Lazada Product Title Quality Challenge

Project Report

Data Analytics Squad
Akshay Sharma
Ashish Shah
Ayush Dattagupta
Nikhil Thakur
Rahul Dhavalikar

Introduction

Having a good quality product title is extremely important in e-commerce systems as product titles are the first details read by users or customers and make the first impression of the product in their minds. But over the years, it was observed that if the titles of products were confusing, lengthy, or poorly formatted, it had a direct and severe impact on the sales of the product. Hence, Lazada's internal quality control team manually reviews the titles judging them based on clarity and conciseness. But this is a time consuming process. It is desirable to build a product title quality machine learning model that can automatically grade the clarity and the conciseness of a product title.

Problem

On Lazada, there are millions of products across thousands of categories. To stand out from the crowd, sellers employ creative, sometimes disruptive efforts to improve their search relevancy or attract the attention of customers. In this problem, we are provided with a set of product titles, description, and attributes like price, country, together with the associated title quality scores - clarity and conciseness. Our task is to build a product title quality model that can automatically grade the **clarity** and the **conciseness** of a product title.

Clarity - The product title is clear if within five seconds of reading it, one can understand the title, what the product is, and quickly figure out the key attributes such as color, size, model etc.

Conciseness - The product title is concise if it is short enough to contain all the necessary and relevant information. Otherwise the title maybe too long with many unnecessary and irrelevant words or it is too short not giving any specific details about the product.

A few examples highlighting the problem statement are below:

Example 1: ***hot red clutch rug sack travel backpack unisex cheap with free gift***
This is not a concise title as it contains many irrelevant words included purely to push the product ranking up in the search engine results.
Hence the title can be something as follows: ***Red unisex travel rucksack backpack***

Example 2: ***1 Pair of Unisex Touch Screen Sensitive Gloves Knitted Winter Warm Christmas Glove Red***
Similarly, the above title is not clear or concise as it is talking about things which would confuse the customer such as "Christmas, Winter, Unisex". This is a clear and concise representation of the title: ***Touchscreen sensitive knitted gloves red***

Example 3: ***Hot Tom Clovers Womens Mens Classy Look Cool Simple Style Casual Canvas Crossbody Messenger Bag Handbag Fashion Bag Tote Handbag Gray***

The above title is a perfect example of lack of clarity. It is talking about multiple items and leads to confusion among the customer. It takes much longer than the ideal 5 seconds time-limit to understand the true details of the product.

It can be rephrased to something like this: ***Tom Clovers canvas crossbody messenger bag gray***

Given Data & its Format

Before preprocessing, first let's take a look at the given data and its format.

The data contains product titles that are stratified sampled based on sales figures across multiple product categories in Singapore, Malaysia, Philippines. Each line in the file refers to a specific product. For example:

my, NO037FAAA8CLZ2ANMY, RUDY Dress, Fashion, Women, Clothing, "<ul> <li>Short Sleeve</li> <li>3 Colours 8 Sizes</li> <li>Dress</li> </ul> ", 33.0, local

The comma-separated values in each line correspond to the following features:

country : The country where the product is marketed, with three possible values: my for Malaysia, ph for Philippines, sg for Singapore

sku_id : Unique product id, e.g., "NO037FAAA8CLZ2ANMY"

title : Product title, e.g., "RUDY Dress"

category_lvl_1 : General category that the product belongs to, e.g., "Fashion"

category_lvl_2 : Intermediate category that the product belongs to, e.g., "Women"

category_lvl_3 : Specific category that the product belongs to, e.g., "Clothing"

short_description : Short description of the product, which may contain html formatting, e.g., "<ul> <li>Short Sleeve</li> <li>3 Colours 8 Sizes</li> <li>Dress</li> </ul> "

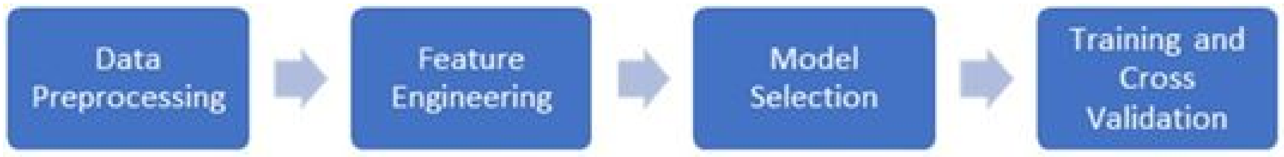
price : Price in the local currency, e.g., "33.0". When country is my, the price is in Malaysian Ringgit. When country is sg, the price is in Singapore Dollar. When country is ph, the price is in Philippine Peso.

product_type : It could have three possible values: local means the product is delivered locally, international means the product is delivered from abroad, NA means not applicable.

The dataset for the competition is available here:
https://drive.google.com/drive/folders/0B-rwT7IHM52ockZabnpvYVE3Z00

Overview

At a high level, the project was divided into 4 parts as follows.



1. Data Preprocessing: This involves the cleaning, processing and conversion of the dataset into a uniform format. It is done because a lot of the datasets have incomplete, noisy or inconsistent data.

2. Feature Engineering: This involves building of features used to train the model. Quite simply, this is the process of manually constructing new attributes from raw data. It involves intelligently (using domain knowledge) combining or splitting existing raw attributes into new one which have a higher predictive power.

3. Model Selection: Multiple approaches were implemented and executed as a part of this project. A comparative analysis among all approaches was carried out to judge the benefits and drawbacks of each approach. As expected, we observed certain approaches performing better than others in certain tasks. A detailed analysis of the advantages and drawbacks of each approach have been well documented in the report.

4. Training and Cross Validation: This involved the training of models using each of the above discussed approaches and cross validated to compare the error among them.

Below are the details of each part performed as a part of this project.

1. Data Preprocessing

Data preprocessing are techniques that involve transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing prepares raw data for further processing.

