Claremont Graduate University, Claremont, California
Advanced Big Data Analytics, Spring 2021

Triplet Mining of Wide-and-Deep Models to Predict News Article Click Behavior

Marc Laugharn

May 15 2021

Instructor ..... Professor Weiqing Gu

1 Introduction

News articles are a great stage for experimentation of machine learning models. They are widely available, often for free, and the corpus is always being expanded. Additionally, news articles contain information that can be analyzed with both cutting-edge and more classical machine learning methods, because in addition to the text itself, news articles often include pre-processed metadata. For my humanities coursework at Harvey Mudd, I have also been exposed to many of the social concerns about "behavioral data mining" of interaction data. I am already quite passionate about learning about machine learning, and being expoesd to the critiques against behavioral analysis, I wanted to learn about the process by creating my own attempt at "behavior mining". I searched, and found the MIND dataset: pre-processed news articles combined with rich behavioral data extracted from visits to Microsoft's news service in the form of clicks on news impressions, which are groups of news articles presented simultaneously. The MIND dataset provided a great platform to try new techniques. Because the whole dataset is so large, and for rapid prototyping purposes, my model was trained on the MIND-small variant.

Instead of using cross-entropy loss to predict "click" or "not clicked", I decided that I wanted to use metric learning techniques: I would create an embedding space for news impressions, and try to make predictions based on the embeddings themselves. This has a number of benefits: you can experiment with a wider variety of loss functions depending on how you want the different classes to behave in the latent space. You could also experiment with mining hard pairs of embeddings, as well as potentially creating a loss function that indicates not only *if* an impression will be clicked on, but *which user* will click on the

impression: since metric learning can scale up to an unbounded number of classes, you could have different users' behaviors as their own independent label.

I also chose a Wide and Deep architecture for my model. I had learned from a 2016 Google paper about the use of Wide and Deep models for recommender systems [1], which are suitable for learning both direct feature correlations, and also latent similarities of inputs: they combine the best of both linear additive models and deep neural networks.

2 Dataset

> The MIND dataset for news recommendation was collected from anonymized behavior logs of Microsoft News website. We randomly sampled 1 million users who had at least 5 news clicks during 6 weeks from October 12 to November 22, 2019. To protect user privacy, each user is de-linked from the production system when securely hashed into an anonymized ID. We collected the news click behaviors of these users in this period, which are formatted into impression logs. We used the impression logs in the last week for test, and the logs in the fifth week for training. For samples in training set, we used the click behaviors in the first four weeks to construct the news click history for user modeling. Among the training data, we used the samples in the last day of the fifth week as validation set. In addition, we release a small version of MIND (MIND-small), by randomly sampling 50,000 users and their behavior logs. Only training and validation sets are contained in the MIND-small dataset.

[2]

The MIND dataset, which stands for MIcrosoft News Dataset, is a freely avaible dataset released by Microsoft for their MIND Competition. MIND contains 160k English news articles and 15 million impression logs generated by 1 million users. The dataset has the following components: the news articles, the behavioral data, and the entity embeddings.

A **news article** had these fields:

- *news_id*, an integer identifier
- *category*, a string for the category of news article. There were 16 categories in total, such as 'news' or 'sports'
- *subcategory*, a string for the subcategory of news article. There were 273 subcategories in total, such as 'newsus' or 'basketball_nba'
- *title*, the headline text
- *abstract*, the text of the abstract
- *title_entities*, a list of the entities present in the article
- *abstract_entities*, a list of the entities present in the abstract

A **behavior** example consists of the following information:

- *impression_id*, the id of the impression
- *user_id*, the id of the user shown the impression

- *time*, the time of the impression. Note that no published date is given for news articles, so the only way to infer a lower bound on a news article's publication data is via the first impression on that news article
- *history*, a list of *news_id*s that this user has previously clicked on
- *impressions*, a list of tuples of the form $(news\_id, was\_clicked)$ where *was_clicked* is True if the user clicked on the article during that impression, or False otherwise. The *clicked* feature was what I was trying to predict.
- *numchoices*, a feature I created myself, which is simply the number of articles presented in the impression

Microsoft has preprocessed the news articles so that every reference to an object available on WikiData was denoted as an entity, to be listed in the entity lists. Then, Microsoft used the TransE method to generate vector embedding representations of the entities. An entity contains the following mineable information, but due to time I was not able to incorporate this information without hurting model performance. The pre-computed entity embeddings themselves were 100 dimension vectors.

