AUTOMATIC ASSIGNMENT OF DOMAIN TAGS TO EDUCATIONAL VIDEOS

Anirudh Ravi

Master of Technology Thesis

June 2018

International Institute of Information Technology, Bangalore

AUTOMATIC ASSIGNMENT OF DOMAIN TAGS TO EDUCATIONAL VIDEOS

Submitted to International Institute of Information Technology,
Bangalore
in Partial Fulfillment of
the Requirements for the Award of
Master of Technology

by

Anirudh Ravi
IMT2013005

International Institute of Information Technology, Bangalore
June 2018

Dedicated to my parents

Thesis Certificate

This is to certify that the thesis titled **Automatic assignment of domain tags to educational videos** submitted to the International Institute of Information Technology, Bangalore, for the award of the degree of **Master of Technology** is a bona fide record of the research work done by **Anirudh Ravi**, **IMT2013005**, under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

_____

Prof. Manish Gupta

Bengaluru,
The 1st of June, 2018.

AUTOMATIC ASSIGNMENT OF DOMAIN TAGS TO EDUCATIONAL VIDEOS

Abstract

Generally, Educational videos tend to have subtitles (mostly in English) associated with them to ease the process of understanding the concept covered in the video. Assigning domain tags to these videos can, therefore, be done based on these transcripts or subtitles. The problem of assigning domain tags automatically to educational videos can be reduced to and considered as a use case of Extreme Multi-Label Text Classification (XMTC). XMTC refers to the problem of assigning to each document its most relevant subset of class labels from an extremely large label collection, where the number of labels could reach hundreds of thousands or millions. In order to solve the problem of automatic video tagging, I am using a family of Convolutional Neural Network(CNN) models that are tailored for multi-label classification in particular as proposed in the paper *Deep Learning for Extreme Multi-label Text Classification* [1]. My objective is to come up with a model of this family with the right parameters to solve the problem of automatic video tagging.

Acknowledgements

I would like to thank Prof.Manish Gupta for guiding me throughout my thesis. I would also like to thank Mr.Kuldeep and Mr.Dalon from VideoKen for being available whenever I needed them to clear my doubts.

Contents

List of Figures

List of Tables

List of Abbreviations

CNN . . . . . Convolutional Neural Network

GloVe . . . . . Global Vectors for Word Representation

IIITB . . . . . International Institute of Information Technology Bangalore

RNN . . . . . Recurrent Neural Network

TF . . . . . Term Frequency

XMTC . . . . . Extreme Multi-Label Text Classification

CHAPTER 1

INTRODUCTION

Assigning educational videos top k tags based on the content is normally done manually. I would like to ease this process by automatically assigning based on the transcript (subtitles) of the video. There is an implicit assumption when working with transcripts, which is, the speaker in the video should speak fluently so that the transcript does not have unnecessary words or ”noise” such as ”hmmm” or something similar. Although, this seems very idealistic coming up with such transcripts is all together another research problem.

Automatic video tagging can be thought of as an use-case of XMTC problem. The reason for that is as follows: in XMTC tasks, a document (which is video transcript in our scenario) is to be assigned its most relevant subset of labels from an extremely large space of categories, which is exactly what we would like to do with videos. Multi-label classification is fundamentally different from the traditional binary or multi-class classification. Binary classifiers treat class labels as independent target variables, which is clearly sub-optimal for multi-label classification as the dependencies among class labels cannot be leveraged.

Solving XMTC problems is difficult because of three main issues:- huge label space, severe data sparsity issue and scalability. There have been several approaches to deal with the above issues. Most of these fall into two categories: target-embedding and tree-based ensemble methods. While the former addresses the data sparsity issue, the latter addresses the scalability issue.

CHAPTER 2

LITERATURE REVIEW

This chapter covers a brief overview of the model used and word representations

2.1 Word Representations

2.1.1 Global Vectors for Word Representation(GloVe)

GloVe [2] is an unsupervised learning algorithm for obtaining vector representations for words. The resulting representations showcase interesting linear substructures of the word vector space. The model is essentially a log-bilinear model with a weighted least-squares objective. The main intuition behind the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning. The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. Using the fact that the logarithm of a ratio equals the difference of logarithms, this objective associates (the logarithm of) ratios of co-occurrence probabilities with vector differences in the word vector space. Because these ratios can encode some form of meaning, this information gets encoded as vector differences as well. For this reason, the resulting word vectors perform very well on word analogy tasks.

2.2 Target-Embedding and Tree-based Ensemble Methods

Target-Embedding methods address the data sparsity issue in training XMTC classifiers by finding a set of low-dimensional embeddings of the label vectors for training instances in the target space. Similar to classical decision-tree learning, Tree-based methods induce a tree structure which recursively partitions the instance space or subspaces at each non-leaf node, and has a base classifier at each leaf node which only focuses on a few active labels in that node. The methods are different from classical decision trees in a sense that the new methods learn a hyperplane (equivalent to weighed combination of all features) to split the current instance space at each node, instead of selecting a single feature based on information gain for splitting. SLEEC [3] is considered as representative of the former category and FastXML [4] is the top performing method in the latter category.