The Data Preprocessing task in our project had the following steps:

- Data Cleaning is the process of detecting and removing corrupt or inaccurate records from a record set or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data. Removal of undesirable elements in the dataset such as special characters including punctuation marks was done using data cleaning techniques.
  For eg: The dataset contains titles such as: ***ANMYNA Complaint Silky Set æŸ”é¡ºæ´—å‘é…å¥— (Shampoo 520ml + Conditioner 250ml)*** which has undesirable characters that provide no useful information for feature engineering.
  Such titles were cleaned and special characters were removed for better feature engineering. This title was transformed to ***anmyna complaint silky set shampoo 0 ml conditioner 0 ml.***

- The description and title were wrapped and formatted in HTML elements. We parsed this HTML formatting and cleaned the titles and descriptions such that we obtained only the actual text that we need to extract features.
  Example: The dataset contains the following description, ***<ul> <li>100% Authentic</li> <li>Refresh and brighten skin</li> <li>Anti-wrinkle and deep cleansing effects</li> </ul>***. The text was parsed and html tags were removed to obtain only the useful information. This title was transformed to ***0 authentic refresh brighten skin anti - wrinkle deep cleansing effect.***

- Once all the special characters and unwanted tags were removed, we brought the text to a uniform standard such as all lowercase characters. This step ensured a uniformity among all the text data.
  For eg: The words - “RED” and “red” may be considered as different elements in regular expressions or TF-IDF models if we do not convert it into uniform case.
  We have examples like the following in our database where the colors are not in uniform fashion: ***NIKE AIR MAX 1 ESSENTIAL ACTION RED BLACK WHITE UK7.5*** vs ***SMZ658 Professional 1.1M In-ear Headset Perfect Hifi Sound Earphone Flat Wire Good Sound Insulation (Red)*** vs ***Silicone Lokai Bracelet Purple-pink-blue Camouflage-Size L***

- We also observed that the price of each product was in the local currency. We converted this local currency into a normalized currency. We picked Singapore Dollar as a uniform measure. Specifically, the dataset given to us contains currency from three different countries - Malaysia, Philippines and Singapore. We converted each of these currencies to Singapore Dollar for a uniform standard.

- The data was checked for incomplete/missing features in a row. This problem can be handled in multiple ways. Either remove the entire row if missing data is below a predetermined threshold or take the average of the data and fill the feature.

- We removed the stop words from our training and testing data as they do not contribute towards Clarity or Conciseness. A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that would affect our model since we’re using TF-IDF frequent words as features and stop words are generally more frequent in datasets. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. Example: ***Phone case for iPhone 6/6s wan Love cover for Apple iPhone 6 / 6s*** became ***phone case for iphone 0 0 wan love cover for apple iphone 0 0***

- We also remove all the numbers from the titles and replaced them with the number 0
  Example: ***iPhone 6/6s gold edition 32GB brand new [unlocked]***
  became ***iphone 0 0 gold edition 0 gb brand new unlocked***

Challenges faced in Data Preprocessing:

- Even after data preprocessing and applying all the above discussed techniques, we found a few cases which were not being handled by these techniques. This was due to poor text representation such as non-stop concatenation of the entire string which lead to ineffective string preprocessing. Example: ***wireless wifi ip p2p alarm clock spy hidden cameradigitalvideorecorder surveillance*** and ***pattern clear tpu silicone gel back skin soft case for asus zenfone 0 multicolorexportintl.***

- There were some titles which contained only stop words or numbers which became invalid to begin with. Two such examples in our training data are titles ***2016*** and ***Ms.*** These titles specifically cause problems when using tyring to calculate entropy, word vectors and also have a negative effect on the model performance.

- And there were a few words in the title and description which contained some common typos in them. In one of the examples, the word android was misspelled: ***reasonable price durable practical top sale item product size 0 x 0 x 0 mm cpu amlogic quad core light source osram led andriod44 andriod44 led lifetime 0 hour***

- Also, there were some titles which contained words not in the English vocabulary. But due to our feature engineering, such non-English words are automatically well. For example, the tile ***yinglunqishi men fashion sneaker board shoe lover jc04 COLOR*** contains an out of vocabulary word.

2. Feature Engineering

The features in our data will directly influence the predictive models we use and the results achieved. Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. Feature engineering is fundamental to the application of machine learning, and is both difficult and expensive. The better the features that one prepares and chooses, the better the results we will achieve.

1. **Text Classification using Word Vectors**
   We have experimented with the word2vec model for representation of words to vectors. Word2vec is a group of related models that are used to produce word embeddings. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. The basic idea is that semantic vectors (such as the ones provided by Word2Vec) should preserve most of the relevant information about a text while having relatively low dimensionality which allows better machine learning treatment than straight one-hot encoding of words. We used the pre trained word vectors google news word2vec model for converting our titles to word vectors.

2. **Grouping by Category**
   We ran the models on two sets of features. They are as follows.
   - First we extracted the feature set from the whole data space.
   - Second we broke down the dataset into smaller subgroups based on the level -1,2,3 category specified in the problem and work on these smaller datasets instead. The intuition for this approach is that, the factors that contribute to a clear and concise title varies greatly from category to category. For eg: Color may be of great importance in the Fashion and Clothing category but is not that important while buying storage devices like flash drives.

3. **Term Frequency–Inverse Document Frequency (TF-IDF)**
   TF-IDF is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf value increases proportionally to the number of times a word appears in the document, but is often offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general. We ran this model on the both title and description column. For the title column, we prepared a list of 2700 most common words appearing across titles from all categories and found number of intersecting words in each and used it as a feature. We also found the number of intersecting words for each title with list of tfidf words prepared from product descriptions as well.

4. We extracted and added intuitive features such as the length of title and description to compare any relation between the two.

5. Cosine distance between word vector of title and intersecting words and word vector of title and important words like location, brand, unit, shipping, color, sexy.

