrectly predicted, the fraction of the wrong labels to the total number of labels.

Models training and testing :

We used four multi- label models

1- labelPowerset classifier:

- Label Powerset is a problem transformation method for multi-label classification, which transforms the multi-label problem into a multi-class problem by creating a new class for each

unique combination of label. it treats each combination of labels as a separate class and trains a multi-class classifier to predict the presence or absence of each combination.

- We use a base estimator to train the multi-label classifier. A base estimator is a single-label classifier that is trained on the individual labels of the multi-label dataset, and then combined to form a multi-label classifier.

- In this technique, the label space is partitioned into all possible combinations of labels, creating a power set of labels. Each binary classifier is then trained to predict whether an instance belongs to a particular label combination or not. The final prediction is obtained by combining the predictions of all binary classifiers.

We have too many choices foe base estimator but not all of them will satisfy the purpose of the model, so we must try them all to find out which one achieves the best result.
We got five options of single-label classifiers to use:

- **Linear svc" svm"**
- ✓ **Random forest**
- **Gradient boosting**
- **k- nearest neighbors**
- **logistic regression**
- **mlp**

explanition for each of the above five base

logistic regression :

- Logistic Regression: Logistic regression is a binary classification algorithm that models the probability of a binary output variable (0 or 1) as a function of one or more input variables. It works by fitting a logistic function to the training data to predict the probability of a binary output.

The logistic function is defined as:

- where is a linear combination of the input variables and model parameters:

$$z=\theta 0+\theta 1x1+\theta 2x2+...+\theta$$

The logistic regression algorithm learns the values of the model parameters.

k-nearest neighbors:

- KNN is a non-parametric supervised learning algorithm that can be used for classification and regression. It works by finding the k-nearest data points in the training set to a new input data point and predicting the output based on the majority class of those k-nearest neighbors.

- In KNN, the value of k is a hyperparameter that controls the number of neighbors to consider. A larger value of k reduces the effect of noise in the data but may also reduce the model's ability to capture local patterns.

random forest:

- _Random forest is an ensemble-based binary classification - algorithm that consists of multiple decision trees, each trained on a random subset of the training data. The algorithm aggregates the predictions of the decision trees to make a final prediction.

- Each decision tree is trained by recursively splitting the data based on the feature that maximizes the information gain or Gini impurity. The algorithm stops splitting when a stopping criterion is met, such as reaching a maximum depth or minimum number of samples.

- The random forest algorithm learns the values of the model parameters by training multiple decision trees on different subsets of the training data and aggregating their predictions.

Mlp:

- MLP is a neural network-based supervised learning algorithm that can be used for classification and regression. It consists of multiple layers of interconnected nodes that perform nonlinear transformations on the input data.

- In MLP, the input layer receives the input data, and the output layer produces the predicted output. The hidden layers

perform nonlinear transformations on the input data and help the model learn complex features.

SVM:

SVM is a binary classification algorithm that finds the hyperplane that maximizes the margin between the two classes in the training set. It works by mapping the input data into a high-dimensional feature space and finding the hyperplane that separates the two classes with the largest margin.

2-chains classifier:

- a machine learning method for problem transformation in multi-label classification. It combines the computational efficiency of the binary relevance method while still being able to take the label dependencies into account for classification.
- Method explanation: In this algorithm, the first binary classifier is trained to predict the first label, and then the output of the first classifier is fed as input to the second classifier along with the input features to predict the second label. This process is repeated for all labels until all labels have been predicted.

3- binary relevance:

- Binary relevance is arguably the most intuitive solution for learning from multi-label examples. It works by decomposing the multi-label learning task into several *independent* binary learning tasks (one per class label)

- We use a base estimator with it as well and that base estimator is svm single label classifier.

4-onevsrest classifier:

- The One-vs-Rest **(OvR)** strategy is a commonly used method for extending binary classification algorithms to multi-class classification problems. The OvR strategy involves training a separate **binary classifier** for each class, where the positive class is the class of interest, and the negative class is all other classes combined. It also needs a base estimator, but **it must be a binary classifier**.

- The **OneVsRestClassifier** algorithm is a meta-estimator that implements the OvR strategy for multi-label classification problems. The algorithm takes a base estimator as input, which is typically a binary classifier such as logistic regression or SVM. The base estimator is then trained on each label independently, treating it as the positive class and all other labels as the negative class. The resulting binary classifiers are then combined to form a multi-label classifier.

- We use logistic regression for The **OneVsRestClassifier**

algorithm.

