SAN JOSÉ STATE UNIVERSITY

CMPE 239 - Data Mining
Prof: David Anastasiu

Project Report
Multilabel Classification on Stack Overflow Data

Submitted by:

Anudeep Rentala : 011417666

Jayam Malviya : 011435567

Navneet Jain : 011419291

Problem Statement :-

In this problem, we are provided the titles, text, and tags of Stack Exchange questions from six different sites. We have challenged ourselves for tag predictions on untagged questions. The given Data set contains data in six different csv files from different categories namely biology, cooking, crypto, diy, robotics and travel. And a test data csv file. The data format for each document is Title, Text and Tags. For test data, we are given only Title and Text, no tags.

Method 1 : One vs Rest Classifier with Perceptron

Pre-processing steps:

- Removal of numbers, digits, special characters and non-alphabetical characters from the Content.
- Removal of stopwords of the NLTK library from Title and Content columns was helpful to remove unimportant words
- Removal of tags content for any extra spaces surrounding them
- Removal of HTML Tags such from the Title, Content and the Tags part of each document for each csv file.
- Removal of URL's helped improve feature quality and removed features that do not contribute to the model
- Merging of Title & Text as Content helped as they have tags assigned to them as a whole and having a combination of title and text helped improve feature list
- Splitting of Content into separate words that will become the features of the each document.
- Removal of words with lower character lengths such as 'a', 'be', 'is' or in other words, any word with length less than 3 are removed to improve the quality of features
- After doing all these cleanups we can drop the title and content columns from the original data frame and can return only Content and tags for further processing.

Tag generation from same set :

- First, we read the set data sets - 'crypto.csv', 'cooking.csv', 'robotics.csv', 'travel.csv' and aggregate them into a single training data set

- After cleaning the data in the steps stated above, we generated the feature list by taking a common set intersection of the feature list in training and test data. By doing so, we are eliminating features that would not contribute to the prediction as they would be redundant.

- Similarly, we've taken a set intersection of the tags in training and validation set. As our test data did not have any tags associated with them, there was no way to truly test our results against the test data

- The above steps helps us retrieve the predicted labels back from the MultiLabelBinarizer using the inverse Transform method, as there are equal number of features in training and testing data set.

Model:

- We have used MultiLabelBinarizer document for transforming the "tags" of the document into sparse matrix. It also has an interesting and helpful feature such as inverse_transformation, that is helpful in mapping a resultant matrix to the indices of tags that the classifier has predicted

- We also need a sparse matrices or CSR representation of the training and testing Content. We have used CountVectorizer and a TfidfTransformer or a TfidfVectorizer for various situations and found the same result.

- For classification we have used Perceptron. The way it works is but adjusting weights and biases as the training data is encountered. It has been known to perform well on linearly separable data

- One vs Rest Classifier is the transformer that we used, and the way it works is by fitting one classifier against all other classes. The classifier itself was something that we've toggled and experimented with and found varying results.

Learnings and Outcomes:

- Stochastic Gradient Descent (SGDClassifier) was one of the estimators we've tried with OneVsRestClassifier. The results were poor for several metrics such as accuracy and f-1 score. By varying parameters such as loss parameters to hinge over log, elasticnet penalty, higher iterations and smaller learning rate helped improve the results but only marginally. By having balanced classes, recall dramatically improved but resulted in very poor precision. Hence, experiments with SGDClassifier did not result in much success
- Logistic Regression seemed to perform better in terms of speed but resulted in low accuracy and f-1 score. Overall, it seemed ill suited for the problem and varying different parameters really did not help either.
- Perceptron - Although, perceptrons underlying equivalent is an SGDClassifier, by setting increasing the number of iterations and penalty to 'elasticnet', we were able to achieve a fair balance between accuracy, precision and recall.

Method 2 : Binary Relevance problem transformer with Stochastic Gradient Descent Classifier

Preprocessing of CSV files includes below cleaning procedures:

- Removal of unnecessary HTML Tags such as <p>, <li> <ui> etc from the Title, Content and the Tags part of each document for each csv file.
- Striping of Tags content for any extra spaces surrounding them.
- Merging of Title & Text as Content : As both Title and Content are particular to each document and they have tags assigned to them as a whole that is why I am merging these two and treating them as a single attribute called as Content.
- Splitting of Content into separate words that will become the features of the each document.
- Removal of Smileys and Emoticons from the Content, these were present in many documents along with the questions asked.
- Removal of URL's : Many questions included urls pointing to some resource online, I had to remove these too as they don't have any relevance for the question in training context.
- Removal of numbers and digits from the Content.
- Removal of Stopwords: I have used NLTK package to remove english stop words from the document so that we don't carry unimportant words with each of our document.
- After doing all these cleanups I finally dropped the old title & content columns from the original data frame and returned only Content and tags for further processing.

