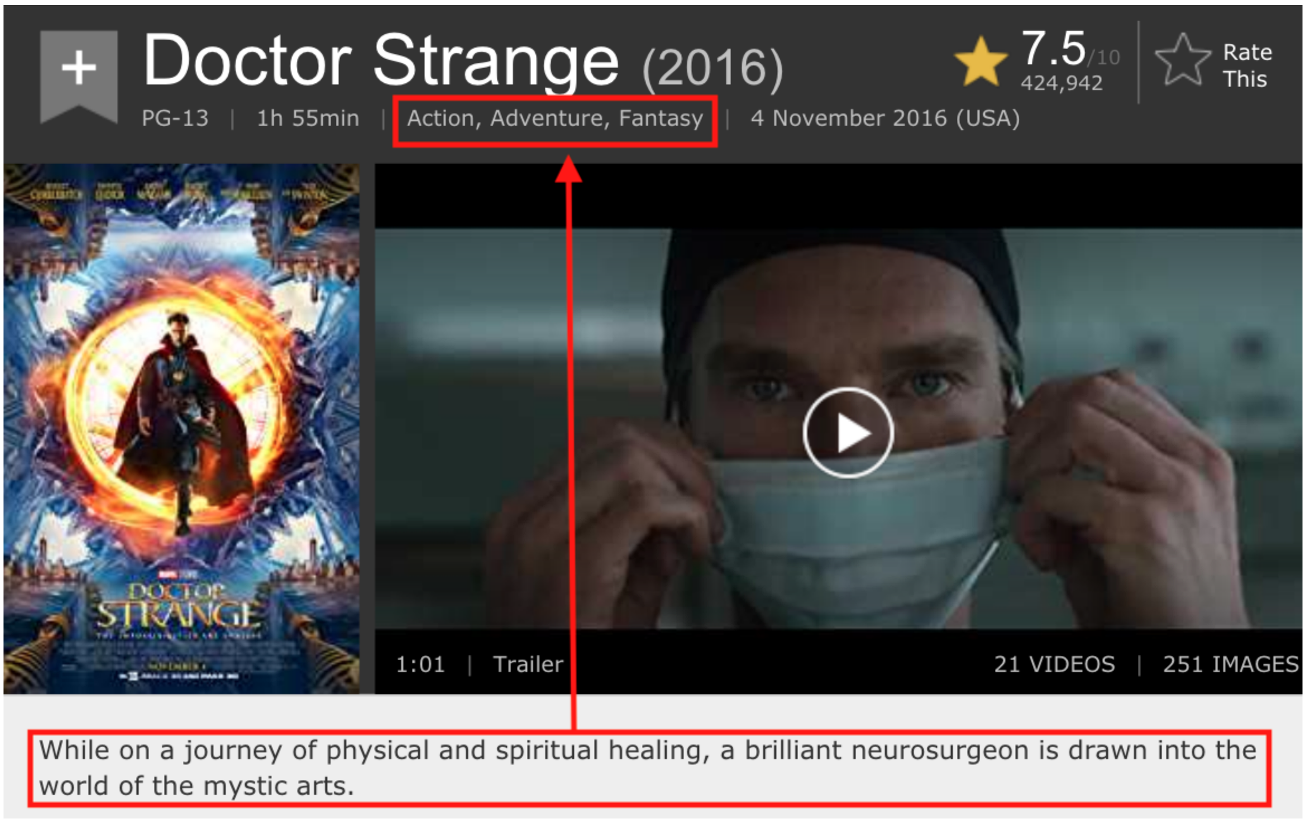
Deep dive into multi-label classification..! (With detailed Case Study)

Toxic-comments classification.

Kartik Nooney Follow
Jun 8, 2018 · 12 min read

. . .



While on a journey of physical and spiritual healing, a brilliant neurosurgeon is drawn into the world of the mystic arts.

Fig-1: Multi-Label Classification to finde genre based on plot summary.

. . .

