EE6550 Final Project Report

107061212 劉亦傑、107061218 謝霖泳

107061234 張博閔、107062217 鍾凱恩

1. Title

Using Random Forest, LSTM, Transformer and CatBoost to Perform Behavior Classification of Exposition Visitors

2. Data Preprocessing

Intuitively, the route is only related to the visiting order. Therefore, we only consider the order of locations visited for a certain person (mac_hash) and ignore the absolute visit time. However, the number of locations that a person visited is not identical in the dataset, which varies from 1 to 14. Therefore, the preprocess phase of the dataset is needed. We will attempt the following two methods.

- With grouping method

  Distribute the people (mac_hash) into several groups by the number of locations that he/she visited. After analysis, we group the people who visited 1~2 locations together, people who visited 3~7 locations for the second group, and people who visited 8~14 locations for the last group. To maintain the length of samples in each group, we will perform padding up to the limits of the group. For instance, for the group with people who visited 3~7 locations, we will pad all the samples in this group to 7 locations.

  Since the train data in the first group all belong to label 0, we don't train a model but classify the data with 1~2 locations to label 0 directly. For the other two groups, we will train a model and the corresponding model will be applied in the inference stage according to the length of the test sample.

- Without grouping method

  Treat all the samples as a same group, and padding all the samples to 14 locations is needed.

3. Methods and Architectures

A. Random Forest

Random Forest is a supervised machine learning algorithm used widely to solve Classification and Regression problems. In this algorithm, people construct decision trees on different features and make a majority vote of them to establish the outcomes based on the predictions of the decision trees. Finally, the predicted results can be used for Classification or getting average value in Regression problems.

One of the most important features of Random Forest is that it is able to deal with continuous variables in a data set, and this is the reason why Random Forest can apply in regression problems. However, in general, this algorithm performs better for classification problems.

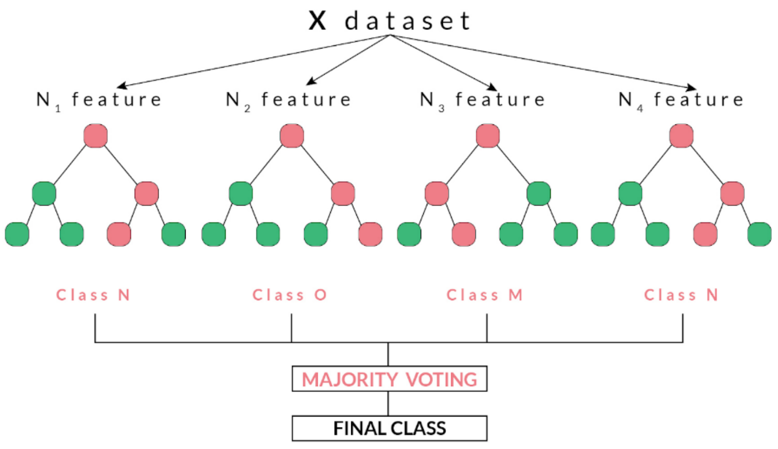


Fig 1. Random decision forests

Algorithm 1: Pseudo code for the random forest algorithm

```
To generate $c$ classifiers:
for $i = 1$ to $c$ do
    Randomly sample the training data $D$ with replacement to produce $D_i$
    Create a root node, $N_i$ containing $D_i$
    Call BuildTree( $N_i$ )
end for

BuildTree(N):
if $N$ contains instances of only one class then
    return
else
    Randomly select x% of the possible splitting features in $N$
    Select the feature $F$ with the highest information gain to split on
    Create f child nodes of $N$ , $N_1$ ,..., $N_f$ , where $F$ has $f$ possible values ( $F_1$ , … , $F_f$ )
    for $i = 1$ to $f$ do
        Set the contents of $N_i$ to $D_i$ , where $D_i$ is all instances in $N$ that match
        $F_i$
        Call BuildTree( $N_i$ )
    end for
end if
```

Fig 2. Pseudo code for Random forests algorithm

B. LSTM / RNN

A feed-forward neural network allows information to flow only in the forward direction, from the input nodes, through the hidden layers, and to the output nodes. There are no cycles or loops in the network.

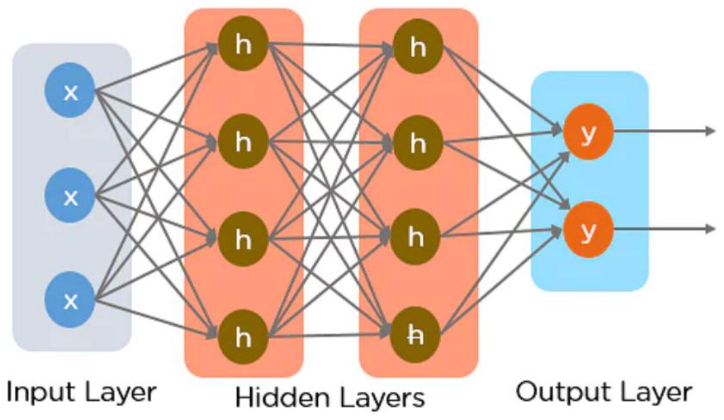


Fig 3. Feed-forward Neural Network

RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.

