A Kaggle Competition: predict pet adoption speed

He Zi, Qinyan Li, Menglai Li, Weizhe Xing and Hailong Zeng
Department of Electrical Computer Engineering
University of California, San Diego
March 17th, 2019

Abstract

This paper is a review of a Kaggle competition on predicting pet adoption rate based on data provided by PetFinder.my. First, since the data consists of tabular, text and image, various feature selection and extraction methods were explored to make the most of all the three types of data. Then, the selected/extracted features were combined to train classifiers, where three different learning algorithms are used: light gradient boosting machine (LightGBM), extreme gradient boosting (XGB) and support vector machine (SVM). Next, exhaustive experiments were carried out to cross-validate the parameters for different models to achieve the best performance on test set. Finally, the performance of different algorithms were compared.

1. Introduction

For this Kaggle competition, the background is that a Malaysia's animal welfare platform called PetFinder.my is trying to predict how fast a pet will be adopted based on the data they have collected. The database contains information of more than 150,000 animals. There are three types of data: tabular, text and image. There are also videos. But due to the fact that there are very few pets having videos. They are omitted. The tabular data contains 24 different features, which are age, type, name, breed, gender etc. These data are mainly discrete or categorical. Be aware that the text data is also one column embedded in the table. Most pets have one or several images. The output is the adoption speed in discrete values: 0-4, which indicated how fast each pet is adopted from the fastest to the slowest. For example. 0 means that the pet was adopted on the same day as it was listed.

Our goal here is to develop algorithms to predict the adoptability of pets - especially how quickly a pet is adopted? If successful, the algorithm can be adapted into PetFinder's AI tools, which will guide shelters and rescuers around the world on improving their pet profiles' appeal, reducing animal suffering and euthanization.

2. Problem Formulation

The main difficulty of this project is that we have three types of data. The question is how to effectively select or extract features from these data and how to train a classifier based on features from three different types of sources.

2.1. Tabular data

Although the tabular data is generally tidy and clean, some features seem to be irrelevant to the classification problem. So some features can be omitted, like Rescuer ID and PetID. Also, the description part in the table should be tackled independently.

2.2. Text data

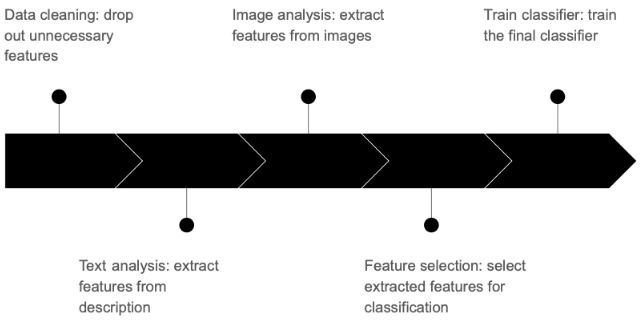
The text data is a profile of each pet. The language is mainly in English with only a very small portion in Malay or Chinese. The text data was already run through Google's Natural Language API to produce a sentiment data file. Since the analysis of the text data could be classified as a sentiment analysis type of problem. So perhaps first it is worthwhile to extract some features from this sentiment data and experiment with it to see how well it performs with different classifiers. But aside from sentiment analysis, can we still try to extract other features from the text?

2.3. Image data

The image data can show the potential adopter how the pet is really like. As a result, we believe there is a strong correlation between the image and the adoption speed. The image data was already run through Google's Vision API to produce a Metadata file containing information about Face Annotation, Label Annotation, Text Annotation and Image Properties. So as above, perhaps we can try to use these features in out classifiers first. however, it is also necessary for us to try to extract some features by ourselves from the images, not only to experiment on new features, but also as a comparison to the standard high throughput tool like Google's Vision API. So the question remains that how can we extract the features from the images which are also highly relevant to the classification perspective.

2.4. Build a classifier

Once we have all the features prepared, the next question is how to utilize them all together. Depending on how we combine these features, the optimal classifiers could be different and the optimal parameters for the classifier could also be different.