> *With continuous increase in available data, there is a pressing need to organize it and modern classification problems often involve the prediction of multiple labels simultaneously associated with a single instance.*

> *Known as Multi-Label Classification, it is one such task which is omnipresent in many real world problems.*

> *In this project, using a Kaggle problem as example, we explore different aspects of multi-label classification.*

> **DISCLAIMER FROM THE DATA SOURCE:** the dataset contains text that may be considered profane, vulgar, or offensive.

. . .

Bird's-eye view of the project:

- **Part-1:** Overview of multi-label classification.
- **Part-2:** Problem definition & evaluation metrics.
- **Part-3:** Exploratory data analysis *(EDA)*.
- **Part-4:** Data pre-processing.
- **Part-5:** Multi-label classification techniques.

. . .

Part-1: Overview of Multi-Label Classification:

- Multi-label classification originated from the investigation of text categorisation problem, where each document may belong to several predefined topics simultaneously.
- Multi-label classification of textual data is an important problem. Examples range from news articles to emails. For instance, this can be employed to find the genres that a movie belongs to, based on the summary of its plot.

Harry Potter
AND THE
SORCERER'S
STONE

```
Horror: 0.02%
Romance: 0.02%
Adventure: 99.96%
Documentary: 0.0%
```

Fig-2: Multi-label classification to find genres based on movie posters.

Or multi-label classification of genres based on movie posters. *(This enters the realm of computer vision.)*

In multi-label classification, the training set is composed of instances each associated with a set of labels, and the task is to predict the label sets of unseen instances through analyzing training instances with known label sets.

Difference between multi-class classification & multi-label classification is that in multi-class problems the classes are mutually exclusive, whereas for multi-label problems each label represents a different classification task, but the tasks are somehow related.

For example, *multi-class classification* makes the assumption that each sample is assigned to one and only one label: a fruit can be either an apple or a pear but not both at the same time. Whereas, an instance of *multi-label classification* can be that a text might be about any of religion, politics, finance or education at the same time or none of these.

. . .

Part-2: Problem Definition & Evaluation Metrics:

Problem Definition:

- Toxic comment classification is a multi-label text classification problem with a highly imbalanced dataset.
- We're challenged to build a multi-labeld model that's capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate. We need to create a model which predicts a probability of each type of toxicity for each comment.
- Kaggle link to this problem can be found *here*.

Evaluation Metrics:

> ***Note:*** *Initially evaluation metric in the Kaggle challenge was* Log-Loss*, which was later changed to* AUC. *But in this post we throw light on other evaluation metrics as well.*

- The evaluation measures for single-label are usually different than for multi-label. Here in single-label classfication we use simple metrics such as precision, recall, accuracy, etc,. Say, in single-label classification, accuracy is just:

$$\frac{1}{N}\sum_{i=1}^{N}\mathcal{I}[\hat{y}^{(i)} = y^{(i)}]$$

Fig-3: Accuracy in single-label classification

- In multi-label classification, a misclassification is no longer a hard wrong or right. A prediction containing a subset of the actual classes should be considered better than a prediction that contains none of them, i.e., predicting two of the three labels correctly this is better than predicting no labels at all.

Micro-averaging & Macro-averaging (Label based measures):

To measure a multi-class classifier we have to average out the classes somehow. There are two different methods of doing this called *micro-averaging* and *macro-averaging*.

In *micro-averaging* all TPs, TNs, FPs and FNs for each class are summed up and then the average is taken.

- Microaveraging Precision $Prc^{micro}(D) = \frac{\sum_{c_i \in \mathcal{C}} TPs(c_i)}{\sum_{c_i \in \mathcal{C}} TPs(c_i) + FPs(c_i)}$
- Microaveraging Recall $Rcl^{micro}(D) = \frac{\sum_{c_i \in \mathcal{C}} TPs(c_i)}{\sum_{c_i \in \mathcal{C}} TPs(c_i) + FNs(c_i)}$

Fig-4: Micro-Averaging

In *micro-averaging* method, you sum up the individual true positives, false positives, and false negatives of the system for different sets and the apply them. And the micro-average F1-Score will be simply the harmonic mean of above two equations.

Macro-averaging is straight forward. We just take the average of the precision and recall of the system on different sets.