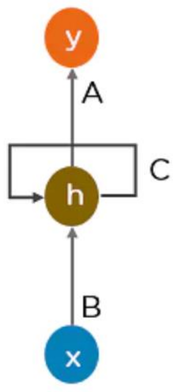


Fig 4. Simple Recurrent Neural Network

The main difference between RNN and LSTM is in terms of which one maintains information in the memory for a long period of time. Here LSTM has an advantage over RNN as LSTM can handle the information in memory for a long period of time as compared to RNN.

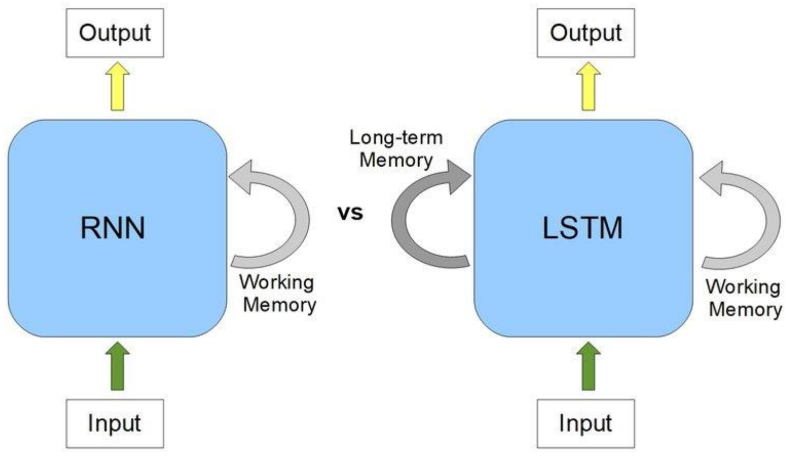


Fig 5. The main difference between RNN and LSTM

C. CatBoost

CatBoost is based on gradient boosted decision trees. During training, a set of decision trees is built consecutively. Each successive tree is built with reduced loss compared to the previous trees. The number of trees is controlled by the starting parameters. To prevent overfitting, we can use the overfitting detector. When it is triggered, trees stop being built.

Building stages for a single tree:

- Preliminary calculation of splits.
- (Optional) Transforming categorical features to numerical features.
- (Optional) Transforming text features to numerical features.
- Choosing the tree structure. This stage is affected by the set Bootstrap options.
- Calculating values in leaves.

D. Transformer – BERT

The main difference between Transformer and LSTMs / RNNs is in terms of the former processes the entire sequential input (such as natural language) all at once by using self-attention mechanisms. Moreover, positional encoding can be included to help the model recognize the sequence order. For a typical seq2seq transformer, it contains encoder and decoder parts. But in our task, we only need the encoder part to extract features for classification instead of generating another sequence with decoder.

The classical transformer encoder architecture called BERT is used in our project. The inputs of this model will be processed to be feature embedding, which contains input sequence, token ID, position ID, and attention masks. In our task, token ID is fixed since we have only one sequence in our input, position ID is the sequence order of the input sequence and we set attention masks as “unmask” for the whole sequence as default.

Note that since we have relatively short input sequence length and small training set compared with common NLP tasks, it’s inappropriate to apply such a large pretrained model directly. Therefore, we propose our own configuration for model architecture to adapt to our data and train it from scratch. And we choose the pooler output (output corresponds to the first input token) as our encoder output to represent the “semantic” of the input sequence for the further fully connected layer and softmax layer to obtain classification probability.