- Binary relevance , one vs rest , chains classifiers only deal with binary base estimators such as svm or logistic regression.

Conclusion:

between all these classification methods the most efficient method is labelpowerset

---

The following tables will show the varying of values of scores by following factors:

- different methods usage
- different feature extraction methods and their hyperparameters
- different models used.

First model: **labelPowerset**

- ✓ **Using tf-idf**
- ✓ **Using stemming**

1- Svm:

Data size, max_features

(Size, max)	Accuracy	Precision	Recall_score	F1_score	Hamming_loss
(52418,20000)	0.5973864	0.738234807	0.673398117	0.71404199984	2.79397176650
(52418,40000)	0.601297214	0.7453516	0.6759793501	0.71918889672222	2.7413201068
(52418, 51426)	0.6056848	0.748139	0.6773458	0.7218622	2.71766501

2- K-Neighbors classifier: A non-parametric method that classifies new instances based on the closest training examples in feature space but can be sensitive to the choice of distance metric and the number of neighbors.

(Size, max)	Accuracy	Precision	Recall_score	F1_score	Hamming_loss
(52418,40000)	0.3864937	0.5433581	0.4332675	0.47621272	4.969095

3- Random forest: A tree-based ensemble method that can handle high-dimensional data and non-linear relationships between features.

(Size, max)	Accuracy	Precision	Recall_score	F1_score	Hamming_loss
(52418,20000)	0.6151278138	0.79102182444	0.645915578	0.7352297317	2.654330408
(52418,40000)	0.612075543	0.7866564201	0.641588217	0.73319738722	2.69744372
(52418, 51426)	0.612075543	0.7909206	0.64098086	0.735399445	2.6879053

4- MLP classifier:

MLPClassifier stands for Multi-layer Perceptron classifier which in the name itself connects to a Neural Network. Unlike other classification algorithms such as Support Vectors or Naive Bayes Classifier, MLPClassifier relies on an underlying Neural Network to perform the task of classification.

- **Takes very long time as it's an iterative algorithm that working on reduction of loss.**

Performed

dataset ---> 90605 x 30000

stared with **Iteration 1, loss = 3.54273385**

Iteration 2, loss = 2.33295325

Ended by **Iteration 163, loss = 0.02132096**

Iteration 164, loss = 0.02114236

Results in:

Accuracy score: 0.5927928922244909

Recall score: 0.7022358614642701 Precision score: 0.73655938320867
Hamming loss: 2.7415705535014623 F1 score: 0.7229287984963482

With reaching iteration 163 if we continued it may give higher accuracy, but it takes very long-time processing.

second model: chains classifier

- ✓ Using tf-idf
- ✓ Using stemming

(size,max_feat)	accuracy	precision	recall	F1-score	Hamming-loss
(52418,40000)	0.582315909	0.80356027	0.67726996	0.7486783	2.42579168
(52418,20000)	0.579072872	0.79383830	0.67954752	0.7495812	2.47004959

Third model: binary relevance

- ✓ Using tf-idf
- ✓ Using stemming

(size,max_feat)	accuracy	precision	recall	F1-score	Hamming-loss
(52418,40000)	0.5500763	0.822778648	0.6425751	0.733193807	2.4448683708
(52418,20000)	0.54998092	0.81570343	0.64697843	0.73240704	2.4635635

Fourth model: one vs rest

- ✓ Using tf-idf
- ✓ Using stemming

(size,max_feat)	accuracy	precision	recall	F1-score	Hamming-loss
(52418,40000)	0.612075543	0.78665642	0.641588217	0.73319738	2.6974437
(52418,20000)	0.4849294	0.8671163	0.54456422	0.69347204	2.65013353