Data cleaning: drop out unnecessary features
Image analysis: extract features from images
Train classifier: train the final classifier
Text analysis: extract features from description
Feature selection: select extracted features for classification

Figure 1. The diagram of our idea

3. Proposed Solution

Based on the questions listed above, here are our proposed solutions.

3.1. Text feature

So first of all, the sentiment data generated from Google's Natural Language API was explored. The sentiment analysis inspects the given text and identifies the prevailing emotional opinion within the text, especially to determine a writer's attitude as positive, negative, or neutral. It attempts to determine the overall attitude (positive or negative) expressed within the text. Sentiment is represented by numerical score and magnitude values [1]. More detailed introduction can be seen from the Google Natural Language API. The sentiment data provides results on 'documentSentiment', 'language' and 'sentences'. The 'documentSentiment' contains the overall sentiment of the description, which includes two sub fields: score and magnitude. The 'language' contains the language type. The 'text' separates the description into sentences and gives the score and magnitude of each of the sentence. All these features were extracted for utilization in classification. To account for the variations of sentiment among the sentences, features like score/magnitude mean, standard deviation, range and summation were also computed as features.

Aside from doing sentiment analysis, next we start to consider other analysis methods of text for feature extraction, for example, semantic or syntactic analysis. One of the most popular representation of document vocabulary is word embedding. It is very efficient in learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships [2]. One of the most widely used technique to implement word embedding is called Word2Vec, which was developed by Tomas Mikolov in 2013 at Google. Basically, word embedding is to use a dense vector to represent a word instead of using a large sparse vector to represent the entire vocabulary. The simplest model for such word embedding is a continuous bag of words model with only one word in the context as shown in the figure below [3]. The input is a one-hot encoded vector. In this project, We used pre-trained vectors from another similar model fasttext to get word embedding from the description text and then feed them into a neural network, for example a convolutional neural network (CNN). Please note that here only features extracted via word embedding were used for classification. This step is mainly for exploration of the importance/relevance of the extracted features to the classification objective.

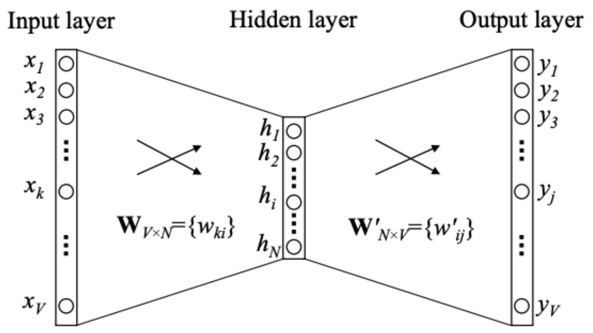


Figure 2. A simple CBOW model

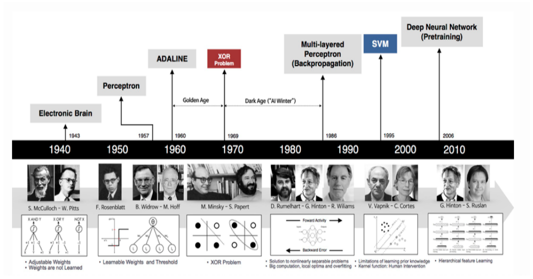
To evaluate the relevance or importance of the words in the description, we also used a information retrieval method called TF-IDF. It is short for frequency-inverse document frequency. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus [4]. Specifically, TF-term frequency, measures how frequently a term occurs in a document. IDF-inverse document frequency measures how important a word is. This quantity actually give less weight to words that appear quite often but might be less meaningful and scale up the rare words. There is one more thing here that need to pay attention to. The TF-IDF results for the description text is a large sparse matrix. As a result, we have to implement dimensionality reduction. Here the dimensionality reduction technique we used is singular value decomposition (SVD).

3.2. Image feature

From the method shown above, in order to solve this problem, the image feature should be extracted from image to help figure out.

In order to realize the goal of feature extraction, some techniques should be introduced.