A fundamental limitation of the aforementioned methods is that the methods are based on bag-of-word representations of documents i.e. words are treated as independent features out of context.

2.3 Deep Learning models

Deep learning approaches have performed better than linear predictors (e.g., linear SVM) with a bag-of-word based features as input in multi-class text classification, which is a special case of multi-label classification where each document is restricted to have only one label. These approaches are designed for multi-class settings, not taking multi-label settings into account in model optimization.

Sections 2.3.1 and 2.3.2 cover techniques used for multi-class text classification tasks and section 2.3.3 extends over the architecture of CNN-Kim [5] as discussed in 2.3.2 which works well for multi-label text classification tasks.

2.3.1 CNN and RNN

Although standard deep learning approaches like CNN and RNN work well for multi-class text classification tasks, they do not work well in the multi-label text classification settings as these models do not take label co-occurrence patterns into consideration.

2.3.2 CNN-Kim

CNN-Kim is basically a simple CNN with one layer of convolution on top of word vectors obtained from an unsupervised neural language model for sentence-level classification tasks.

2.3.2.1 CNN-Kim Model

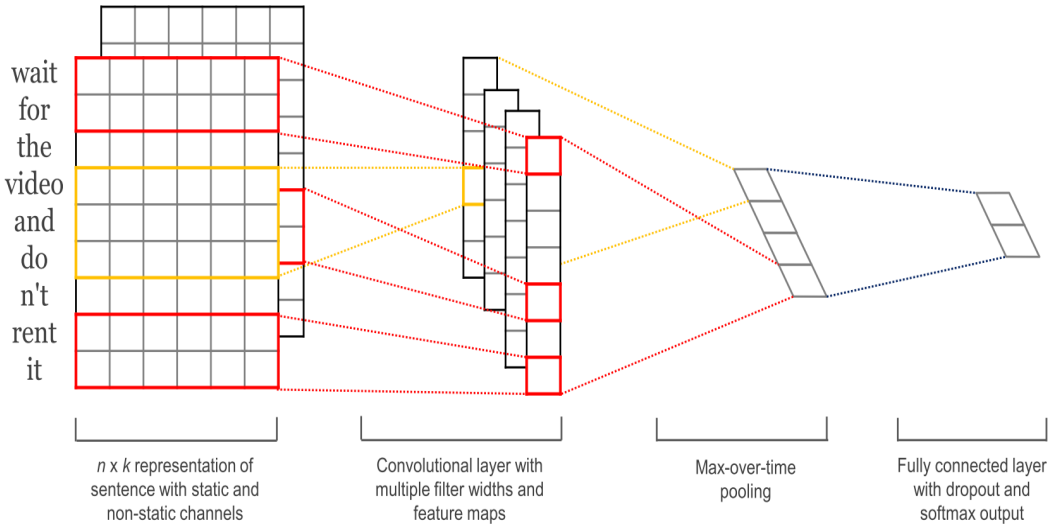


Figure FC2.1: Model architecture with two channels for an example sentence

Let $x_i \in \Re^k$ be the k-dimensional word vector corresponding to the i^{th} word in the

sentence. In general, a sentence of n words (padded where necessary) is represented as

$$x_{i:i+n-1} = x_i \oplus x_{i+1} \oplus \cdots \oplus x_{i+n-1} \quad \text{(Eqn 2.1)}$$

where $\oplus$ is the concatenation operator. A new feature is obtained through a convolution operation i.e. by applying a filter $w \in \Re^{hk}$ on a window of h words. For example, a feature c_i is generated from a window of words $x_{i:i+h-1}$ by

$$c_i = f(w \cdot x_{i:i+h-1} + b) \quad \text{(Eqn 2.2)}$$

Here $b \in \Re$ is a bias term and f is a non-linear function such as hyperbolic tangent.

This filter is applied to each possible window of words in a sentence. A sentence of n words is represented as $x_{1:n}$. Window of words for the above sentence with window size h is given as $\{x_{1:h}, x_{2:h+1}, ..., x_{n-h+1,n}\}$. A feature map is obtained by applying the filter on the above window of words which can be represented as

$$c = [c_1, c_2, ..., c_{n-h+1}] \quad \text{(Eqn 2.3)}$$

with $c \in \Re^{n-h+1}$. A max-over-time pooling operation [5] is applied over the feature map and the maximum value $\hat{c} = max\{c\}$is taken as the representative feature corresponding to this filter. By doing this, most important feature having the highest value is captured. There can be multiple filters. The same process is applied for every feature maps each corresponding to a filter and finally most important features from all filters are obtained. These features form the penultimate layer and are passed to a fully connected softmax layer whose output is the probability distribution over labels.