The visualization of the highest results

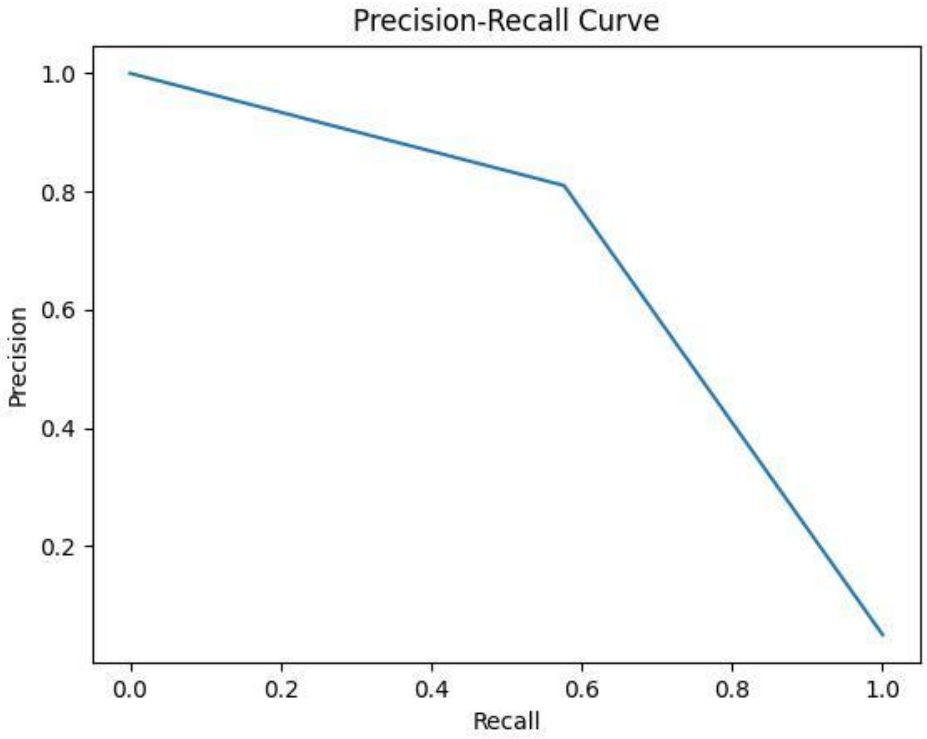
a. Report
b. Precision_recall_curve
c. Confusion matrix

In highest score case

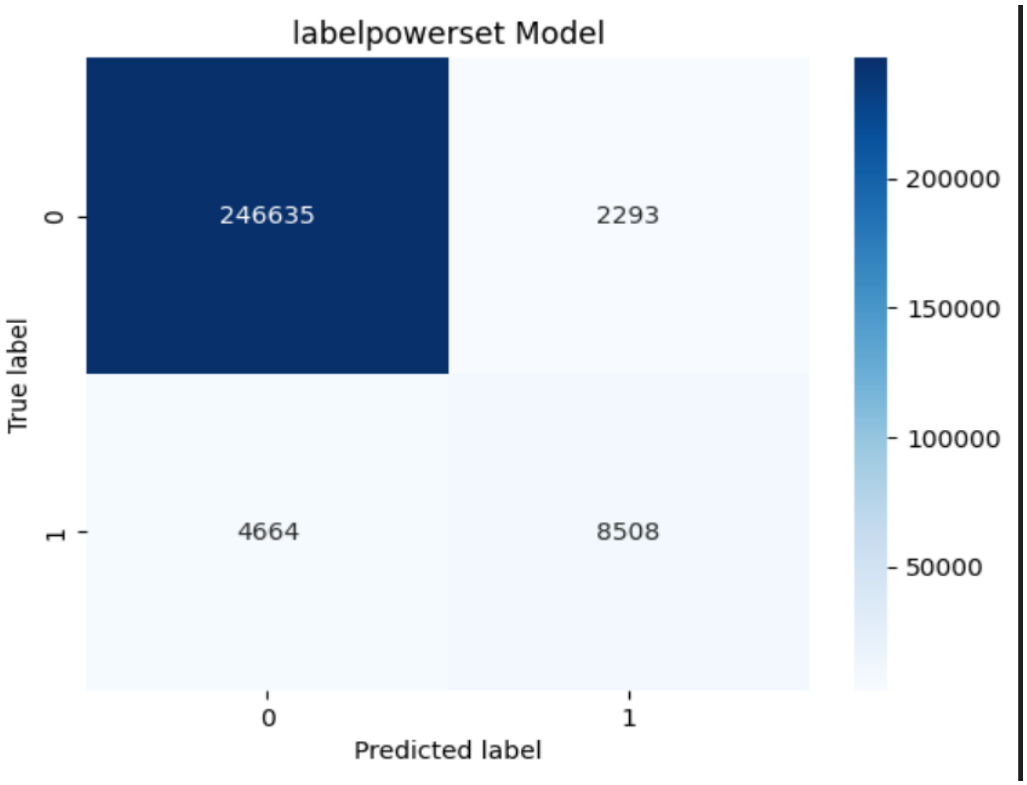
1- Label powerset using tf-idf -----> features = 20000
Using stemming

(Report of scores of labelpowerset with random forest)

Accuracy:61.5



(Precision_recall_curve of labelpowerset with random forest)



(Confusion matrix of labelpowerset model)