6. We extracted three features out of the product ID column in each title. ASCII values of characters at positions 1-5, 6-7, 8-9 represented three features which gave information about brands of products. As an example, the prefix product ID ***OD580EL*** belongs to a brand ***OddStickers*** and category Electronics and prefix product ID ***NP819EL*** belongs to brand ***N-power*** and category Electronics (denoted by EL, the 6th and 7th character). Similarly, the product with ID *NO990**HL**AA* belongs to the category ***Home and Living*** (denoted by ***HL***, the 6th and 7th character) whereas *NR633**HB**AA* belongs to category ***Health and Beauty***.

7. We also designed a number of binary features that helped the model perform better:
    - Contains color: Product titles generally contains what color the product is and this is a very important feature in retail. As expected, more than 60% (22005 of 36283) of our dataset contains what color the product is, making this feature extremely important.
    - Contains keyword 'New': This is intuitively a very important feature contributing towards clarity and is present in 5% of our training examples.
    - Contains measurement unit: Almost 55% of the titles in our dataset contained at least one measurement unit and contributing to our model.
    - Contains number

8. We designed a number of features which contributed towards clear and concise titles:
    - Number of repeating words
    - Number of All digit words
    - Number of non-digit words

9. For the categorical features in the training data, we used label encoding to convert the columns into numeric inputs. One hot encoding is a process by which categorical variables are converted into a form that could be provided to machine learning algorithms to do a better job in prediction. Some algorithms can work with categorical data directly. For example, a decision tree can be learned directly from categorical data with no data transform required (this depends on the specific implementation). Many machine learning algorithms cannot operate on label data directly. Since we tried multiple models, most of which required all input variables and output variables to be numeric, it was an obvious choice. We performed Label encoding on the three hierarchical category features and the product type.

10. Another important feature which contributes greatly towards text classification tasks such as ours is part of speech tagging. Specifically, a clear and concise title should contain appropriate number of nouns and adjectives. Part-of-speech tagging (POS tagging) is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition and its context—i.e., its relationship with adjacent and related words in a phrase, sentence, or paragraph. We used python's spacy library to calculate the following features:

- Number of nouns
- Number of adjectives
- Number of out of vocabulary words
- Number of characters

11. **Entropy of words**

The entropy for all words in the title was calculated using formula below,

$$\sum - p_i * log(p_i)$$

The above formula gives the Shannon entropy and tells us what is the minimal number of bits per symbol needed to encode the information in binary form (if log base is 2). If the entropy is on the lower side, it may indicate the title may have a high chance to be concise since it can be represented with less lower number of bits. Sum of entropy of all words in the title was taken as a feature.

Challenges faced in analysis and training

1. **Clarity and conciseness are subjective.**
   As one can expect, clarity and conciseness are completely subjective terms. A product title maybe clear for one person but completely unclear for another. A person may find a title verbose but another person may need those extra words to gain a clear understanding of the product.

2. **Limited dataset available for training**
   Due to a small dataset of just 36000 rows, we were able to achieve an accuracy of 84%.

3. **No labels for validation and testing datasets**
   Since the competition has been closed, we were not able to get hold of their testing dataset. Hence we used k-fold cross validation technique to split the existing dataset into training and validation, thus reducing our training set even further. And on the submission page, we had to upload our own labels for testing.

4. **Imbalanced dataset** with 94% positive examples and 6% negative examples for Clarity
   When measuring clarity, the dataset was skewed towards positive samples. It accounted for 94% of the entire dataset with just 6% of it depicting negative samples.

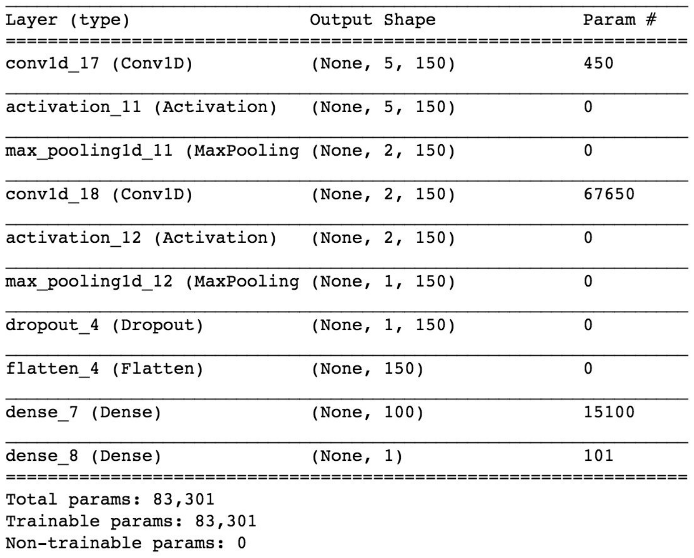
3. Model Selection

1. **Logistic Regression**
   Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). The idea of Logistic Regression is to find a relationship between features and probability of a particular outcome.

2. **Deep Learning**
   Deep learning is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. The contain multiple hidden layers in the neural network through which they are able to learn better representations of the input.

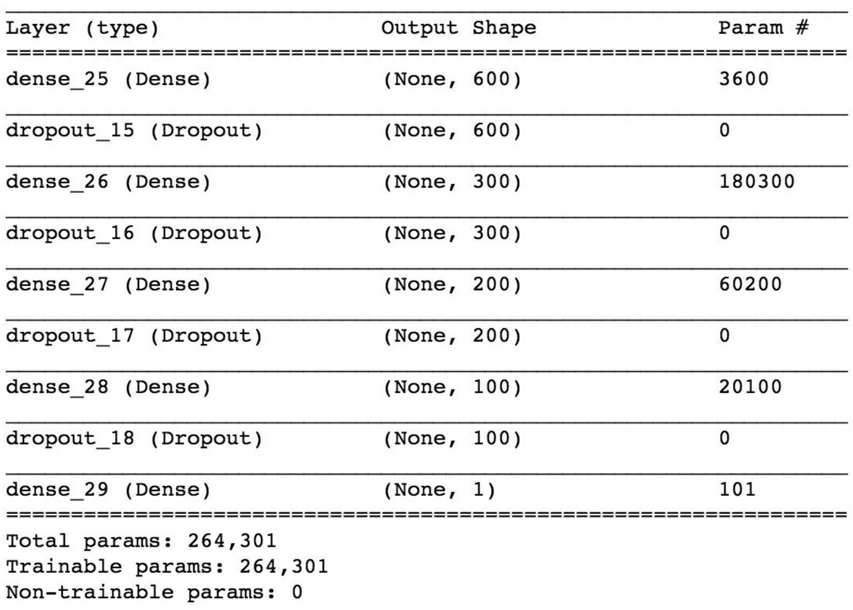
   Architecture of the Network
   A Convolutional Neural Network(CNN) consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers. Convolutional layers apply a convolution operation to the input, passing the result to the next layer. Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. A dense layer refers to a fully connected layer and performs a linear operation on the layer's input vector.