The architecture of CNN-Kim allows multiple channels of word vectors. These channels can simply be different ways of word vector representations. For example, there can be two channels of word vectors with one kept static throughout training and the other is fined-tuned via backpropagation. In multichannel architecture, illustrated

in FC2.1, each filter is applied to both channels and the results are added to calculate c_i in Eqn 2.2.

2.3.3 XML-CNN

XML-CNN [1] is a CNN model for multi-label text classification tasks.

2.3.3.1 XML-CNN Model

The model is based on CNN-Kim, the CNN model for multi-class text classification as described in 2.3.2.

Similar to CNN-Kim, XML-CNN, as illustrated in FC2.2, also passes a document through various convolutional filters and learns a rich number of feature representations. The key characteristics of this model lie in the following connected layers. This model captures more fine-grained features from different regions of the document by adopting a dynamic max pooling scheme. A binary cross-entropy loss over sigmoid output is used as it is more tailored for XMTC. To learn compact document representations, an additional hidden bottleneck layer is inserted between pooling and output layer, which reduces model size and boosts model performance.

Let $e_i \in \Re^k$ be the k-dimentional word embedding corresponding to the i^{th} word in a document, $i = 1,2,...,m$. The whole document is represented by the concatenation of its word embeddings $e_{i:m} = [e_i,...,e_m] \in \Re^{km}$. In general, the text region from i^{th} word to the j^{th} word is represented by $e_{i:j} = [e_i,...,e_j] \in \Re^{k(j-i+1)}$. A convolution filter $\upsilon \in \Re^{kh}$ is applied to a text region of h words $e_{i:i+h-1}$ to produce a new feature

$$c_i = g_c(\upsilon^T e_{i:j+h-1}) \qquad \text{(Eqn 2.4)}$$

where g_c is the nonlinear activation function for this convolution layer, such as sigmoid or ReLU. A feature map c associated with the applied filter υ is basically all $c_i's$ put together, $c = [c_1,...,c_m] \in \Re^m$. Padding at the end of the document is done to obtain m

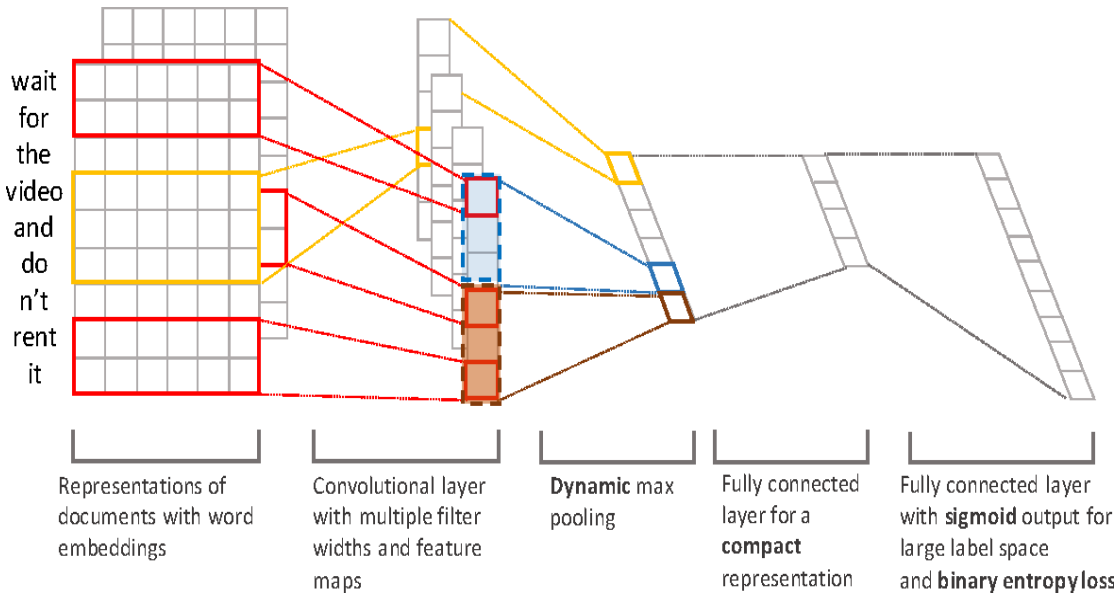


Figure FC2.2: Model architecture of XML-CNN with an example sentence

features if the length of the document is short. Rich semantic information is captured by using multiple filters with different window sizes in a convolution layer. If t filters are used then there will be t feature maps as a result, $c^{(1)},...,c^{(t)}$, each corresponding to a filter. A pooling operation $P(.)$ is then applied to each of these feature maps to produce tp-dimentional vectors $P(c^{(i)}) \in \Re^p$. The choice of the pooling function is discussed in Section 2.3.3.2. The output of the pooling layer is followed by a fully connected bottleneck layer with h hidden units and then an output layer with L units corresponding to the score of each label, denoted by $f \in \Re^L$:

$$f = W_o g_h(W_h[P(c^{(1)}),...,P(c^{(t)})]) \qquad \text{(Eqn 2.5)}$$

Here $W_h \in \Re^{h \times tp}$ and $W_o \in \Re^{L \times h}$ are the weight matrices associated with the bottleneck layer and the output layer; g_h is the element-wise activation function applied to the bottleneck layer. The key attributes that make XML-CNN [1] especially suited for XMTC are the pooling operation, loss function and the hidden bottleneck layer between pooling layer and output layer. These key attributes are discussed in below sections.