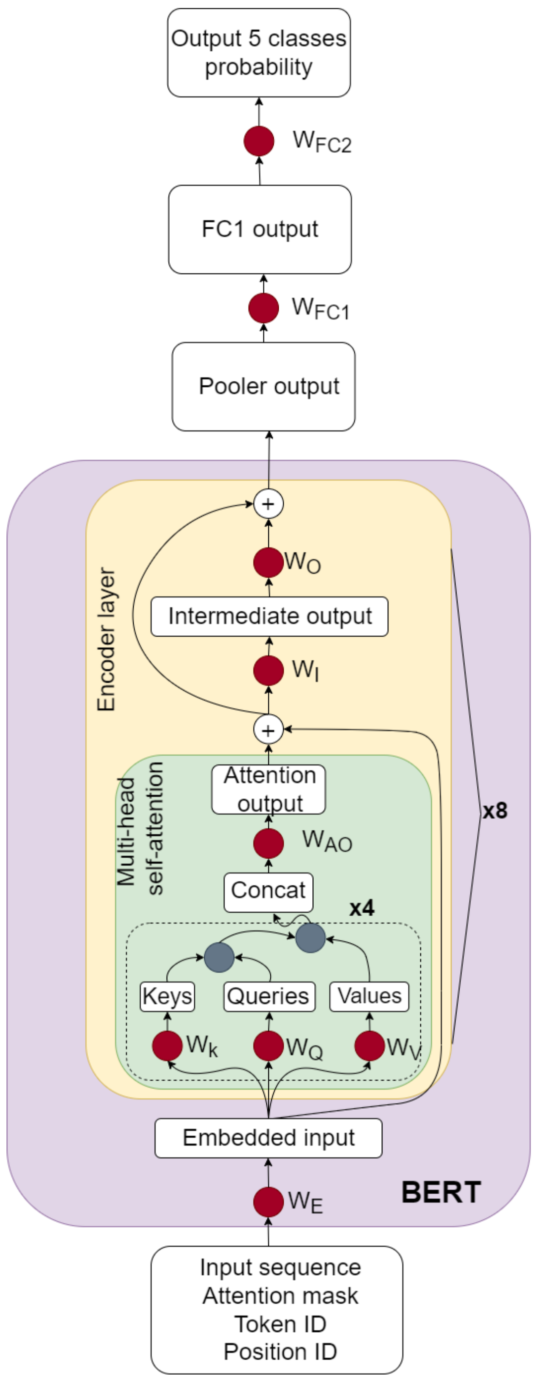
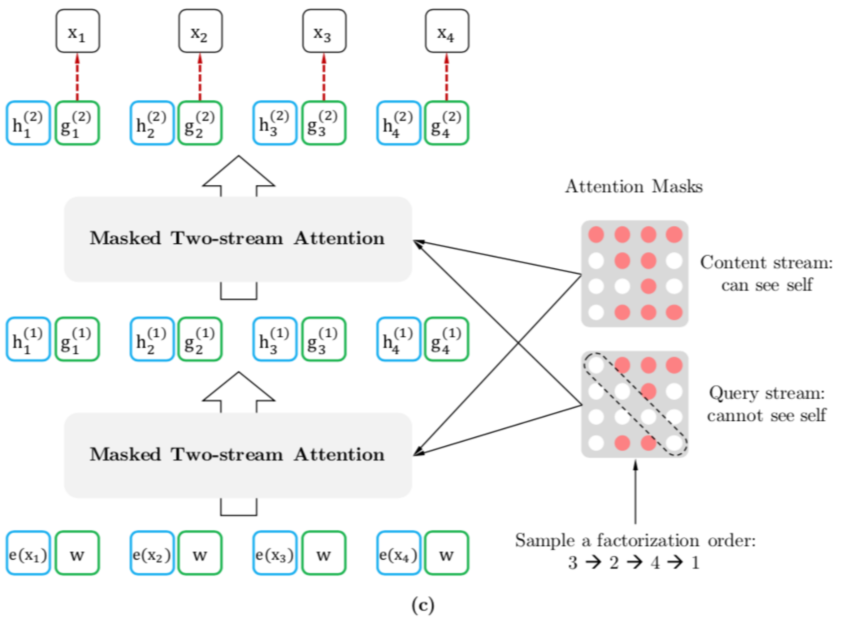


Fig 6. Main architecture (Transformer BERT, encoder part)

E. Transformer – XLNet

Since we tried BERT as our model and achieved an excellent result, we wanted to try the later architecture proposed by Google, XLNet, a.k.a. state-of-the-art NLP model in 2019.

It features the properties of auto regressive and auto encoding, which is similar to BERT. However, XLNet has another feature, two-stream self-attention. This feature enables the model to process a longer input sequence rather than a limited length. With the two-stream self-attention mechanism, the model can understand the sequence better.



Masked Two-stream Attention
Masked Two-stream Attention
Attention Masks
Content stream: can see self
Query stream: cannot see self
Sample a factorization order: 3 → 2 → 4 → 1
(c)

Fig 7. Self-attention in XLNet

4. Experiment Result

A. Evaluation & general settings

	Aidea score	Validation acc.
LSTM	0.1498253	0.9598
RNN	0.1368455	0.9411
Random Forest	0.1781107	0.9534
Catboost (w/ grouping)	0.1276041	0.9611
Catboost	0.1115650	0.9625
BERT (w/ grouping)	0.0991397	0.9842
BERT	**0.0468931**	**0.9871**
BERT+XLNet *	**0.0439630**	-
* When the highest probability of the 5 classes is under a threshold, and the inference results are different then we will change the probabilities into ones predicted by XLNet.		