   We first experimented with 2 different architectures,
   - We added 3 iterations of combinations of a Convolutional layer and a max pooling layer. After this, we added a Dropout to control overfitting. Next layer is a dense layer with 200 neurons. The last layer is a dense layer with 1 neuron and a sigmoid activation to output the result.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_17 (Conv1D)           (None, 5, 150)            450       
_________________________________________________________________
activation_11 (Activation)   (None, 5, 150)            0         
_________________________________________________________________
max_pooling1d_11 (MaxPooling (None, 2, 150)            0         
_________________________________________________________________
conv1d_18 (Conv1D)           (None, 2, 150)            67650     
_________________________________________________________________
activation_12 (Activation)   (None, 2, 150)            0         
_________________________________________________________________
max_pooling1d_12 (MaxPooling (None, 1, 150)            0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 1, 150)            0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 150)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 100)               15100     
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 101       
=================================================================
Total params: 83,301
Trainable params: 83,301
Non-trainable params: 0
```

- Our next model consists of 4 hidden layers and an output layer. Each of the 4 layers contain 500,300,200 and 100 neurons and a single neuron in the output layer.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_25 (Dense)             (None, 600)               3600      
_________________________________________________________________
dropout_15 (Dropout)         (None, 600)               0         
_________________________________________________________________
dense_26 (Dense)             (None, 300)               180300    
_________________________________________________________________
dropout_16 (Dropout)         (None, 300)               0         
_________________________________________________________________
dense_27 (Dense)             (None, 200)               60200     
_________________________________________________________________
dropout_17 (Dropout)         (None, 200)               0         
_________________________________________________________________
dense_28 (Dense)             (None, 100)               20100     
_________________________________________________________________
dropout_18 (Dropout)         (None, 100)               0         
_________________________________________________________________
dense_29 (Dense)             (None, 1)                 101       
=================================================================
Total params: 264,301
Trainable params: 264,301
Non-trainable params: 0
```

Effect of batch size:
Using larger mini-batches in SGD allows us to reduce the variance of our stochastic gradient updates (by taking the average of the gradients in the mini-batch), and this in turn allows us to take bigger step-sizes, which means the optimization algorithm will make progress faster. However, the amount of work done (in terms of number of gradient computations) to reach a certain accuracy in the objective will be the same: with a mini-batch size of n, the variance of the update direction will be reduced by a factor n, so the theory allows us to take step-sizes that are n times larger, so that a single step will take us roughly to the same accuracy as n steps of SGD with a mini-batch size of 1.

Adam optimizer:
Adam is different from classical stochastic gradient descent. Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training. A learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

Adam combines the advantages of two other extensions of stochastic gradient descent. Specifically:

- Adaptive Gradient Algorithm (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).
- Root Mean Square Propagation (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

Adam realizes the benefits of both AdaGrad and RMSProp. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance). Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages.

Filter sizes:
We used three filters of sizes of three, four and five in the Convolutional Neural Networks. The filter sizes in case of NLP can be interpreted as the n-gram size. So depending on the average size of the input sentences that you can expect you choose your filter size. After looking at couple of sentences in the title we observed that we cannot choose a very high value for the size of filters as the sentences on an average were not very long.

ReLU vs Sigmoid activation:
The definition of a ReLU is $h = max(0, a)$

where $a = W * x + b * a = W * x + b.$

In addition to ReLU being faster than Sigmoid while training the data, two additional major benefits of ReLUs are sparsity and a reduced likelihood of vanishing gradient. One major benefit is the reduced likelihood of the gradient to vanish. This arises when a>0a>0. In this regime the gradient has a constant value. In contrast, the gradient of sigmoids becomes increasingly small as the absolute value of x increases. The constant gradient of ReLU's results in faster learning. The other benefit of ReLU's is sparsity. Sparsity arises when a≤0a≤0. The more such units that exist in a layer the more sparse the resulting representation. Sigmoids on the other hand are always likely to generate some non-zero value resulting in dense representations. Sparse representations seem to be more beneficial than dense representations.

Glorot initialization:
Glorot initialization finds a good variance for the distribution from which the initial parameters are drawn. This variance is adapted to the activation function used and is derived without explicitly considering the type of the distribution.

Dropout:
Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. It is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods.

Since our dataset for clarity is skewed and we have 94% positive examples, the possibility of our model overfitting the training data is very large. This makes the use of dropping out random neurons very important in our case.

3. **Support Vector Machines (SVM)**
They are supervised learning models used to analyze data for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

4. **XGBoost**
XGBoost stands for extreme gradient boosting. It is an implementation of gradient boosting created by Tianqi Chen. Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It involved creating new models that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models. Like other boosting methods, gradient boosting combines weak "learners" into a single strong learner in an iterative fashion. To learn the set functions for various weak learners of functions, we minimize the following regularized objective

$$L(\varphi) = \sum l(y'_i, y_i) + \sum \Omega(f_k)$$

$where\ \Omega(f) = \gamma T + 0.5 * \lambda||w||^2$

Here l is a differentiable convex loss function that measures the difference between the prediction y'_i and the target y_i. The second term Ω penalizes the complexity of the model (i.e., the regression tree functions). The additional regularization term helps to smooth the final learnt weights to avoid over-fitting.