- Macroaveraging Precision $Prc^{macro}(D) = \frac{\sum_{c_i \in \mathcal{C}} Prc(D, c_i)}{|\mathcal{C}|}$
- Macroaveraging Recall $Rec^{macro}(D) = \frac{\sum_{c_i \in \mathcal{C}} Rcl(D, c_i)}{|\mathcal{C}|}$

Fig-5: Macro-Averaging

Macro-averaging method can be used when you want to know how the system performs overall across the sets of data. You should not come up with any specific decision with this average. On the other hand, *micro-averaging* can be a useful measure when your dataset varies in size.

Hamming-Loss (Example based measure):

In simplest of terms, *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.

$$\frac{1}{|N| \cdot |L|} \sum_{i=1}^{|N|} \sum_{j=1}^{|L|} \operatorname{xor}(y_{i,j}, z_{i,j}), \text{ where } y_{i,j} \text{ is the target and } z_{i,j} \text{ is the prediction.}$$

Fig-6: Hamming-Loss

Exact Match Ratio (Subset accuracy):

It is the most strict metric, indicating the percentage of samples that have all their labels classified correctly.

$$ExactMatchRatio, MR = \frac{1}{n} \sum_{i=1}^{n} I(Y_i = Z_i)$$

Fig-7: Exact Match Ratio

The disadvantage of this measure is that multi-class classification problems have a chance of being partially correct, but here we ignore those partially correct matches.

There is a function in *scikit-learn* which implements subset accuracy, called as **accuracy_score.**

> ***Note:*** *We will be using* ***accuracy_score*** *function to evaluate all our models in this project.*

. . .

Part-3: Exploratory Data Analysis (EDA):

> *Exploratory Data Analysis is one of the important steps in the data analysis process. Here, the focus is on making sense of the data in hand*

> *— things like formulating the correct questions to ask to your dataset, how to manipulate the data sources to get the required answers, and others.*

First let us import the necessary libraries.

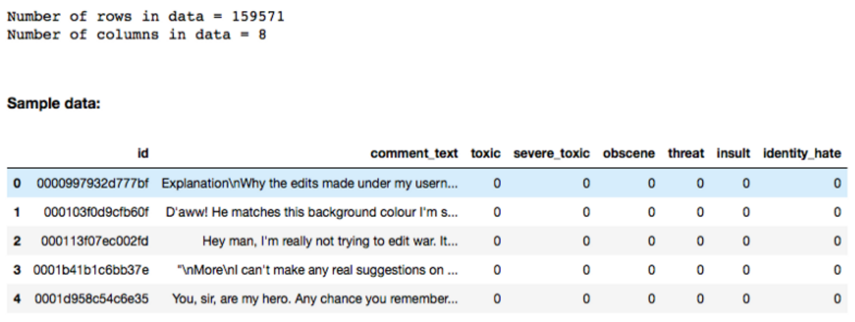
```
import os
import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Next we load the data from csv files into a pandas dataframe and check its attributes.

```
data_path = "/Users/kartik/Desktop/AAIC/Projects/jigsaw-
toxic-comment-classification-challenge/data/train.csv"

data_raw = pd.read_csv(data_path)

print("Number of rows in data =",data_raw.shape[0])
print("Number of columns in data =",data_raw.shape[1])
print("\n")
print("**Sample data:**")
data_raw.head()
```



```
Number of rows in data = 159571
Number of columns in data = 8
```

Sample data:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

Fig-8: Data Attributes

Now we count the number of comments under each label. (For detailed code, please refer to the GitHub link of this project.)