2.3.3.2 Dynamic Max Pooling

A max-over-time pooling scheme is the one that is usually adopted. This means that taking the maximum element of a feature map: $P(c) = \hat{c} = max\{c\}$. The idea with max-over-time scheme is to capture the most important feature i.e. the entry with the highest value, in each feature map. Using this scheme each filter generates a single feature, so the output of the pooling layer is $[P(c^{(1)}), ..., P(c^{(t)})] = [\hat{c}^{(1)}, ..., \hat{c}^{(t)}] \in \Re^t$. However, There is one drawback of max-over-time pooling, which is, for each filter only the largest value is chosen to carry information to subsequent layers, which is problematic when the document is long. This pooling scheme does not capture any information about the position of the largest value.

XML-CNN however adopts a dynamic max pooling scheme. Instead of choosing only one feature per document, p features are chosen to capture richer information. For a document containing m words, its m-dimensional feature map is divided into p chunks, each chunk is pooled to a single feature by taking the largest value within that chunk. By doing this, the information about different parts of the document can then be forwarded to the following layers. Under this scheme, each filter produces a p-dimensional feature (assuming m is divisible by p):

$$P(c) = [max\{c_{1:\frac{m}{p}}\}, ..., max\{c_{m-\frac{m}{p}+1:m}\}] \in \Re^p \qquad \text{(Eqn 2.6)}$$

which captures both important features and position information about these important features.

2.3.3.3 Loss Function

A binary cross-entropy loss (BCE) over sigmoid function activation works better than other loss methods like rank loss [6] when applied to multi-label classification datasets in a simple feed-forward network [7]. The binary cross-entropy objective is as

follows:

$$\min_{\Theta} \quad -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{L}[y_{ij}\log(\sigma(f_{ij}))+(1-y_{ij})\log(1-\sigma(f_{ij}))] \qquad \text{(Eqn 2.7)}$$

where σ is sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$.

2.3.3.4 Hidden Bottleneck Layer

Unlike CNN-Kim where pooling layer and output layer are directly connected, in XML-CNN model, a fully connected hidden layer with h units is added between pooling layer and output layer, referred to as the hidden bottleneck layer, as the number of its hidden units is far less than the pooling and output layer. There are two main reasons behind this. Firstly, if the pooling layer is directly connected to the output layer, the number of parameters between these two layers would be $O(pt \times L)$. When the document is long, and the number of labels is large, more filters and more chunks are needed to obtain good document representation, and in XMTC setting, L can go up to millions, so model size of $O(pt \times L)$ might not fit into a common GPU memory. With an additional hidden layer inserted between pooling and output layer, the number of parameters reduces to $O(h \times (pt + L))$ which is much less than $O(pt \times L)$. Secondly, without this hidden bottleneck layer, the model only has one hidden layer of non-linearity, which is not expressive enough to learn good document representation and classifiers.

XML-CNN adds on to the strengths of existing CNN models by taking into account multi-label co-occurrence patterns in both the design of the neural network architecture and the optimization objective. This deep learning method also scales successfully to large XMTC datasets. The method gave best or second best results among all competing methods like SLEEC and FastXML on all benchmark datasets as shown via experiments in [1]. Hence, to solve my problem of video tagging, I decided to use XML-CNN model and find the hyper-parameters of the model that can give good results on my dataset. The details about the dataset are discussed in a later section.

CHAPTER 3

DATA SET PREPARATION AND EVALUATION METRICS

This chapter covers the details regarding the evaluation metrics used and how the data set is prepared for training and testing purposes.

3.1 Data Set Preparation

The videos that I am using as an input are taken from youtube. All these videos have subtitles' information (in English) associated with them. The subtitles' information is extracted from all these videos in the form of SRT files. An example raw SRT file (transcript) is shown in FC3.1.

```
1
00:00:03,710 --> 00:00:09,510
when I first got rolling it was kind of

2
00:00:07,170 --> 00:00:12,030
easier back then in some way is it

3
00:00:09,510 --> 00:00:13,830
harder in other ways the actual code was

4
00:00:12,030 --> 00:00:16,140
easier back then at the same time today

5
00:00:13,830 --> 00:00:18,930
you much better resources you have

6
00:00:16,139 --> 00:00:20,339
code.org you have a search results for

7
00:00:18,930 --> 00:00:23,280
pretty much any tech question you would

8
00:00:20,340 --> 00:00:26,280
ever want to ask you have school courses
```

Figure FC3.1: Raw transcript (SRT) file

Basically, a transcript represents a video. My goal is to assign a video a subset of tags/labels from a known target list of tags based on this transcript information.