Model Analysis

Influence of dataset
Even though the data is same, we have used different approaches for Clarity and Conciseness because of the difference in the distribution of the data labels.
Conciseness has a good distribution with 68.5% positive labels and 31.7% negative labels. Clarity on the other hand has a skewed imbalanced distribution with 94.3% positive labels while 5.7% negative labels.

Baseline performance measure
As the competition is currently closed and does not allow submissions anymore, we have chosen a Logistic Regression model with 2 features - the length of title and whether it contains a number or not as a baseline performance measure to compare the performance of our models. Logistic Regression is one the most classic approaches to classification problems.

Metrics for Performance Evaluation
We have used different metrics for evaluation so as to get better insights of the performance of the models.
TP = True Positives
FP = False Positives
TN = True Negatives
FN = False Negatives

Accuracy: Percentage of test data for which our predicted labels match the actual labels.

$$Accuracy = \frac{TP + FP}{TP + FP + TN + FN}$$

Precision: Of all the data that our model predicted as positive, what percentage are actually positive.

$$Precision = \frac{TP}{TP + FP}$$

Recall: Of all the positive examples in the test set, what percentage did our model predict as positive

$$Recall = \frac{TP}{TP + FN}$$

F1 score: Combined metric of precision and recall

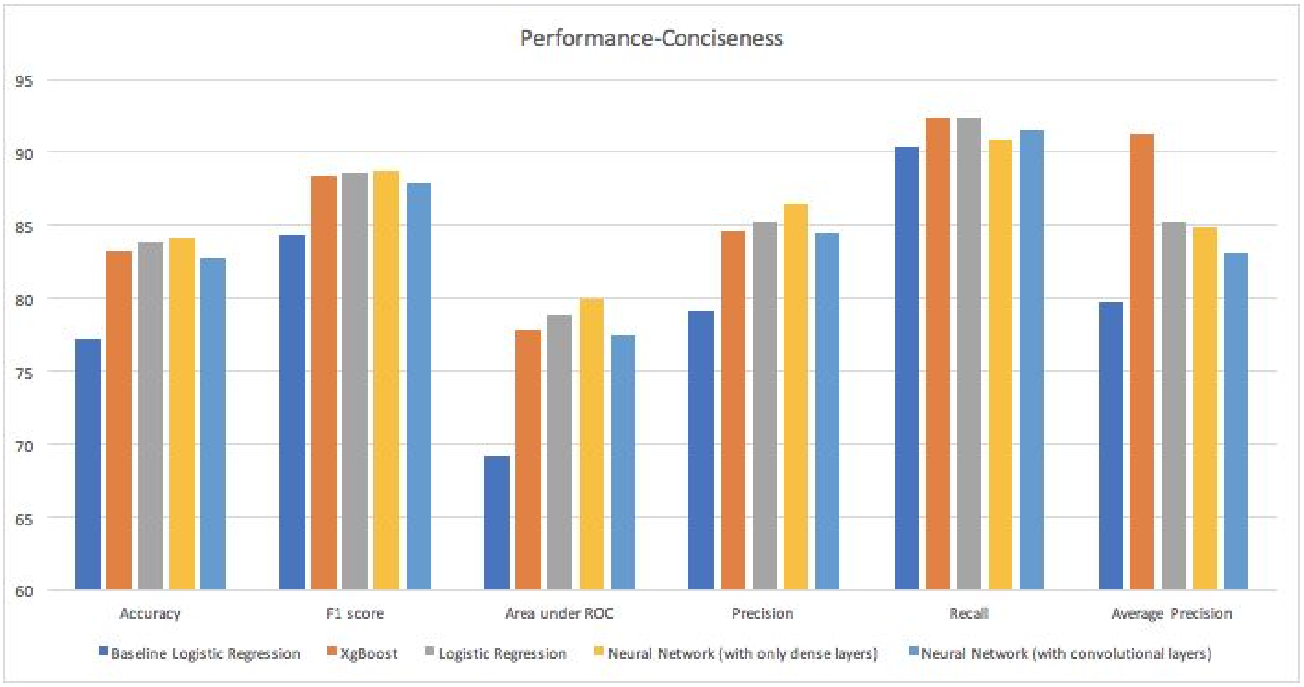
$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

Area under ROC: It gives the % of area under the Receiver Operating curve.

Average Precision: It gives the % of area under the Precision Recall curve.

Modelling for Conciseness

- We have experimented with 3 models - Logistic Regression, Xgboost and a neural network model.
- As extension of the baseline model, we used logistic regression with the full feature set to get an improvement in performance. This gave us an insight in the effectiveness of the feature engineering process.
- We then proceeded to experiment with some other models which might be more suitable for the dataset and started with modeling our neural network.
- As this problem has some traits of text classification, we experimented with neural networks with a combination of convolutional layers and dense layers.
- On repeated tweaking of the hyperparameters the model, we finalized on the model which gave us the best results.
- A deep net can capture things like image, audio and possibly text quite well by modeling the spatial temporal locality. While tree based models solves tabular data very well and have some certain properties a deep net does not have such being easier to interpret and invariant to input scale, much easier to tune.
- On looking at the overall success of the XgBoost technique(a tree based model) in various machine learning problems & competitions we choose to experiment with XgBoost for our problem as well.
- The XgBoost model worked very well for this problem statement and had results comparable to the neural network. For the metrics of recall and average precision, It gave a result even better than the neural network. So choosing XgBoost was definitely a wise decision.
- After having a close look at the various columns available in the data we observed that there maybe scope for some sort of clustering or grouping in category columns. We tried to group the data according to Category 1 column and trained a classifier for each of the groups.
- As the dataset is comparatively small, we did not get good distributions in these groups and did not get good performance on the individual models.
- The results of the different evaluation metrics are compiled in the graph below. It shows a good comparison between the performance of different selected models.



