What is a multilabel problem?

A multilabel problem is a classificatoin problem in which the input can be mapped into different classes. This can be better considered with the movie genres. We would have seen many labels for the movies in theatres. A single movie will be classified as 'Romance", "Comedy" genres at the same time.

How to Solve the Problem? We can approach this in 3 ways:

#one v/s All

#one v/s one

#Error Correcting Output Codes

One v/s All

This strategy, also known as one-vs-all, is implemented in OneVsRestClassifier. The strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. Advantage of this approach is its interpretability. Since each class is represented by one and only one classifier, it is possible to gain knowledge about the class by inspecting its corresponding classifier. This is the most commonly used strategy and is a fair default choice. So we shall go by this method
As you can see throughout the whole experiment, I have used OneVsRest extensively. Let's talk about what is OneVsRest and why is it useful for multi-label classification tasks?

OVR as it's known is a very intuitive approach for solving multi-label classifications tasks in which the problem is decomposed into multiple binary classification problems, in which the labels should be mutually exclusive of each other. In OVR, we pick one class and train a binary classifier with the samples of the selected class on one side and all the other samples on the other side.

Thus, we end up with N classifiers for N labels. While testing, we simply classify the sample as belonging to the class with the maximum score among the N classifiers. Let's understand this with a simple example Let's consider the movie The Dark Knight and let us assume that it's associated with three tags — "adventure", "drama", "thriller". While training or testing the model, the OVR will work like this — is the tag "adventure" or not? Is the tag "drama" or not? Is the tag "thriller" or not? Here, we have broken down the multi-label problem into three binary classification problems.

At the end of the training phase, we will combine the outputs of these three binary classification problems into one single output. Thus, it's all about fitting one classifier per class. For each classifier, the class is fitted against all the other classes.

Embeddings

LIke any other machine learning algorithms these classifiers cannot work on the texts. so we need to find a way to convert thetexts into numerical equivalents We have two ways to do that :

TFIDF Word2Vec I think these two terminologies are much familar to every1. hence we can see a way to implement them

TF-IDF : Term Frequency and Inverse Document Frequency¶

In information retrieval, TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

TF stands for term frequency. Intuitively, TF intuitively means the probability of occurrence of a word in a sentence. It's a very useful technique to pick up the most frequently occurring words in a document corpus. However, this technique has it's own drawback. There are certain words which occurs extremely frequently than others. And their are certain words which occurs very rarely. TF values lies between 0 and 1.

For words which occurs very rarely, we can use the inverse document frequency (IDF). IDF is given by the formula log(N/n), where N is the total number of documents in our corpus, and n is the number of times each word appears across the entire corpus. Intuitively, IDF will be higher for words which occur rarely and will be less for words which occurs more frequently. So for each word in each review we will consider the product of (TF and IDF), and represent it as a d dimensional vector. TF-IDF basically doesn't consider the semantic meaning of words. But what is does is that it gives more importance to words which occurs less frequently in the whole data corpus and also gives importance to the most frequent words that occurs in each datapoint.

CountVectorizer

TfidfVectorizer() assigns a score while CountVectorizer() counts. TfidfVectorizer is used on sentences, while TfidfTransformer is used on an existing count matrix, such as one returned by CountVectorizer.

CountVectorizer implements both tokenization and count of occurrence.

In a corpus, several common words makes up lot of space which carry very little information about content of document. If we feed these counts directly to a classifier then those frequently occurring words will shadow the real interesting terms of the document. So we re-weight count feature vectors using tf-idf transform method and then feed the data into classifier for better classification.

TfidfVectorizer combines all options of CountVectorizer and TfidfTransformer in a single model.

Word counts with bag of words

Converts text into matrix where every row is an observation and every feature is a unique word. The value of each element in the matrix is either binary indicator marking the presence of that word or an integer of the number of times a word appears

Word2Vec

Word2Vec is a technique in which we convert a word to numerical vectors. In this case study, we will use a pre-trained W2V model by Google to convert words to it's corresponding vector form. For a given sentence, we will convert each of the word to vectors, sum them up and take their average value to represent an average word2vec word embedding for a given sentence.

Suppose we have N words in a sentence {w1,w2,w3,w4,w5,w6 … , wN}. We will convert each word to a vector, sum them up and divide by the total number of words (N) present in that particular sentence.

So our final vector will look like (1/N) * [word2vec(w1) + word2vec(w2) + word2vec(w3) …. + word2vec(wN)]

Word2Vec models are state of the art when it comes to preserving semantic relationships, relation between words and are very powerful for converting text to numerical vectors.
#Each word is represented as a vector of 32 more dimension instead of single number

#Semantic Information and relation between different words is also preserved

#Word2vec, like doc2vec, belongs to the text preprocessing phase.