Table 1. Evaluation result

These results are based on padding 0's in front. For each method we use, the training pipeline is to train the model with validation_split = 0.1 (i.e. 10% of our training data will be applied for validation in the training phase) at first.

Next, we will evaluate the accuracy by ourselves and interpret the training quality. Then, we're going to put the whole dataset to train.

The table already shows our different experiment results. Since the transformer-based method generally performs better than other methods, we will focus on the different settings and discuss the experiment result about the transformer-based models.

For our BERT & XLNet models, the hyperparameters settings are listed as below.

	BERT	XLNet
vocab_size	15	15
hidden_size (d_model)	16	16
num_hidden_layer (n_layer)	8	8
num_attention_heads (n_head)	4	4
intermediate_size (d_inner)	64	64
max_position_embeddings	14	-
optimizer	adam	adam
learning rate	1e-3	1e-3
learning rate decay	1e-5	1e-5
loss	Categorical Crossentropy	Categorical Crossentropy

Table 2. Hyperparameters settings on BERT and XLNet

B. Further analysis about BERT

- without grouping
  - BERT + padding with 0 before original sequence

    We pad each training & test sample with 0 before the original list. Besides, XLNet will be trained by the same analogy here. (val acc: 0.987)
  - BERT + average sequence output

    Since our usage is to perform classification, the original BERT will take the last layer hidden-state of the first token of the sequence as the classification token. But, we think it perhaps is not a good summary of the semantic content of the input. So, we apply some average operations for all the sequence outputs to generate our classification token, which performs classification tasks. (val acc: 0.9784)

- BERT + append [CLS] in front to imitate original BERT paper implementation

  Due to most NLP tasks, we will use [CLS] to represent the semantics. We try to imitate that situation and select [CLS] to be 101 in our experiment. So, our sequence length is increased by 1. (val acc: 0.9871)
- BERT + mask padded sequence

  This setting will adjust the attention_mask input. We will look at the original sequence and mask the position that it's padded 0. (val acc: 0.9756)
- BERT + padding with 0 after original sequence

  We pad each training & test sample with 0 after the original list. (val acc: 0.9770)
- BERT + padding the same value of the first element in original sequence

  We pad each training & test sample with the same value in the first element of the original list. (val acc: 0.9727)

- with grouping
  - BERT + padding with 0 before original sequence

    We pad each training & test sample with 0 before the original list. Note that this uses grouping, so it will be padded up to the limit of each group. (val acc: 0.9842)

5. Discussion

A. Random Forest vs CatBoost (Decision Tree-Based Architecture)

Gradient Boosting algorithms such as CatBoost perform better in general situations. However, if we train a model on a small data set or the data set has few features to learn, Gradient Boosting algorithms may overfit even though their parameters are tuned.

Despite the sharp prediction from Gradient Boosting algorithms, in some cases mentioned above, Random Forest takes advantage of model stability from selecting randomly and outperforms CatBoost.

B. RNN vs Transformer (Neural Network)

RNN is the most classic model to process sequence data. However, compared to the Transformer, the main problem of RNN is that it is difficult to process data parallelly. But, transformer-based can execute parallely with the multi-head property. In terms of accuracy, one of the most significant features of Transformer is that it has self-attention mechanism which enables the model to learn more and it will not only focus on local parts of the sequence. Instead, the model can learn the whole sequence by the effect of enhancing some parts of the input while diminishing other parts. So, with the help of self-attention, the network is able to devote more focus on crucial parts of input sequence.

C. With grouping vs Without grouping

After our experiments, the performance of without grouping is better in both Catboost and BERT. Therefore, we may conclude that treating all data as a single group is a better choice. We guess that there are two main reasons. First, after the partition, the amount of data in each group is much smaller, which may easily cause overfitting. Second, it may be inappropriate to separate the data as it's difficult to decide the boundary and data with different locations visited may all be related to each other.

D. Further discussion about BERT settings

After our experiments, we conclude that most of the adjustments have no great impact on the performance of the validation set except for padding the same value of the first element in the original sequence. We guess that padding the same value implies the person stays in that location, which may mislead the model.

E. Our ultimate result (decent models)

Our ultimate result is estimated by BERT together with XLNet. The reason this approach attains the best result is the model holds different confidence for each sample. That is, when our BERT model is not so confident of the predicted result, then we change the predicted result to the prediction of other models.

6. Reference

[1] Leo Breiman. Random Forests. https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

[2] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, Andrey Gulin. CatBoost: unbiased boosting with categorical features, 2017.

[3] Sepp Hochreiter, Jürgen Schmidhuber. Long Short-term Memory. In *Neural Computation*, 1997.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need. In *Neural Information Processing Systems (NIPS)*, 2017.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Google Research*, 2018.

[6] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Google AI Brain Team & Carnegie Mellon University*, 2019.