Modelling for Clarity

The main problem of modelling for clarity is tackling the skewed dataset. Directly using any model does not give us good results.
The main objective of balancing classes is to either increasing the frequency of the minority class or decreasing the frequency of the majority class. This is done in order to obtain approximately the same number of instances for both the classes.
There are 2 ways we can handle such imbalanced classes.

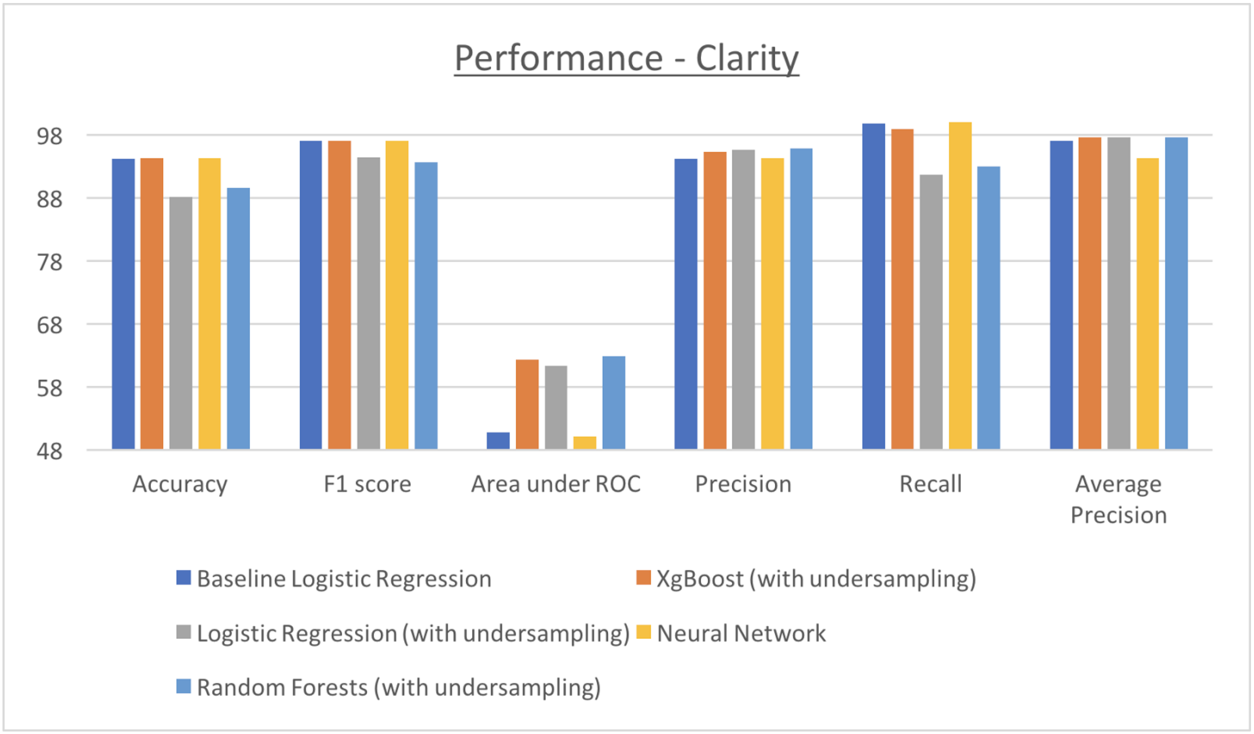
1. Random Undersampling aims to balance class distribution by randomly eliminating majority class examples. This is done until the majority and minority class instances are balanced out.
2. Over-Sampling increases the number of instances in the minority class by randomly replicating them in order to present a higher representation of the minority class in the sample.

Tackling skewed dataset

- To tackle this skewed dataset we use undersampling technique.
- We divide the majority class into N partitions and club each partition with the minority class to get a balanced dataset.
- We then build individual N models for each partition using techniques discussed below.
- We find the class prediction using each of the N models and take a majority vote to get the actual predicted label for a test input.
- This method gives us a better result than directly building a model from the entire dataset.

Modeling technique

- We have experimented with 3 models in this case using undersampling technique - Logistic Regression, Xgboost, Random Forests and also a neural network model for the entire dataset without undersampling.
- Xgboost and Random forests are especially used because they work well with skewed datasets.
- In most performance measures, the random forests model works at par with the logistic regression model.
- We have used a similar neural network in this case as the model for conciseness.
- We choose the XgBoost model when we got a low score with the area under ROC parameter for the neural network. The XgBoost model performs well in this respect as well.
- In case of clarity modelling, the XgBoost technique with undersampling gives the best results - even better than the neural network.
- Grouping of datasets according to the category does not work here as we already have a skewed dataset and further splitting it does not make sense.
- The results of comparison between the models have been compiled in the graph below.