#Specifically, to the part that transforms a text into a row of numbers.

#Word2vec is a type of mapping that allows words with similar meaning to have similar vector representation.

#The idea behind Word2vec is rather simple:

#we want to use the surrounding words to represent the target words with a Neural Network whose hidden layer encodes

#the word representation

OneVsRest multi-label strategy

The Multi-label algorithm accepts a binary mask over multiple labels. The result for each prediction will be an array of 0s and 1s marking which class labels apply to each row input sample.

Model Evaluation

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.
Our Key Performance Indicator for solving this problem will be Micro Averaged F1 score. In addition to Micro averaged F1 score we will also keep a track on Accuracy, Hamming Loss, Weighted Recall, Weighted Precision as our secondary KPIs. We will also keep a track on Macro Averaging performance metrics alongside the Micro Averaged measures.

Precision Score
Precision is basically the ratio of true positives (TP) to all the predicted positives (TP+FP). Precision, in a nutshell, tells us that out of all the points the model has predicted to be positive, how many of them are actually positive?

Recall Score
Recall is the ratio of the true positives (TP) to all the actual positives (TP+FN). Recall tells us that out of the total number of points that are actually positive, what fraction of it is declared by the model to be positive?

Micro averaged precision and recall
In case of micro averaging precision and recall, all the individual True Positives, True Negatives, False Positives and False Negatives for each classes are summed up and their average is taken. In the below formula, individual label is denoted by k:

Micro averaged F1 score

Micro averaged F1 score is simply the harmonic mean between the micro averaged recall and micro averaged precision values obtained using the above formulas.

In case of micro average F1 score, we are actually giving weightage based on how frequently a label occurs. Let's look at an example. Suppose, we have three tags(labels) t1, t2 and t3 spread across the entire dataset. Let's assume t1 occurs in almost 90% of the movies data, t2 occurs in 1% of the data and t3 occurs in 80% of the data. Now, if we consider the individual true positives for all these tags, the contribution of t2 will be less in both the numerator as well as the denominator in both formulas given in Fig 1. Hence, TP and FP for t2 will be small because the overall number itself is small. Subsequently, the contribution of t1 and t3 will be high, since they occur in much more number of data points across the entire dataset. In a nutshell, micro averaged f1 score is taking the label(tag) frequency into consideration. Intuitively speaking, even if the weighted recall and precision for tag t2 is very low, and we have a high precision and recall for tags t1 and t3, we will still get a pretty high micro averaged F1 score because of the higher contribution of tags t1 and t3. So we can easily conclude that micro averaged F1 score works extremely well when we have highly imbalanced distribution of tags.

Macro averaged F1 score
We have three tags t1, t2 and t3. In case of macro averaged F1 score, we will calculate the precision, recall and the subsequent F1 scores for each of these labels (or tags). Then, we will sum of all the individual F1 scores for all the tags and divide it by the total number of samples (in this case 3 samples). Macro averaged F1 score however doesn't work well when it comes to highly imbalanced distribution of tags. It also doesn't take the frequency of occurrence of tags into consideration. Hence, when it comes to multi-label classification and we have a highly imbalanced distribution of tags, micro averaged F1 score should be our primary metric of choice simply because it takes the frequency of occurrence of tags into consideration.

Hamming-Loss (Example based measure):

Example based metrics: include accuracy, hamming loss, etc.These are calculated for each example and then averaged across the test setIn 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.
The hamming loss (HL) is defined as the fraction of the wrong labels to the total number of labels.
For a multi-label problem, we need to decide a way to combine the predictions of the different labels to produce a single result. The method chosen in hamming loss is to give each label equal weight

LOG-LOSS :

Log Loss quantifies the accuracy of a classifier by penalizing false classifications. The exponentially decaying curve it possesses clearly indicates the same
It works only for those problems which have two or more labels. In order to calculate Log Loss the classifier must assign a probability to each class rather than simply yielding the most likely class.
Mathematically Log Loss is defined as:

Log loss formula where N is the number of samples or instances, M is the number of possible labels, y(i, j) is a binary indicator of whether or not label j is the correct classification for instance i, and p(i, j) is the model probability of assigning label j to instance i. After finalizing hamming-loss and log-loss as evaluation metrics, I was ready to study the different algorithms for multi-label classification -

Summary:
The model Decision Tree with TF-IDF vectorizer is performing better than other models after applying grid search and tuning the model to max_depth as 1, with Hamming loss of 3.45.