The text i.e. the subtitle information is extracted from each of these transcripts. An example extracted SRT file (transcript) is shown in FC3.2.

In this video I'll talk about various aspects of the course, the topics that we'll cover, the kinds of skills you can expect to acquire, the kind of background that I expect, the supporting materials and the available tools for self assessment. Let's start with the specific topics that this course is going to cover.
The course material corresponds to the first half of the ten week Stanford course. It's taken by all computer science undergraduates, as well as many of our graduate students. There will be five high level topics, and at times these will overlap. The five topics are first of all, the vocabulary for reasoning about algorithm performance, the design and conquer algorithm design paradigm, randomization and algorithm design, primitives for reasoning about graphs, and the use and implementation of basic data structures. The goal is to provide an introduction to and basic literacy in each of these topics. Much, much more could be said about each of them, than we'll have time for here. The first topic is the shortest, and probably also the driest. But it's a prerequisite for thinking seriously about the design and analysis of algorithms. The key concept here is big-O notation, which, conceptually, is a modeling choice about the granularity with which we measure a performance metric like the running time of an algorithm. It turns out that the sweet spot for clear high level thinking about algorithm design, is to ignore constant factors and lower-order terms. And to concentrate on how well algorithm performance scales with large input sizes. Big O notation is the way to mathematize this sweet spot. Now, there's no one silver bullet in algorithm design. No single problem solving method that's guaranteed to unlock all of the computational problems that you're likely to face. That said, there are a few general algorithm design techniques. High

Figure FC3.2: Extracted transcript (SRT) file

The extracted transcript files are then processed to remove special characters like commas, question marks, full-stops etc. An example file is shown in FC3.3.

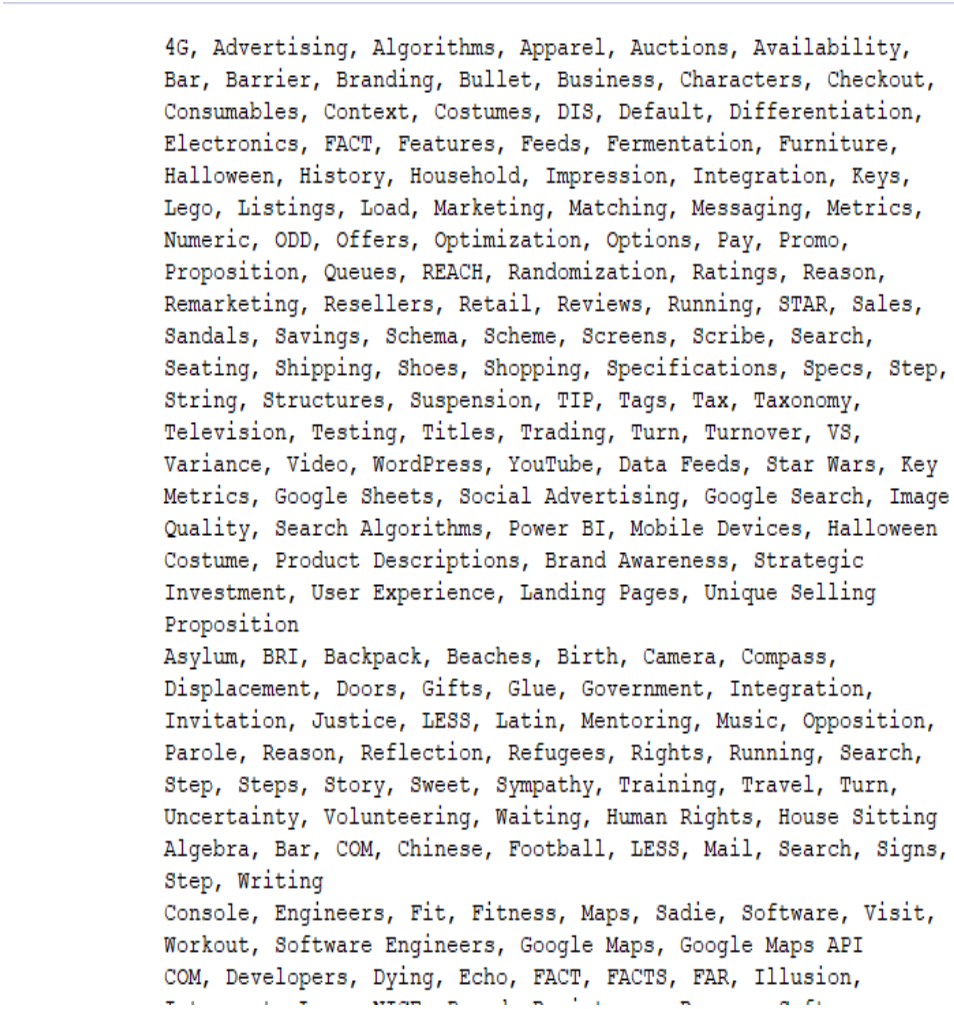
welcome so what i want to do is i want to go back again with the
compound interest which we should know is a equals p times 1
plus r over n raised to the m times t and kind of talk a little
bit more about the compounded because remember a represents our
final value p represents our initial investment our represents
our interest rate our annual interest rate in decimal form t
represents the number of years and our represent and represents
how many times are going to compound something per year right we
re dealing with an annual interest rate and how many times are
going to compound per year so there s a couple couple terms that
you re going to read throughout your problems you might you know
question like well what exactly they asking and they got leap
year here you know what exactly is this so i want to go through
each one of those terms so you don t understand at least what n
is going to represent so if i say you know you put some money
into a interest bearing account and it s going to be yearly well
therefore n is going to equal one all right then sometimes they
call it semi annually right annually they say hey it compounds
annually that s the same thing as yearly nearly or annually all
right usually a lot of times you see early but sometimes your
semi annually well that means you re going to compound it twice
in a year all right so therefore you go with semi annually and a
lot of times we have quarterly so if i say hey you know we re
getting some interest bearing it s going to happen every quarter
well that quarter think about a quarter four quarters and a
dollar so therefore quarterly would be n equals for another one
very very common that we local into would be monthly now you
just think about well if you re compounding something monthly
per year how many times is that per year well since there s 12
months in the year we could say n equals 12 sometimes they do
because you know a lot of businesses think about businesses a