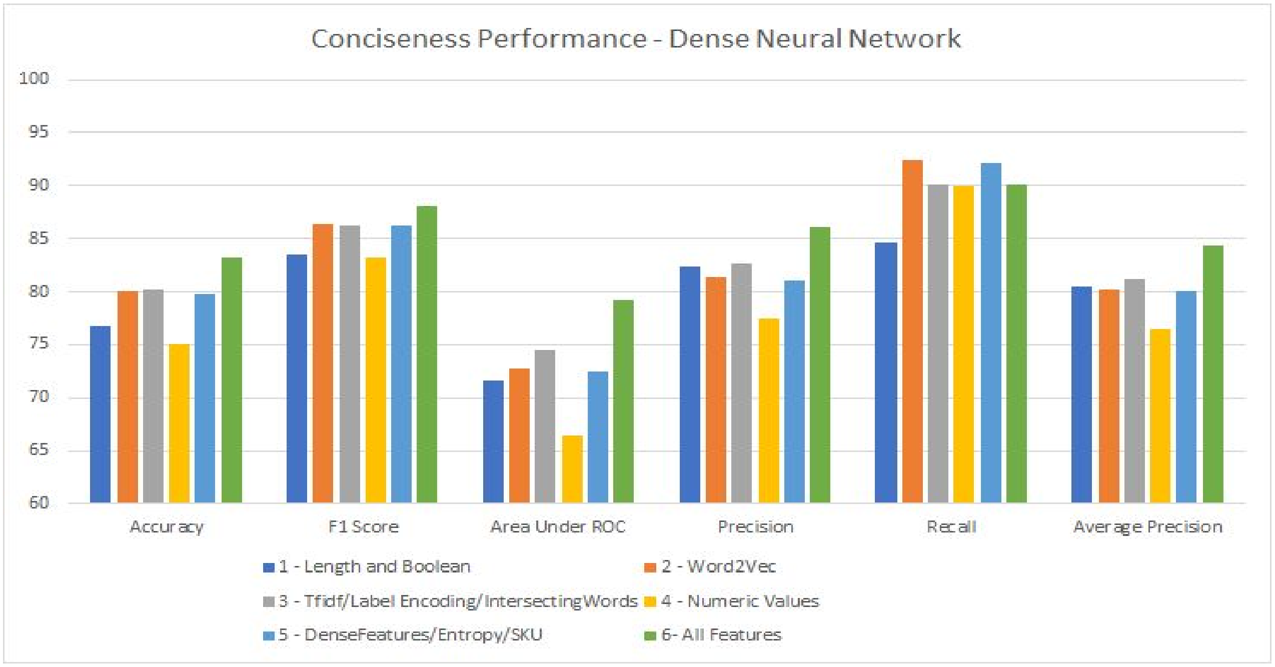
Analysis of Features

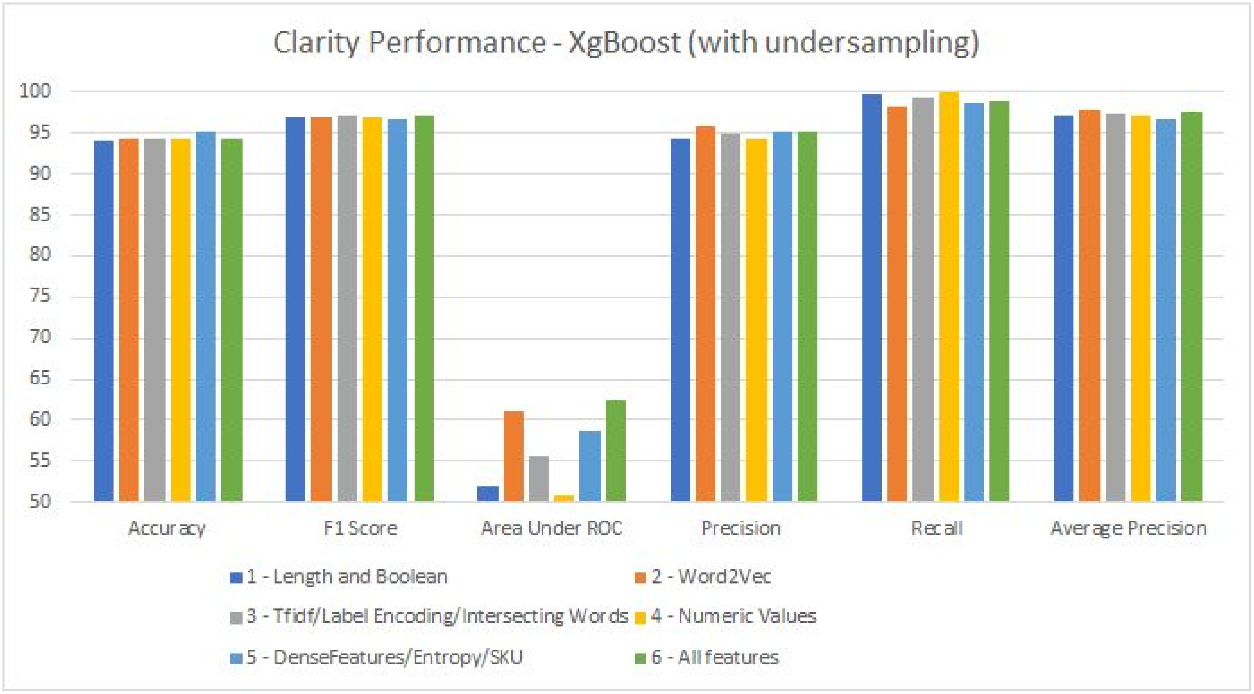
In addition to comparing the performance of different models, we also compare and analyze the performance of different sets of features within the models. As there are many possible permutations of features, we have classified the features into five logical groups. These groups are as follows:

- **Word2Vec embeddings** - This group includes the 300 dimensional word to vector embeddings that were generated using the Google news vectors pre trained corpus.
- **Boolean features and Length** - This group includes all simple boolean features such as existence of color, number, dimension units 'new' keyword in the title as well as the length of the title itself.
- **Tfidf, Label Encoding and Intersecting words** - This group includes the 2700 dimension tfidf vector representing which words from the top 2700 exist in the title, Label encoding for the 3 - level category as well as the number of intersecting words in the title and tfidf list.
- **Numeric Features** - This category includes numeric features such as the number of words, digits, non digit words, repeating words and normalized price.
- **Dense features, Entropy and SKU** - This set includes dense features such as ratios of nouns, adjectives, characters per words etc, entropy feature of the title, and different substrings of the SKUID.

For analyzing the contributions of the individual features in each group we ran our tests on the best performing models for each category i.e. Clarity and Conciseness to see the contribution of each set of features to the model.