```
categories = list(data_raw.columns.values)
sns.set(font_scale = 2)
plt.figure(figsize=(15,8))

ax= sns.barplot(categories, data_raw.iloc[:,2:].sum().values)

plt.title("Comments in each category", fontsize=24)
plt.ylabel('Number of comments', fontsize=18)
plt.xlabel('Comment Type ', fontsize=18)

#adding the text labels
rects = ax.patches
labels = data_raw.iloc[:,2:].sum().values
for rect, label in zip(rects, labels):
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width()/2, height + 5,
label, ha='center', va='bottom', fontsize=18)

plt.show()
```

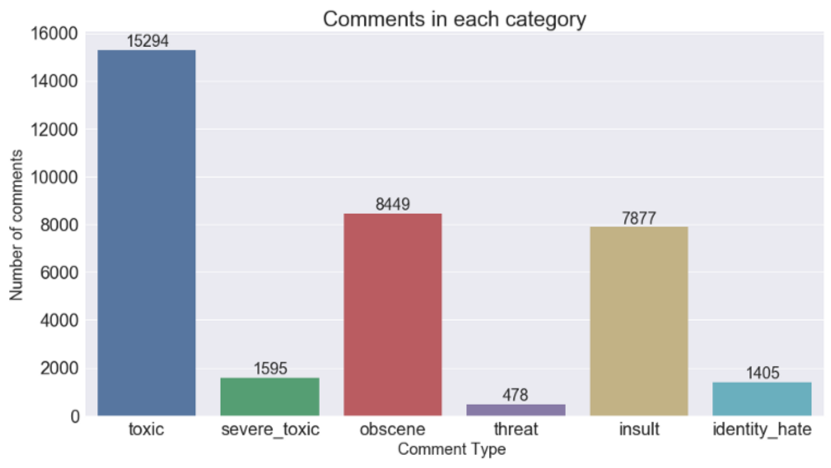


Fig-9: Count of comments under each label

Counting the number of comments having multiple labels.

```
rowSums = data_raw.iloc[:,2:].sum(axis=1)
multiLabel_counts = rowSums.value_counts()
multiLabel_counts = multiLabel_counts.iloc[1:]

sns.set(font_scale = 2)
plt.figure(figsize=(15,8))

ax = sns.barplot(multiLabel_counts.index,
multiLabel_counts.values)
```

```
plt.title("Comments having multiple labels ")
plt.ylabel('Number of comments', fontsize=18)
plt.xlabel('Number of labels', fontsize=18)

#adding the text labels
rects = ax.patches
labels = multiLabel_counts.values
for rect, label in zip(rects, labels):
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width()/2, height + 5,
label, ha='center', va='bottom')

plt.show()
```

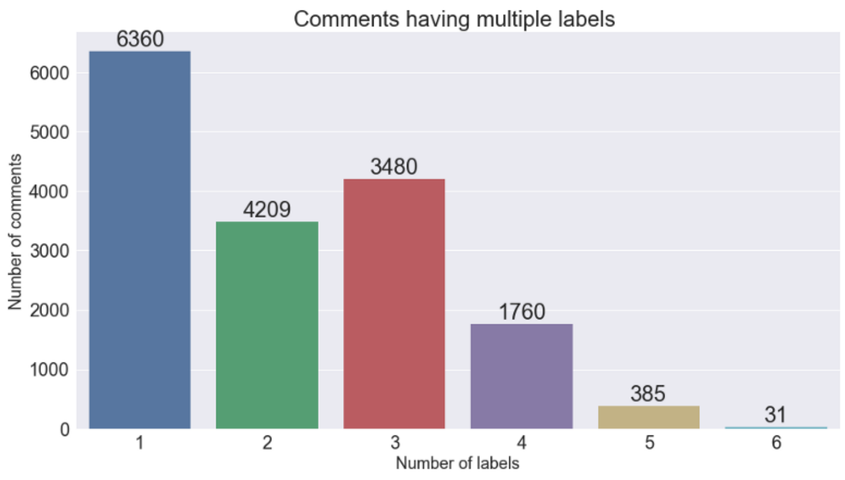


Fig-10: Count of comments with multiple labels.

WordCloud representation of most used words in each category of comments.