Data Splitting for Cross Validation and Tags from same set :

- First, we have ignored the documents that contain the tags that have appeared in only one document. i.e the documents that contains the tags which have document frequency as 1 are removed from the training data set.

- After the initial cleaning, we have splitted the training data set into two parts with 90-10 split, means 90% of the data will be used for training the model and 10% of data will be used for testing the model & get predictions. In this way I amusing training data set for cross validation.

- Also, to keep the number of features same for both the training and testing data in tags. We take the set of unique tags of the train data set y_train_Set and set of unique tags of the testing data set y_test_Set and then I have taken the intersection of these tag sets. This intersection tags set is then used to remove any tag from training & testing set that that are not present in the intersection. In this way, I have made sure that the tags that are being sent to the model for training and testing both have tags from same set and there is no such tag found in test data that is not present in the training data.

- The above step is also enables us to get the predicted labels back from the multi labelBinarizer using the inverse Transform method. As there are equal number of

features in training and testing data set.

Training Model / Pipeline :

- First, I have used CountVectorizer for the document “text”, to convert the text into a sparse matrix.

- For transforming the “tags” of the document I have used MultiLabelBinarizer document to convert the tags into sparse matrix. MultiLabelBinarizer is also useful at the end of our prediction when I want to get the labels back from the prediction matrix results.

- Binary Relevance transforms the problem the multilabel classification problem into a single label classification problem. If there are T tags for classification, Binary Relevance creates T data sets, one for each label and then trains single label classifier on each data set. Thus, one classifier answers only in Binary.

Observations :

- Huge, Very sparse and High Dimension data with almost 126000 unique words and 4200 unique tags.

- I have found this pipeline to work fairly well for small size of data, and have received accuracy score ranging between 0.95 - 1.00. But this is only valid for small data. When I switch to the whole data files that are provided as part of the data set we saw dramatic decrease in the accuracy. On the actual data set the accuracy score was found to be ranging between 0.07 - 0.1, which is low as compared to the score on small data.

- I have also tried other problem transformer and classifier like OneVsRest with Logistic Regression and OneVsRest with Stochastic Gradient Descent but didn’t found any improvements in the score for full data.

- Another difficulty was the time required for processing full data. I have found our programs getting killed when ran for full data. And if not killed they would easily run for hours before getting terminated by giving “MemoryError” on the console. Thus I had to limit ourselves to only 4 files from the given 6 csv files in the data set. This was one of the major challenge.

- The Predicted results are written back to files as the output of classification pipeline. The output file contains the timestamp of the results the method that was used in classification and the calculated score.

Method 3:- MultiLabel Knn Classifier

This problem includes depicting the tags for the stack exchange questions. The input training data includes files with title and content for each document and some tags related to each document. The test document includes title and content for each document and we need to predict the tags for each document.
My approach to this problem includes Preprocessing of data, Dimensionality Reduction and Classification of the tags for the test document.

1. Preprocessing :-

In preprocessing step, I have used various techniques to clean the data from the input file. After reading the data from the input file, I store the data in a data frame with the followings headings i.e. title, content and the tags. The title and contents will be merged together under one column named text. After reading the data, I applied the following functions:-

a. removehtmlTags

This function takes each document as input and removes the HTML tags from the document. This function returns a plain text document which can be used for further pre-processing.

```
def removehtmlTags(input):

        r"""This function removes the HTML tags from the data and converts it into plain text format"""

        cleanr = re.compile('<.*?>')

        cleantext = re.sub(cleanr, '', input)

        return cleantext
```

b. removestopword

This function removes the stop words from the documents which are passed as input parameters. In this function I have used NLTK corpus list of English stop words and appended my own list of stop words for better pruning of data.

```
def removestopword(document):

        r"""This function removes the stop words from the dataset basically reduces the dimensionality"""

        text = ' '.join([word for word in document.strip().lower().split() if word not in cachedStopWords])

        return text
```