As we all known, deep learning, as a branch of Machine Learning, can be tracked back to 1943 when Walter Pitts and Warren created a computer model based on the neural networks of human brain. And the development history of the deep learning can be shown in the following figure:



Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
ADALINE
XOR Problem
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

Figure 3. Milestones in the Development of Neural Networks[1]

As the above figure depicts, the core of deep learning were in place by the 80-90s when the multi-layered perceptron was introduced. And in recent years, it is widely applied in many fields, such as Natural Language Processing, Computer vision, image processing and so on.

This paper here are mainly about the application of image processing—feature extraction. The details of feature extraction are as follow:

3.2.1 Extraction of interested part in image

Main method:

For every image in the training set, it is not hard to find out that only a small part of the image containing pet's body. In order to extract the pet part from the image, we use a small subset of the training set which are labeled by ourselves to train a Cascade Object Detector. The trained detector can segment an image based on classification result of whether this part is a pet or not.

The main idea of this detector is similarly to gradient boosting. It consists of stages, where each stage is an ensemble of weak learners. The weak learners are simple classifiers called decision stumps. Each stage is trained using boosting. Boosting provides the ability to train a highly accurate classifier by taking a weighted average of the decisions made by the weak learners. The stages are designed to reject negative samples as fast as possible and use the error from each stage to minimize the cost.

Implementation:

We apply the Cascade Object Detector via Matlab's computer vision toolbox. The results are shown in the table:

Table 1. Accuracy of different combinations of Num and Feature

$Accuracy$ \ $Feature$ / Num	$Haar$	LBP	LOG
5	63.2%	60.8%	55.3%
10	73.5%	63.2%	57.8%
15	74.4%	70.1%	58.2%

The hyper-parameter we are going to tune is Number of Cascade Stages and the type of feature. According to the accuracy of different combination of the two parameters, we finally choose:

Number of Cascade Stages = 15

type of feature = Harr

For an image in test set the area of region chosen by the detector has more than 60% overlap rate compared with ground truth, then the number of correct segmentation increase one. We can compute the accuracy based on the ratio of correct segmentation over total number of image in test set.

The interested part of image here is just the whole body information of pet, not included the background part. Just let it be our training image to obtain the image feature in the next part.

And one of the results is as follow:

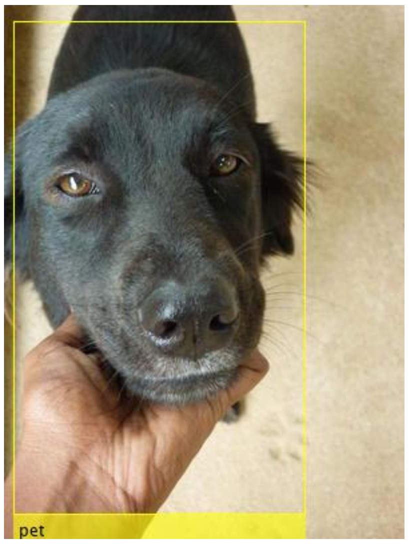


Figure 4. Objection Example

3.2.2 CNN to feature extraction

Architecture:

Our model is a simplified version of VGG16 which achieved the ILSVRC14 2nd in classification, 1st in localization. Compared with VGG16, our network cropped out the last part of the convolution layer and reduce each layer's dimension to a lower number.

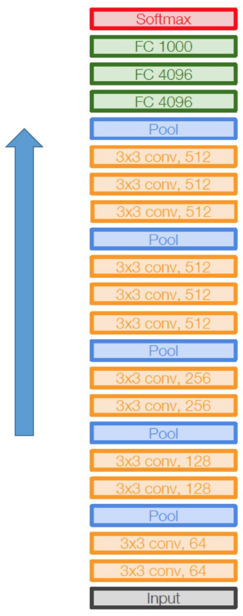


Figure 5. Architecture of VGG 16

The reason we choose VGG16 as our baseline, is that we apply popular neural network namely, Alexnet, VGG and Densenet pre-trained via ImageNet to our training set. We find out that VGG16 achieve the best performance.