```
from wordcloud import WordCloud,STOPWORDS

plt.figure(figsize=(40,25))

# clean
subset = data_raw[data_raw.clean==True]
text = subset.comment_text.values
cloud_toxic = WordCloud(
                          stopwords=STOPWORDS,
                          background_color='black',
                          collocations=False,
                          width=2500,
                          height=1800
                         ).generate(" ".join(text))
plt.axis('off')
plt.title("Clean",fontsize=40)
plt.imshow(cloud_clean)

# Same code can be used to generate wordclouds of other
categories.
```

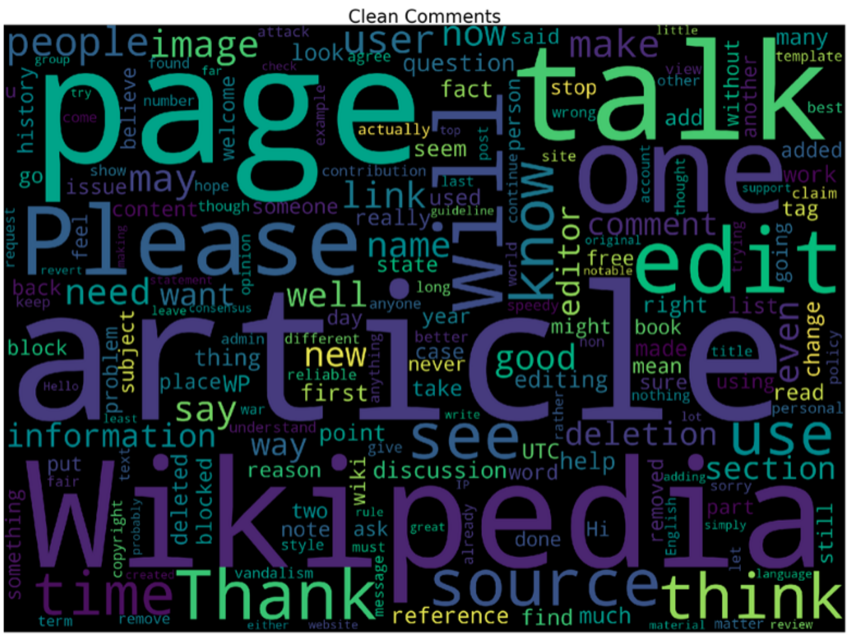


Fig-1: Word-cloud Representation of Clean Comments

. . .

Part-4: Data Pre-Processing:

- We first convert the comments to lower-case and then use custom made functions to remove *html-tags, punctuation and non-alphabetic characters* from the comments.

```
import nltk
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
import re
import sys
import warnings

data = data_raw

if not sys.warnoptions:
    warnings.simplefilter("ignore")


def cleanHtml(sentence):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', str(sentence))
    return cleantext


def cleanPunc(sentence): #function to clean the word of any
punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    cleaned = cleaned.strip()
    cleaned = cleaned.replace("\n"," ")
    return cleaned


def keepAlpha(sentence):
    alpha_sent = ""
    for word in sentence.split():
        alpha_word = re.sub('[^a-z A-Z]+', ' ', word)
        alpha_sent += alpha_word
        alpha_sent += " "
    alpha_sent = alpha_sent.strip()
    return alpha_sent


data['comment_text'] = data['comment_text'].str.lower()
data['comment_text'] = data['comment_text'].apply(cleanHtml)
data['comment_text'] = data['comment_text'].apply(cleanPunc)
data['comment_text'] = data['comment_text'].apply(keepAlpha)
```

Next we remove all the ***stop-words*** present in the comments using the default set of stop-words that can be downloaded from *NLTK* library. We also add few stop-words to the standard list.

Stop words are basically a set of commonly used words in any language, not just English. The reason why stop words are critical to many applications is that, if we remove the words that are very commonly used in a given language, we can focus on the important words instead.

```
stop_words = set(stopwords.words('english'))
stop_words.update(['zero','one','two','three','four','five','
six','seven','eight','nine','ten','may','also','across','amon
g','beside','however','yet','within'])
re_stop_words = re.compile(r"\b(" + "|".join(stop_words) +
")\\W", re.I)
def removeStopWords(sentence):
    global re_stop_words
    return re_stop_words.sub(" ", sentence)


data['comment_text'] =
data['comment_text'].apply(removeStopWords)
```

Next we do ***stemming***. There exist different kinds of stemming which basically transform words with roughly the same semantics to one standard form. For example, for amusing, amusement, and amused, the stem would be amus.