Figure FC3.3: Processed transcript (SRT) file

3.1.1 Proposed two-step process for data set preparation

Initially, I only had these transcripts, each representing a video, to work with. I did not have the initial transcript-tags mapping information. Ideally, a video-tags mapping should be done manually but when working with huge data sets that is not possible. So, in order to build the ground truth tags corresponding to each video, a two step process is used. The first step involves taking an intersection of the target label list (space) and the transcript corresponding to each and every video. During this step, Unigrams, Bigrams and Trigrams are generated corresponding to each transcript and three inter-

sections: Unigrams & label list, Bigrams & label list and Trigrams & label list are done to generate initial labels corresponding to a transcript. A file is then prepared combining all the three files obtained after the intersection process. Each line in the consolidated file contains all the initial tags: Unigram, Bigram and Trigram tags corresponding to a unique video. The target label list is updated by removing the labels that do not appear at all in any of the documents. A portion of the file is shown in FC3.4.



```
4G, Advertising, Algorithms, Apparel, Auctions, Availability,
Bar, Barrier, Branding, Bullet, Business, Characters, Checkout,
Consumables, Context, Costumes, DIS, Default, Differentiation,
Electronics, FACT, Features, Feeds, Fermentation, Furniture,
Halloween, History, Household, Impression, Integration, Keys,
Lego, Listings, Load, Marketing, Matching, Messaging, Metrics,
Numeric, ODD, Offers, Optimization, Options, Pay, Promo,
Proposition, Queues, REACH, Randomization, Ratings, Reason,
Remarketing, Resellers, Retail, Reviews, Running, STAR, Sales,
Sandals, Savings, Schema, Scheme, Screens, Scribe, Search,
Seating, Shipping, Shoes, Shopping, Specifications, Specs, Step,
String, Structures, Suspension, TIP, Tags, Tax, Taxonomy,
Television, Testing, Titles, Trading, Turn, Turnover, VS,
Variance, Video, WordPress, YouTube, Data Feeds, Star Wars, Key
Metrics, Google Sheets, Social Advertising, Google Search, Image
Quality, Search Algorithms, Power BI, Mobile Devices, Halloween
Costume, Product Descriptions, Brand Awareness, Strategic
Investment, User Experience, Landing Pages, Unique Selling
Proposition
Asylum, BRI, Backpack, Beaches, Birth, Camera, Compass,
Displacement, Doors, Gifts, Glue, Government, Integration,
Invitation, Justice, LESS, Latin, Mentoring, Music, Opposition,
Parole, Reason, Reflection, Refugees, Rights, Running, Search,
Step, Steps, Story, Sweet, Sympathy, Training, Travel, Turn,
Uncertainty, Volunteering, Waiting, Human Rights, House Sitting
Algebra, Bar, COM, Chinese, Football, LESS, Mail, Search, Signs,
Step, Writing
Console, Engineers, Fit, Fitness, Maps, Sadie, Software, Visit,
Workout, Software Engineers, Google Maps, Google Maps API
COM, Developers, Dying, Echo, FACT, FACTS, FAR, Illusion,
```

Figure FC3.4: Consolidated labels' file

The second step is to map the list of labels obtained via intersection to small number of top tags corresponding to every transcript. This is done by calculating the TF (Term Frequency) values of every label corresponding to each transcript and taking the labels

that have top TF values. At the end of this two-step process, we have a ground truth i.e. subset of top tags associated with every transcript. Depending on the number of top tags, *k*, each line in the resultant file corresponding to a video transcript can have atmost *k* tags. For my experiments, I chose *k* values to be 1,3 and 5 i.e. finding the top one, top three and top five labels of a particular video transcript respectively.