c. filterLen

This function removes the words with the length passed in the input parameter. It takes document and minimum length of the words as input and returns a document after removing the words with length less than or equal to minimum length.

```
def filterLen(document, minlen):
```

```
    r"""This function filters the words with small length as that are not significant in tags classification"""
    text=' '.join(word for word in document.split() if len(word)>=minlen)
    return text
```

d. createWordSet

This function takes the document as an input and extracts all the unique words from the document and saves it into a set. This set will be used as dimensions while creating the CSR matrix for the train data.

```
def createWordSet(document):
    r"""This function takes documents as input and return a unique set of words from the document."""
    for w in document:
    doc = w.split()
    for x in doc:
    if x not in dataSetWords:
    dataSetWords.add(x)
    return
```

e. createTagSet

This function takes the document as an input and extracts all the unique words from the document and saves it into a set. This set will be used as dimensions while creating the CSR matrix for the train data.

```
def createTagSet(document):
    r"""This function will take tag documents as input and return a unique set of tags which can be used tp train the
    alorithm."""
    for w in document:
    doc = w.split()
    for x in doc:
    if x not in dataSetLabels:
    dataSetLabels.add(x)
    return
```

2. Dimensionality Reduction

In dimensionality reduction step, I tried using various algorithms like PCA and TruncatedSVD but it reduced my accuracy score. The below function will take CSR matrix with n dimensions and

truncate it to m dimensions as mentioned while creating the object for TruncatedSVD.

```
def svd_dr(X):
        r"""This is dimensionality reduction method which uses the TruncatedSVD approach to reduce the dimensions."""
        print('Old Shape : {}'.format(X.shape))
        svd = TruncatedSVD(n_components=35000, n_iter=5, random_state=42)
        # svd = PCA(copy=True, iterated_power='auto', n_components=1000, random_state=None,svd_solver='full', tol=0.0, whiten=False)
        svd.fit(X)
        print(svd.explained_variance_ratio_)
        X_new = svd.fit_transform(X)
        print('New Shape : {}'.format(X_new.shape))
        return X_new
```

3. Classification of Tags

In this part of my algorithm I classify the test data and assign tags to each of the document. From the train data and test data, I will create a CSR matrix with the train document and training word set. CSR creation on train data is done using TfidfVectorizer. The fit_transform function will give me a sparse matrix.

```
tf = TfidfVectorizer(norm='l2', vocabulary=list(dataSetWords))
training_M = tf.fit_transform(train_data_frame["text"])
testing_M = tf.fit_transform(test_data_frame["text"])
```

From the train data tags, I will create a CSR matrix with train tags and tags word set. For CSR creation I used MultiLabelBinarizer class from sklearn preprocessing library. The fit_transform function of this class will give me a binary CSR matrix.

```
mlb = preprocessing.MultiLabelBinarizer(classes=list(dataSetLabels))
Y_train = mlb.fit_transform(tagsData)
```

For the purpose of classification, I have used MLKNN Classifier from skmultilearn library. This classifier takes number of neighbors that need to be considered for classification as input parameter while initialization. The fit function takes the train data CSR matrix and tag data CSR matrix. This will train the classifier. The predict function of this classifier will then take test data CSR as input and predict the tags for the corresponding input. The output will be a sparse binary matrix which can be converted to dense matrix and traversed using rows and columns. The output from the traversal can be matched with the unique tag list and thus tags can be predicted.

```
mlk = mlknn.MLkNN(k=10, s=0.0, ignore_first_neighbours=0)

mlk.fit(training_M, Y_train)

output = mlk.predict(testing_M)  #Output will be a sparse binary matrix
```

4. Challenges

Overall it was a very challenging problem of classification and below are few challenges which I faced :-

a. Data Dimensionality

The train data provided for the classification problem had high dimensionality and thus filtering the unique dimensionality set was a challenge. Even after removing the unwanted dimensions and noise from the data set, still the dimensions are too largeto be handled.

b. Low computer memory

Everytime I took the whole data set, my laptop ran out of memory space and the execution is stopped. So I ran the algorithm on a small data set and got less accuracy.

c. Classification algorithm

While solving this classification problem, I came across many classification algorithms and each has its own pros and cons. Few classification algorithms have very less documentation to get the information. For this problem of multi label classification, I tried the following classification algorithms :- MALARMfast, MultiOutputClassifier and RandomForestClassfier. They gave very less accuracy and hence I shifted to MLKNN Classifier which gave me an accuracy score of around 20% on a small data set.

References: https://www.kaggle.com/c/transfer-learning-on-stack-exchange-tags