```
stemmer = SnowballStemmer("english")
def stemming(sentence):
    stemSentence = ""
    for word in sentence.split():
        stem = stemmer.stem(word)
        stemSentence += stem
        stemSentence += " "
    stemSentence = stemSentence.strip()
    return stemSentence

data['comment_text'] = data['comment_text'].apply(stemming)
```

After splitting the dataset into train & test sets, we want to summarize our comments and convert them into numerical vectors.

One technique is to pick the most frequently occurring terms (words with high ***term frequency*** or *tf*). However, the most frequent word is a less useful metric since some words like '*this*', '*a*' occur very frequently across all documents.

Hence, we also want a measure of how unique a word is i.e. how infrequently the word occurs across all documents (i***nverse document frequency*** or *idf*).

So, the product of tf & idf (***TF-IDF***) of a word gives a product of how frequent this word is in the document multiplied by how unique the word is w.r.t. the entire corpus of documents.

Words in the document with a high tfidf score occur frequently in the document and provide the most information about that specific document.

```
from sklearn.model_selection import train_test_split
```

```
train, test = train_test_split(data, random_state=42,
test_size=0.30, shuffle=True)

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(strip_accents='unicode',
analyzer='word', ngram_range=(1,3), norm='l2')
vectorizer.fit(train_text)
vectorizer.fit(test_text)

x_train = vectorizer.transform(train_text)
y_train = train.drop(labels = ['id','comment_text'], axis=1)

x_test = vectorizer.transform(test_text)
y_test = test.drop(labels = ['id','comment_text'], axis=1)
```

TF-IDF is easy to compute but its disadvantage is that it does not capture position in text, semantics, co-occurrences in different documents, etc.

. . .

Part-5: Multi-Label Classification Techniques:

> *Most traditional learning algorithms are developed for single-label classification problems. Therefore a lot of approaches in the literature transform the multi-label problem into multiple single-label problems, so that the existing single-label algorithms can be used.*

. . .

1. OneVsRest

- Traditional two-class and multi-class problems can both be cast into multi-label ones by restricting each instance to have only one label. On the other hand, the generality of multi-label problems inevitably makes it more difficult to learn. An intuitive approach to solving multi-label problem is to decompose it into multiple independent binary classification problems (one per category).
- In an "one-to-rest" strategy, one could build multiple

independent classifiers and, for an unseen instance, choose the class for which the confidence is maximized.

The main assumption here is that the labels are *mutually exclusive*. You do not consider any underlying correlation between the classes in this method.

For instance, it is more like asking simple questions, say, "*is the comment toxic or not*", "*is the comment threatening or not?*", etc. Also there might be an extensive case of overfitting here, since most of the comments are unlabeled, i,e., most of the comments are clean comments.

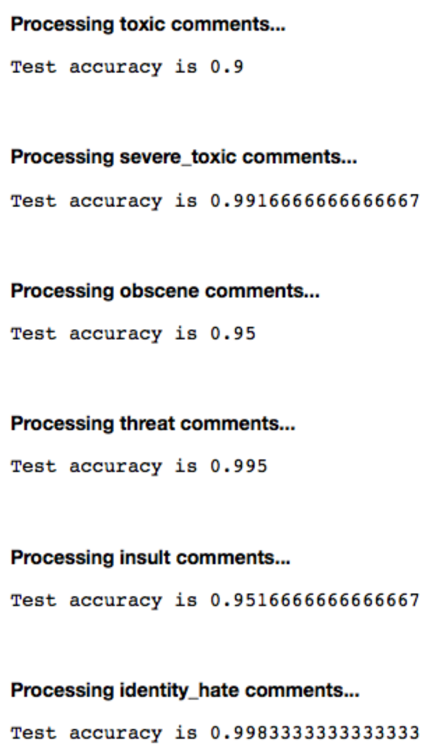
```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.multiclass import OneVsRestClassifier


# Using pipeline for applying logistic regression and one vs
rest classifier
LogReg_pipeline = Pipeline([
                ('clf',
OneVsRestClassifier(LogisticRegression(solver='sag'),
n_jobs=-1)),
            ])


for category in categories:
    print('**Processing {} comments...**'.format(category))

    # Training logistic regression model on train data
    LogReg_pipeline.fit(x_train, train[category])

    # calculating test accuracy
    prediction = LogReg_pipeline.predict(x_test)
    print('Test accuracy is
{}'.format(accuracy_score(test[category], prediction)))
    print("\n")
```



Processing toxic comments...