After the second step, three files are created, one for each value of *k* as shown in the figures FC3.5, FC3.6, FC3.7.

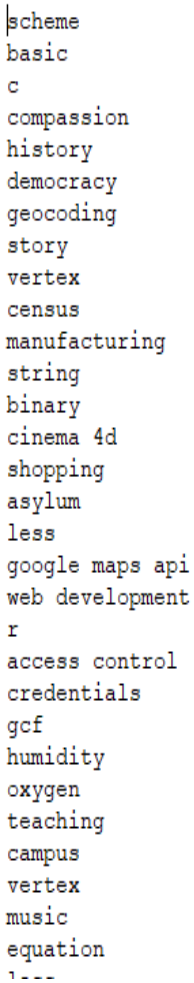


Figure FC3.5: label mappings' file created using TF for k=1

```
scheme, reggae, recall
basic, objects, fit
c, string, characters
compassion, pay, swimming
history, facebook, painting
democracy, government, rights
geocoding, string, json
story, scorecard, piano
vertex, equation, reason
census, birth, registration
manufacturing, additive manufacturing, science
string, teaching, story
binary, f, step
cinema 4d, cinema, 3d
shopping, titles, optimization
asylum, story, camera
less, step, signs
google maps api, google maps, software engineers
web development, software, echo
r, proof, derivatives
access control, trusted computing, mac
credentials, weblogic, enterprise manager
gcf, video, string
humidity, spacecraft, japanese
oxygen, dna, mitochondria
teaching, education, voting
campus, science, engineering
vertex, axis, plot
music, birds, sound
equation, vector, curl
```

Figure FC3.6: label mappings' file created using TF for k=3

```
scheme, reggae, recall, protocol, peace
basic, objects, fit, c, strategy
c, string, characters, print, blueprint
compassion, pay, swimming, turn, story
history, facebook, painting, education, art
democracy, government, rights, presidency, politics
geocoding, string, json, google maps, ios
story, scorecard, piano, offers, gerd
vertex, equation, reason
census, birth, registration, maps, fertility
manufacturing, additive manufacturing, science, orbit, surface
string, teaching, story, print, pop
binary, f, step, music, writing
cinema 4d, cinema, 3d, modeling, max
shopping, titles, optimization, schema, availability
asylum, story, camera, waiting, refugees
less, step, signs, search, mail
google maps api, google maps, software engineers, workout, visit
web development, software, echo, software development, soft
skills
r, proof, derivatives, notation, f
access control, trusted computing, mac, intelligence, defense
credentials, weblogic, enterprise manager, default, ports
gcf, video, string, paint, multiples
humidity, spacecraft, japanese, iss, developing countries
oxygen, dna, mitochondria, membrane, iron
teaching, education, voting, themes, ownership
campus, science, engineering, story, learning environment
vertex, axis, plot
music, birds, sound, cage, sonic
```

Figure FC3.7: label mappings' file created using TF for k=5

We now have 50,000 video transcripts (tagged) obtained through the above two-step process. Out of these 50,000 transcripts, 70% i.e 35,000 are used as training data and the rest 30% i.e. 15,000 form the 1^{st} type of test data.

I have manually tagged 108 videos taken from Machine Learning online course on Coursera. Each video transcript is given a maximum of 5 tags. For example, a video on autonomous driving having content as shown in FC3.8 is assigned 4 tags: ”Autonomous Driving, Neural Networks, Deep Learning and Machine Learning”.

In this video, I'd like to show you a fun and historically important example of neural networks learning of using a neural network for autonomous driving.
That is getting a car to learn to drive itself.
The video that I'll showed a minute was something that I'd gotten from
Dean Pomerleau, who was a colleague who works out in Carnegie Mellon University out on the east coast of the United States.
And in part of the video you see visualizations like this.
And I want to tell what a visualization looks like before starting the video.
Down here on the lower left is the view seen by the car of what's in front of it.
And so here you kinda see a road that's maybe going a bit to the left, and then going a little bit to the right.
And up here on top, this first horizontal bar shows the direction selected by the human driver.
And this location of this bright white band that shows the steering direction selected by the human driver where you know here far to the left corresponds to steering hard left, here corresponds to steering hard to the right.
And so this location which is a little bit to the left, a little bit left of center means that the human driver at this point was steering slightly to the left.
And this second bot here corresponds to the steering direction selected by the learning algorithm and again the location of this sort of white band means that the neural network was here selecting a steering direction that's slightly to the left.
And in fact before the neural network starts leaning initially, you see that the network outputs a grey band, like a grey, like

Figure FC3.8: Autonomous driving transcript

These 108 manually tagged video transcripts serve as 2^{nd} type of test data. For this test data, the previous 50,000 transcripts act as training data.