I also did some preprocessing of the dataset to collect some features specific to each user:

- $cat\_0, cat\_1, cat\_2, cat\_counts0, cat\_counts1, cat\_counts2$, the top 3 clicked-on categories and number of clicks
- $subcat\_0, subcat\_1, subcat\_2, subcat\_counts0, subcat\_counts1, subcat\_counts2$, the top 3 clicked-on subcategories, and number of clicks
- *clicks*, the total number of clicks the user logged

I 'flattened' the dataset to create my training set, so that each news article (in an impression) was its own row, as opposed to groups of articles being in a single row. I used a LRU cache to speed up some of the repetitive preprocessing steps that were otherwise repeated multiple times per single news article.

3 Wide and Deep Approach Component

Wide and Deep models consist of, as the name suggests, a Wide component and a Deep Component. The Wide component is good at memorizing which combinations of features tend to produce the target output. The Deep Model is able to generalize by matching representations in a latent space that are near to one another. I used the 'pytorch-widedeep' library [3] to create the model.

The output is simply their weighted sum after an activation function:

$$\hat{y} = \sigma(W_{wide}^T[x, \phi(x)] + W_{deep}^T a_{deep}^{l_f} + b)$$

where

- $\hat{y}$ is the (scalar or vector) prediction
- σ is the activation function
- W_{wide} is the weight matrix for the wide model

- x is the input to the wide model, and $\phi(x)$ is the crossed column features; this allows one to AND relevant Wide model features together. Here $[\cdot]$ represents concatenation of the features to a *wider* vector
- W_{deep} is the weight matrix for the deep model
- $a_{deep}^{l_f}$ is the output of the activation function on the final layer of the deep model
- b is a bias term

Both the wide and deep model were configured to both output 128-dimension vectors. Since their results are simply summed together, they are simultaneously trained via back-propagation.

The features I used for the wide features were: $cat0, cat1, cat2, subcat0, subcat1, subcat2, category, subcategory, numchoices$

For my AND-ed columns I used $(cat0, cat1), (cat0, subcat0)$.

For the deep model I used a tabular data-specific deep neural network. The tabular model takes two types of features: categorical features (which are turned into embeddings), and continuous features. The categorical features I used for the deep component were $cat0$, $cat1$, $cat2$, $subcat0$, $subcat1$, $subcat2$. The continuous features were $clicks, cat\_counts0, cat\_counts1, cat\_counts2$, $subcat\_counts0, subcat\_counts1, subcat\_counts2, numchoices$.

The tabular neural network was a multilayer perceptron with 3 layers of 128 hidden units with ReLU activations, with a 10% chance of dropout.

4 Training procedure

To perform metric learning, I used the library 'pytorch-metric-learning' [4] as it provided helpful tools to do metric learning. I defined a custom loss function for pytorch-widedeep to take advantage of the design of pytorch-metric-learning. In particular, I created a class called 'MetricLoss', which allows you to pass a loss function and a hard-pair miner from pytorch-metric-learning.

The trainer loops over batches of $(news, label)$, where $label$ was the True if the article was clicked during that impression and False otherwise. Internally, the WideDeep model was outputting embeddings. I used the MultiSimilarityMetric miner from pytorch-metric-learning to find examples that were difficult for the model, and then found the triplet loss of the batch. For my triplet loss, the clicked articles were labeled positive, and the unclicked articles were labeled negative.

This is a heavily class-imbalanced problem: less than 5% of articles were clicked on. I made sure to give the clicked articles 22x the weight of unclicked articles in the loss, to make the model less prone to ignoring the clicked articles.

Triplet loss has an $O(n^3)$ runtime complexity in terms of the batch size n, so the training batch size was limited to 78 examples per batch. In total I did 5 epochs of training.

5 Inference

To generate predictions, I used Spotify's annoy library [5] to rapidly produce approximate nearest neighbors. I built up an index from the training embeddings, and then made a query

for each of the test embeddings. This query returned the index of the nearest neighbor in the training set, and also the distance to that neighbor.

The specified prediction format for the competition was that, for each impression in the test set, you produce a ranked ordering for each news article. To produce rankings, I ranked the 'clicked' predictions as first, and the 'not clicked' as last. Within each class of label, I ordered the predictions by distance to the neighbor in the training set.

6 Analysis

In the end, the model was 93% accurate. However, the vast majority of examples were 'unclicked' - since the target class was so imbalanced, this is not necessarily an indicator of good performance.