The detail of our simplified model's architecture is as follow:

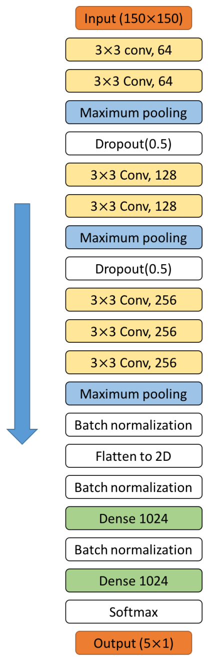


Figure 6. Our simplified VGG

From the figure above, the simplified VGG model are made up of five different layers:

1)Convolution 2D layer:
A layer that consists of a set of filters. The filters take a subset of the input data at a time, but are applied across the full input (by sweeping over the input). The operations performed by this layer are still linear/matrix multiplications, but they go through an activation function at the output, which is usually a non-linear operation.
2) Maximum pooling layer:
Replace each patch in the input with a single output, which is the maximum (can also be average) of the input patch.
3) Dropout layer:
A layer drop out some data to stop overfitting.
4) Batch normalization layer:
Scale the input so that the output has near to a zero mean and unit standard deviation, to allow for faster and more resilient training.
5) Dense(Fully connected) layer:
A linear operation in which every input is connected to every output by a weight (so there are $n_{inputs} * n_{outputs}$ weights - which can be a lot!). Generally followed by a non-linear activation function.
Implementation:

Before training, we do preprocessing of the training image such rotation, mean normalization, standardization, and whitening to try to augment our image data, from like about $50,000$ images to around $100,000$ images, which can help stop overfitting.

After applying to the CNN we find out that it is pretty hard to make a very high accuracy maybe because our model is too simple and the image set is not that easy to classify.

The training and validation loss is given below:

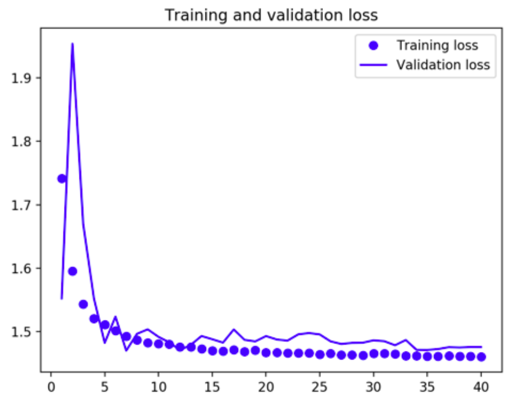


Figure 7. Training and validation loss

It can be told that the network is finally converge but the loss is still high compared with regular result of CNN. The final probability of error of our model is 60%.

By not implementing the last layer—softmax, we can obtain the features of image, which is a vector representing the image features.

3.3. Model training

Once we have our features successfully extracted, the next step is to combine all these features as a whole and train a classifier using these features. Since the analysis of both text and image gives us some new features, our idea to tackle all the three types of data by combining the features from text and image into the table. As a result, now we have a new table with more features.

Nowadays, it is very popular to use gradient boosting machine (GBM) for efficient implementation of machine learning tasks for tabular data. So our approach is to use two GBMs, which are light gradient boosting machine (LightGBM) and extreme gradient boosting (XGB). Also, comparison with another commonly used method - support vector machine (SVM) was carried out.

XGB was created by Tianqi Chen. It is an optimized distributed gradient boosting method. It is based on parallel decision trees so that it is very efficient. Another major difference to GBM is in that it uses a regularized learning objective. So as denoted by Tianqi Chen perhaps it is more appropriate to refer to it as regularized gradient boosting [5]. The equation below gives the regularized objective of XGB.

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$
$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2$$

Figure 8. Regularized learning objective for XGB

LightGBM is another popular GBM based on decision trees, which was shown to be even more time efficient in the cases where the feature dimension and data size is really large. There are two novel features of LightGBM when compared with common GBM. They are gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). GOSS bascially means that only a small proportion of the data which generates large gradients is used to estimage the information gain. As a result, it can achieve a quite accurate estimation of the information gain while only using small data size. EFB is to bundle mutually exclusive features to reduce the dimension of the feature space. As indicated by Guolin Ke et.al, lightGBM speeds up the training process of conventional GBDT by up to over 20 times with almost the same accuracy [6].