3.2 Evaluation Metric

In XMTC datasets, the label spaces are very large but each instance has very few relevant labels. So, it is important to present each test instance a short ranked list of potentially relevant labels and to evaluate the quality of such lists with an emphasis on the relevance of the top portion of each list. Rank-based evaluation metrics are better suited for comparing XMTC methods for these reasons. Precision at top k ($P@k$) belongs to the rank-based evaluation metrics which will be used to compare results. Let $y \in \{0,1\}^L$ denote the vector of true labels of a document and $\hat{y} \in \Re^L$ denote the system-predicted score vector for the same document. The metrics are defined as:

$$P@k = \frac{1}{k} \sum_{l \in r_k(\hat{y})} y_l \tag{Eqn 3.1}$$

$$DCG@k = \sum_{l \in r_k(\hat{y})} \frac{y_l}{\log_2(l+1)} \tag{Eqn 3.2}$$

$$nDCG@k = \frac{DCG@k}{\sum_{l=1}^{min(k,||y||_0)} \frac{1}{\log_2(l+1)}} \tag{Eqn 3.3}$$

where L is the number of labels, $r_k(\hat{y})$ is the set of rank indices of the truly relevant labels among the top-k portion of the system-predicted ranked list for a document and $||y||_0$ is the number of relevant labels in the ground truth label vector y. $P@k$ and $nDCG@k$ are calculated for each test document and then averaged over all the documents.

CHAPTER 4

EXPERIMENTS AND RESULTS

This chapter gives the details about the experiments conducted on the data set and draw conclusion from the results obtained.

4.1 Experiments

There are various number of hyper-parameters that should be taken into account while conducting experiments using the XML-CNN [1] on the dataset. A few major ones are: max sequence length (i.e. number of words) of a document, dimension of word embedding representation, number of filter sizes (could be a list of integer), number of filters (i.e. kernels) in CNN model, number of pooling units in 1D pooling layer and number of hidden units. For the below experiments the values for above hyper-parameters are 500, 300, $\{2,4,8\}$, 32, 32 and 512 respectively. I am conducting experiments on two datasets: one with 35,000 training instances & 15,000 test instances and the other with 50,000 training instances & 108 test instances. Also, I am running the models for 50 epochs.

Initial total number of labels in the given target label list (L) = 33514.

Let k be the number of top tags, N be the number of training instances, M be the number of test instances, L be the updated total number of class labels, $\hat{L}$ be the average number of label per document, $\tilde{L}$ be the average number of documents per label, $\hat{W}$ be the average number of words per document in the training set, $\tilde{W}$ be the average number of words per document in the test set. Basic data statistics about the experiments are

shown in TC4.1 and TC4.2.

4.1.1 Experiment 1

Table TC4.1: Experiment-1 Data Statistics

k	N	M	L	$\hat{W}$	$\tilde{W}$	$\hat{L}$	$\tilde{L}$
1						1	2.8
3	35,000	15,000	17,812	2053.44	2057.35	2.82	7.93
5						4.49	12.6

4.1.2 Experiment 2

Table TC4.2: Experiment-2 Data Statistics

k	N	M	L	$\hat{W}$	$\tilde{W}$	$\hat{L}$	$\tilde{L}$
3	50,000	108	17,812	2054.62	1806.92	2.82	7.95
5						4.48	12.62

4.2 Results

Table TC4.3: Experiment-1 results: $P@k$ represents Precision@k and $G@k$ represents nDCG@k

k	$P@k(in\%)$	$G@k(in\%)$
1	21.53	21.53
3	21.99	24.78
5	21.27	25.77

Table TC4.4: Experiment-2 results: $P@k$ represents Precision@k and $G@k$ represents nDCG@k

k	$P@k(in\%)$	$G@k(in\%)$
3	16.67	17.69
5	14.4	22.94

The Results shown in TC4.3 and TC4.4 can be understood as the following: Let's take k=5 case. What $G@k$ value of 25.77% essentially says is that when we are predicting 5 labels for a document the system is predicting on an average 25.77 % (of 5) labels correctly.

I think these results are because of conducting experiments on comparatively less training instances. As seen in the data statistics of the data set, the average number of documents per label is very less. So, for the model to learn better it is important that the training data be increased to the order of lakhs.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

One possible method is suggested to solve the problem of automatic video tagging by utilizing the video transcript information. The major struggle was to prepare the data set itself. The proposed two-step process is basic and should be used as a starting point. Future work can be done to improve the mapping information by just using the video transcripts and not relying on the target label list. More experiments need to be done after increasing the size of the data set.

Bibliography

[1] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. Deep learning for extreme multi-label text classification. In *Proceedings of the* 40^{th} *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124, August 2017.

[2] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[3] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems*, pages 730–738, 2015.

[4] Yashoteja Prabhu and Manik Varma. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 263–272, 2014.

[5] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research 12*, pages 2493—-2537, 2011.

[6] Min-Ling Zhang and Zhi-Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE transactions on Knowledge and Data Engineering*, 18, 10:1338—-1351, 2006.

[7] Jinseok Nam, Jungi Kim, Iryna Gurevych Eneldo Loza Mencía, and Johannes Fürnkranz. Large-scale multi-label text classification - revisiting neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer*, pages 437—-452, 2014.