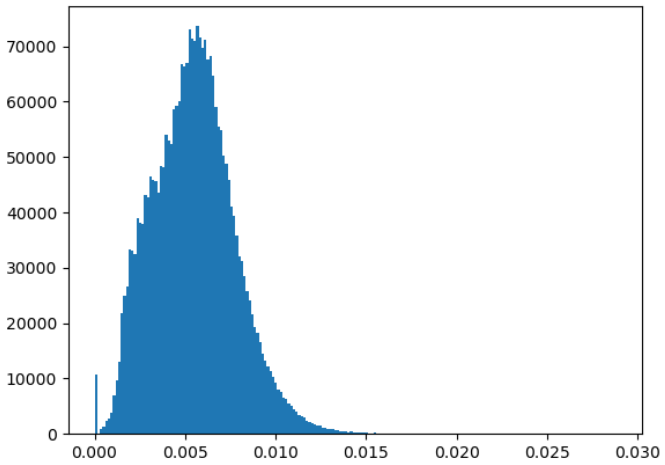
report.png

	precision	recall	f1-score	support
False	0.96	0.96	0.96	2629615
True	0.10	0.10	0.10	111383
accuracy			0.93	2740998
macro avg	0.53	0.53	0.53	2740998
weighted avg	0.93	0.93	0.93	2740998

Figure 1: sklearn classification report of the predicted labels

In fact, the precision and recall were both quite poor for positive training examples. One reason for this may have been due to the sparse representation of a user's history that I used (category and count) compared to the full information available.

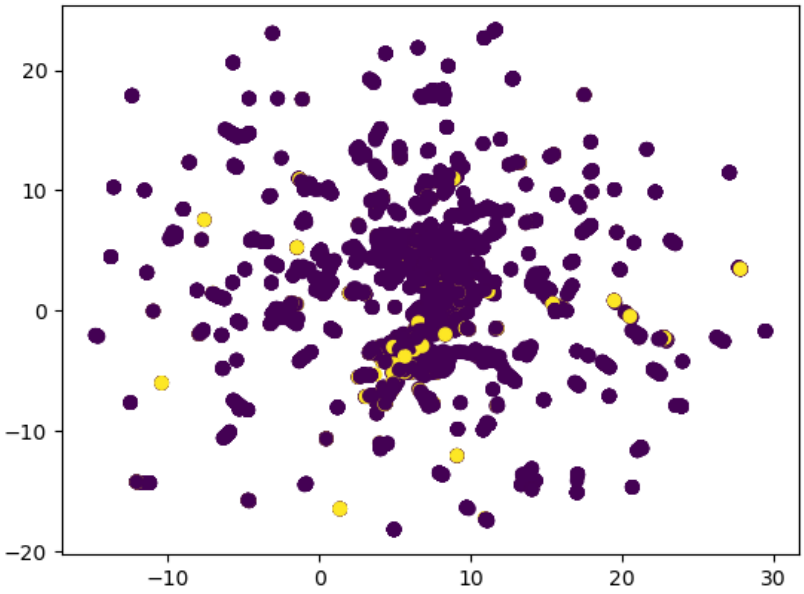
Additionally, I produced a histogram of the distances to the nearest neighbors:



histogram.png

Figure 2: Histogram of distances to test embedding's nearest neighbor in training set

Here is also a visualization of the clusters of embeddings after dimension reduction via UMAP, colored by actual class. Due to RAM limitations this is a visualization of only the first 10000 embeddings of the test set.



viz.png

Figure 3: Visualization of dimensionality reduction performed on 10000 test set embeddings. Purple is 'no click' and yellow is 'clicked'

Although there is some minor noticeable clustering, it does not look particularly non-spurious; although there is a little bit of agglomeration near the center of the image.

7 Conclusion and Possible Enhancements

I produced a Wide and Deep model to try to predict click behavior on news articles. I trained it according to triplet loss, and I used a 'miner' to find the hardest pairs to use with the triplet loss objective. Because of the relative sparsity of positive examples, it was hard for the model to learn to precisely predict when an article was likely to be clicked by the user.

It would be interesting to use metric learning to the fullest extent by having hierarchical labels: you can express labels like (positive class → clicked on by user x). That way, if user x is present in the training set, we can use the cluster for user x if they occur in the test set also. But if we encounter user x in the training set but they were never in the test set, then we can fall back to the parent cluster, which would be the positive set.

Additionally, it would be interesting to play around more with how to generate the deep component of the WideDeep model, such as by using Transformer embeddings for the

text component directly, or instead of using a Tabular MLP, using a new kind of Tabular neural network that is itself a Transformer model: the TabTransformer model that is already present in the pytorch-widedeep package.

References

[1] Google Research Blog, Wide Deep Learning: Better Together with TensorFlow https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html 2016

[2] MIND Dataset, Microsoft https://msnews.github.io/ 2020

[3] PyTorch WideDeep https://github.com/jrzaurin/pytorch-widedeep

[4] PyTorch Metric Learning https://github.com/KevinMusgrave/pytorch-metric-learning

[5] Spotify Annoy https://github.com/spotify/annoy