The results are as follows:





Analysis :

- From the above graphs it is clear that the performance of all the features combined is better than the performance of any individual feature group for almost all performance metrics.
- Within feature groups it was observed that word2vec embeddings, Tfidf/Label Encoding/ Intersecting words and Dense features/Entropy/SKU are the features that have a fairly good performance even by themselves. The intuition behind their performance is as follows:
    - **Word2Vec Embeddings** is arguably one of the most important features as, not only does it provide a way to map words to vectors, the vector values after mapping actually signify the relation between words. Words that are closely related have similar embedding values as opposed to those words which are completely different. Both in the context of conciseness as well as clarity this is a very important feature as products with clear and concise titles would have words that are more closely related than arbitrary words.
    - **Tfidf/Label Encoding/Intersecting** words is an important group as it finds the number of important words in a title, which words from the tfidf list exists in the title and label encoding. There are certain characteristics such as the common occurrence of certain words in a title and the occurrence of multiple tfidf words in a title contributes to the clarity and conciseness.
    - **Dense features** contain the ratio of nouns, adjectives etc. So a title with a balanced ratio of nouns to adjectives suggest a good title. If there are only nouns then there are no

adjectives to describe the product while if there are many adjectives, then it could imply addition of unnecessary adjectives just to make the title catchy.

Challenges:

There were multiple features that we tested but did not include in our final model as it did not help improve our model despite. This section will discuss the features that were tested and will also analyze some of the reasons behind why they may not have worked.

- **Grouping by category:** The initial reason why we thought that grouping by level 1 category would be a good idea is because of the intuition that the decision on whether a title is clear or not heavily depends on the category of the product as certain information like color may not be relevant for a category like health products but might be more important for clothing.
    - Clarity metrics were not improved by grouping by category because of the following
        - The dataset given in the problem is small and grouping my category makes each sub dataset even smaller. On dividing the dataset into 9 level - 1 categories, each category on average has 4000 rows.
        - The problem is exacerbated by the fact that we do not have access to testing results and have to use our training data and perform a cross validation on it, which reduces the number of training examples to about 3000.
        - The skewed nature of the dataset results affects the performance of the model even more for small datasets, especially in predicting the minority class (unclear) and undersampling method reduces the number of majority class examples used for training which further cuts down on an already small dataset.
    - The above mentioned reasons led to a poor performance of the clarity model on the grouped categorical data, especially with metrics such as area under the ROC curve. (as accuracy is not a good metric for the problem)
    - Additionally, grouping by category does not help conciseness because the conciseness of a title has a lesser dependence on the category itself and more on features such as the length, use of nouns, adjectives etc. in a title, which can be better trained on the whole dataset.
    - Moreover, the label encoding feature which performs a one hot encoding on the different category levels is an important feature in our final models and with both XgBoost as well as deep neural networks the models may be able to infer correlations between the different categories and some of the other features which to some extent helps the model learn features which include this category information.

- **Including product description as a feature:** The description by itself does not lend to the clarity or conciseness of the title but the intuition was to analyze the important words in the description and checking for its occurrence in the title. The way we executed this was by calculating the tfidf word list for the description and then computing the number of intersecting words between this list and the title itself. But this feature did not improve the accuracy of our model and the results were consistent with the models without this feature.

    - The above results clearly show that this feature does not add any more useful information given our set of existing features. One reason is that the descriptions given in the current dataset is very noisy and contains a lot of irrelevant information and excess information and the tfidf for the description may not be representative of the important words whose existence in the title makes the predictions easier.
    - The tfidf list for descriptions trained on a specific category might be more relevant to the problem mentioned above but due to issues with grouping by category as explained in the previous section, this approach could not be adopted as well.

- **Feature Normalization -** Normalizing features is extremely important for any machine learning problem as not normalizing the features which often includes data in different ranges, with different data distribution leads to some features (typically with higher values) to influence the model much more than some features with a lower range of values even though these features might contain a lot of information. This issue with normalization is mostly prominent with regression based models and SVM.
    - The reason feature normalization may not always aid improve the model accuracy is because the final models used by us include dense neural networks and xgBoost. In neural networks, using normalized vs unnormalized data could be considered equivalent to adjusting the weights and biases associated with those features and the model (assuming a linear form of normalization) but often helps faster convergence and easier training while using activation functions like sigmoid etc . As observed in our model normalizing the features did not help improve the performance of the model.
    - Often normalization may decrease the performance of the model. This is because certain features do not scale well and finding a right kind of normalization is very important to ensure that the relationship between the features is preserved. Eg: word2vec embedding values represent a very specific detail and correlation between any two word vectors and a z normalization may actually result in loss of some information especially when the distance between the word vectors plays an important role.

- **Cosine distance between words:** Even though intuitively the cosine distance between the word vector of the title and the word vector of the frequent tfidf words makes sense and word vector of important words like location, brand, unit, shipping,color, sexy, it did not contribute to increasing the accuracy of the model.
    - There were many title word vectors that were not of the generic form “McDonald's Coke Can Glass Limited Edition 12oz Purple Color”.
    - Additionally the word2Vec trained corpus contains the names of a lot of famous brands in the west but does not contain a lot of brands from the malaysian, singapore and philippines market.

References:

[1] Why we used Glorot initializer: http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf
[2] Dropout: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf
[3] Product Title Classification vs Text Classification: https://www.csie.ntu.edu.tw/~cjlin/papers/title.pdf
[4] Deep and wide learning for clarity and conciseness classification:
https://github.com/lampts/cikm17_cup_lazada_product_title/blob/master/cikm17_lazada_report.pdf
[5] Bagging model for product title quality with noise:
http://ls3.rnet.ryerson.ca/wp-content/uploads/2017/10/CIKM_Analytic_Cup_2017.pdf
[6] Deep Learning, NLP, and Representations
http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/
[7] Understanding Convolutional Neural Networks for NLP
http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/
[8] https://competitions.codalab.org/competitions/16652#learn_the_details
[9] The Relationship Between Precision-Recall and ROC Curves:
http://pages.cs.wisc.edu/~jdavis/davisgoadrichcamera2.pdf
[10] Understanding Boosted Trees Models: https://sadanand-singh.github.io/posts/boostedtrees
[11] Learning from Imbalanced Classes: https://svds.com/learning-imbalanced-classes/
[12] Keras usage: https://github.com/fchollet/keras/blob/master/examples/imdb_cnn.py