4. Experiments and Results

4.1. Model training

We trained the classifier based on the combined table of all the three types of features from tabular, text and image. We exhaustively selected the parameters of the models by cross-validation. The best results measured in the weighted Kappa score from XGB is 0.4414 for cross-validation mean and 0.3702 for the test set. For LightGBM, we achieved the cross-validation Kappa score mean as 0.4476 and Kappa score as 0.3797 for the test set. The learning cureve of both XGB and LightGBM are shown as follows. Also shown below are the relative importance of different features.

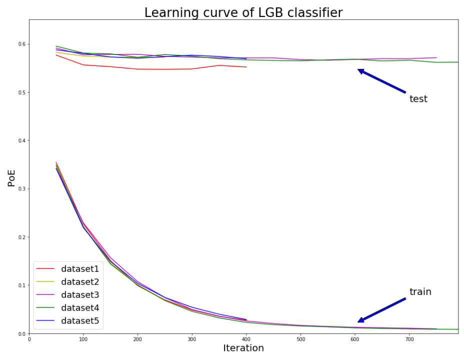


Figure 9. Learning curve for LightGBM

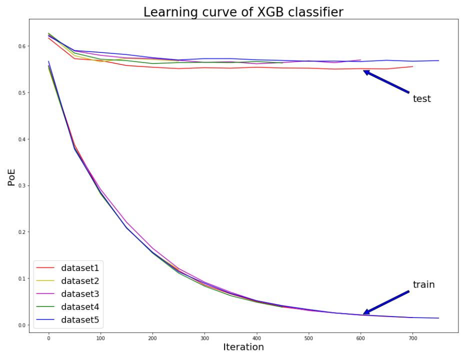
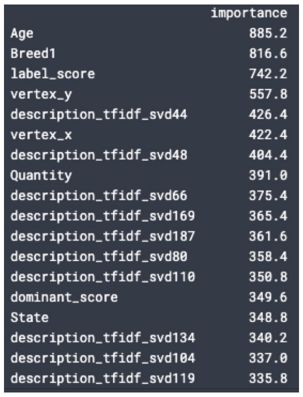


Figure 10. Learning curve for XGB



	importance
Age	885.2
Breed1	816.6
label_score	742.2
vertex_y	557.8
description_tfidf_svd44	426.4
vertex_x	422.4
description_tfidf_svd48	404.4
Quantity	391.0
description_tfidf_svd66	375.4
description_tfidf_svd169	365.4
description_tfidf_svd187	361.6
description_tfidf_svd80	358.4
description_tfidf_svd110	350.8
dominant_score	349.6
State	348.8
description_tfidf_svd134	340.2
description_tfidf_svd104	337.0
description_tfidf_svd119	335.8

Figure 11. Feature importance in LightGBM

The result from support vector machine (SVM) is not as good as LightGBM and XGB. The best cross-validation mean kappa score is 0.1321 though different kernel density with various parameter values were used for cross-validation. Possible reasons are: first, the four classes of adoption speed is not evenly distributed. A large majority of the pets were adopted for longer periods, namely the most common labels are 3 and 4. It is likely that SVM tend to pick the most dominant class to optimize the loss. Another possible reason is that the features extracted are still not representative of the data set. But based on the performance of two GBM and SVM, we believed it is more likely that it is due to the first reason. Resampling was tried to solve this problem, but the result is till not good. Possible reason is that SVM is a large margin classifier that purely depends on a small portion of the support vectors.

Afterwards, we tried to use solely the tabular data to train the classifier. The learning curve is as follows. Here the curve shows that the error decreases as the iteration goes. So we believe that the table data is strongly correlated to the predictive objective of the adoption speed. But the divergence between train and test set is also becoming increasingly larger. So this indicates that only the tabular data is still not enough for classification.