```
Test accuracy is 0.9
```

Processing severe_toxic comments...

```
Test accuracy is 0.9916666666666667
```

Processing obscene comments...

```
Test accuracy is 0.95
```

Processing threat comments...

```
Test accuracy is 0.995
```

Processing insult comments...

```
Test accuracy is 0.9516666666666667
```

Processing identity_hate comments...

```
Test accuracy is 0.9983333333333333
```

Fig-12: OneVsRest

. . .

2. Binary Relevance

- In this case an ensemble of single-label binary classifiers is trained, one for each class. Each classifier predicts either the membership or the non-membership of one class. The union of all classes that were predicted is taken as the multi-label output. This approach is popular because it is easy to implement, however it also ignores the possible correlations between class labels.
- In other words, if there's *q* labels, the binary relevance method create *q* new data sets from the images, one for each label and train single-label classifiers on each new data set. One classifier may answer yes/no to the question "does it contain trees?", thus the "binary" in "binary relevance". This is a simple approach but does not work well when there's dependencies between the labels.
- *OneVsRest & Binary Relevance* seem very much alike. If multiple classifiers in OneVsRest answer *"yes"* then you are back to the binary relevance scenario.

```
# using binary relevance
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.naive_bayes import GaussianNB

# initialize binary relevance multi-label classifier
# with a gaussian naive bayes base classifier
classifier = BinaryRelevance(GaussianNB())

# train
classifier.fit(x_train, y_train)

# predict
predictions = classifier.predict(x_test)

# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions))
```

Output:

```
Accuracy = 0.856666666667
```

. . .

3. Classifier Chains

- A chain of binary classifiers C0, C1, . . . , Cn is constructed, where a classifier Ci uses the predictions of all the classifier Cj , where j < i. This way the method, also called classifier chains (CC), can take into account label correlations.
- The total number of classifiers needed for this approach is equal to the number of classes, but the training of the classifiers is more involved.
- Following is an illustrated example with a classification problem of three categories {C1, C2, C3} chained in that order.

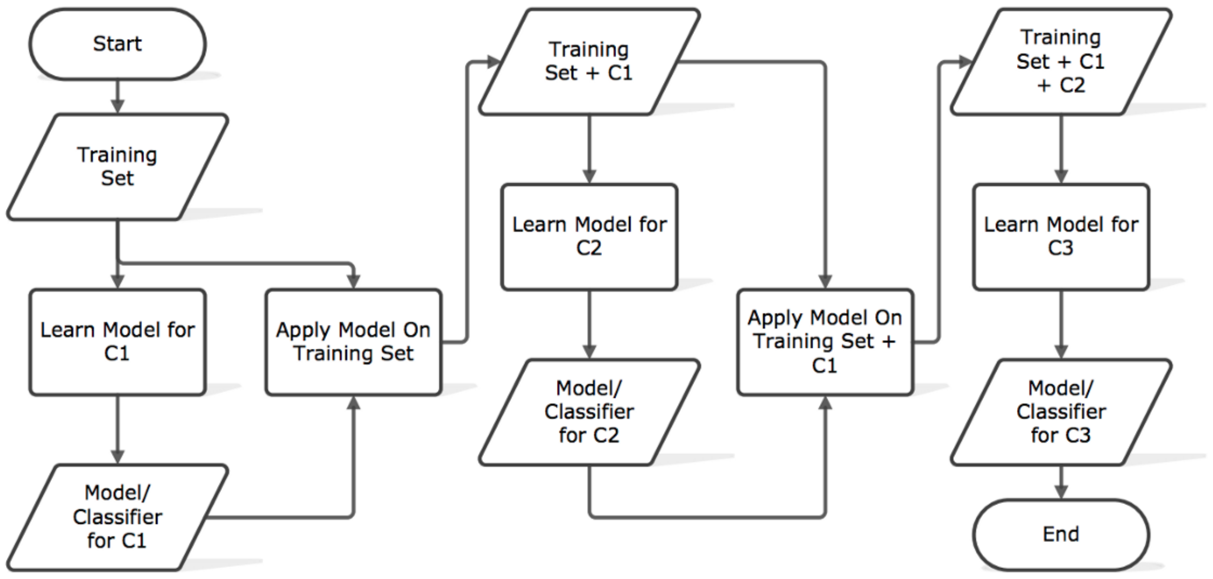


Fig-13: Classifier Chains

```
# using classifier chains
from skmultilearn.problem_transform import ClassifierChain
from sklearn.linear_model import LogisticRegression

# initialize classifier chains multi-label classifier
classifier = ClassifierChain(LogisticRegression())

# Training logistic regression model on train data
classifier.fit(x_train, y_train)

# predict
predictions = classifier.predict(x_test)

# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions))
print("\n")
```

Output:

```
Accuracy = 0.893333333333
```

. . .

4. Label Powerset

- • This approach does take possible correlations between class labels into account. More commonly this approach is called the label-powerset method, because it considers each member of the power set of labels in the training set as a single label.

This method needs worst case (2^|C|) classifiers, and has a high computational complexity.

However when the number of classes increases the number of distinct label combinations can grow exponentially. This easily leads to combinatorial explosion and thus computational infeasibility. Furthermore, some label combinations will have very few positive examples.

```
# using Label Powerset
from skmultilearn.problem_transform import LabelPowerset

# initialize label powerset multi-label classifier
classifier = LabelPowerset(LogisticRegression())

# train
classifier.fit(x_train, y_train)

# predict
predictions = classifier.predict(x_test)

# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions))
print("\n")
```

Output:

```
Accuracy = 0.893333333333
```

. . .

5. Adapted Algorithm

- Algorithm adaptation methods for multi-label classification concentrate on adapting single-label classification algorithms to the multi-label case usually by changes in cost/decision functions.

- Here we use a multi-label lazy learning approach named ***ML-KNN*** which is derived from the traditional K-nearest neighbor

(KNN) algorithm.

The `skmultilearn.adapt` module implements algorithm adaptation approaches to multi-label classification, including but not limited to ***ML-KNN.***

```
from skmultilearn.adapt import MLkNN
from scipy.sparse import csr_matrix, lil_matrix

classifier_new = MLkNN(k=10)

# Note that this classifier can throw up errors when handling
sparse matrices.

x_train = lil_matrix(x_train).toarray()
y_train = lil_matrix(y_train).toarray()
x_test = lil_matrix(x_test).toarray()

# train
classifier_new.fit(x_train, y_train)

# predict
predictions_new = classifier_new.predict(x_test)

# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions_new))
print("\n")
```

Output:

```
Accuracy = 0.88166666667
```

. . .

Conclusion:

Results:

- There are two main methods for tackling a multi-label classification problem: **problem transformation methods** and **algorithm adaptation methods**.
- Problem transformation methods transform the multi-label problem into a set of binary classification problems, which can

then be handled using single-class classifiers.

Whereas algorithm adaptation methods adapt the algorithms to directly perform multi-label classification. In other words, rather than trying to convert the problem to a simpler problem, they try to address the problem in its full form.

In an extensive comparison with other approaches, label-powerset method scores best, followed by the one-against-all method.

Both ML-KNN and label-powerset take considerable amount of time when run on this dataset, so experimentation was done on a random sample of the train data.

Further improvements:

The same problem can be solved using LSTMs in deep learning.

For more speed we could use decision trees and for a reasonable trade-off between speed and accuracy we could also opt for ensemble models.

Other frameworks such as MEKA can be used to deal with multi-label classification problems.

. . .

GitHub link of the project. LinkedIn profile.

. . .

Hope you enjoyed this tutorial. Thanks for reading..!