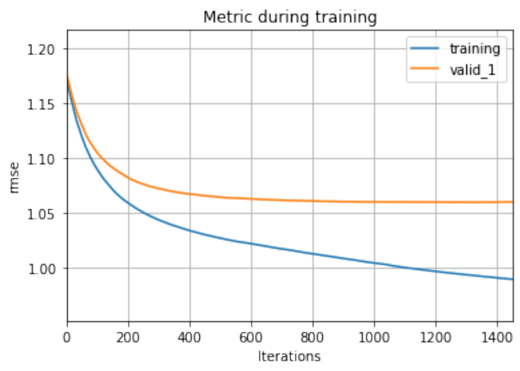


Figure 12. Learning curve only from tabular data

So similarly, after the analysis of the description text, we trained out model solely based on text features extracted. Below are the learning curve for different text features extracted. We can also see that the error decreases as the iteration goes. So all of the features extracted from text data are useful in the classification. But the features should be combined in order to achieve better results.

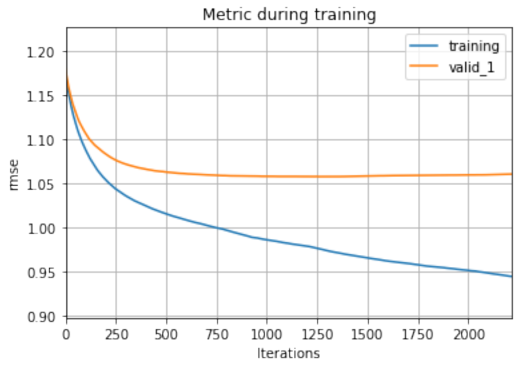


Figure 13. Learning curve from sentiment analysis

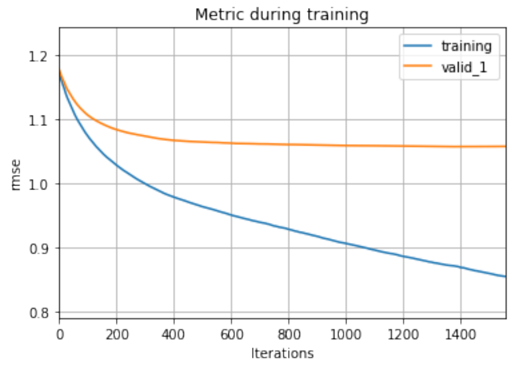


Figure 14. Learning curve from word embedding

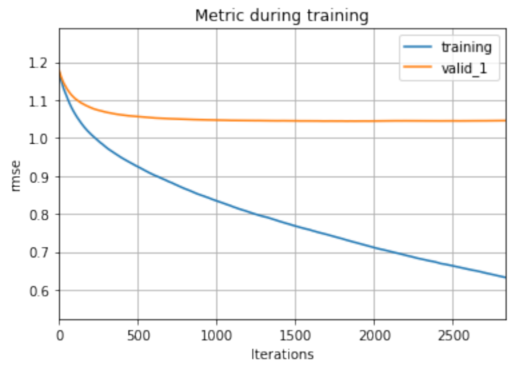


Figure 15. Learning curve from TF-IDF

5. Conclusions

In this Kaggle competition of predicting the adoption speed of pet based on the provided data set, we have implemented various kinds of novel feature engineering techniques, such as word embedding, TF-IDF for text analysis and using deep neural network for image feature extraction. For the model training, two state-of-the-art gradient boosting algorithms: XGB and LightGBM were utilized and compared with another commonly used algorithm

SVM. Also, different parameters of the models were tested via cross-validation to help us select the parameter with the best performance.

Although the result from our model does not appear to be very impressive, we would like to point out that this Kaggle competition was released three months ago and we took part in it for just one month. Until now, the best weighted Kappa score from all competitors is still below 0.5. We believe what's more important is that we get exposed to many state-of-the-art machine learning algorithms and did a relatively thorough comparison between different techniques to see their performance.

References

[1] https://cloud.google.com/natural-language/docs/basics.

[2] Chen K et al. Mikolov T, Sutskever I. Distributed representations of words and phrases and their compositionality. *ACM Transactions on Information Systems,.*

[3] https://arxiv.org/pdf/1411.2738.pdf.

[4] H. Wu, R. Luk, K. Wong, and K. Kwok. "interpreting tf-idf term weights as making relevance decisions". *ACM Transactions on Information Systems*, 26(3), 2008.

[5] Guestrin C Chen T. Xgboost: A scalable tree boosting system[j]. 2016.

[6] https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf.