On the "ConvXGB: A new deep learning model for classification problems based on CNN and XGBoost"

by

Ramin Ziaei

Discussion 2

Under the Supervision of Dr. Cheng

Department of Mathematics

University of Nebraska - Omaha

Omaha, Nebraska

October 29th, 2020

Contents

List of Figures

List of Tables

1 Introduction

Classification problems are a very important part of the supervised learning, and over the course of the past few decades, many researchers have come up with different algorithms to classify an object. These algorithms can range from the simplest ones like logistic regression to more complicated ones like Random Forest (RF), Support Vector Machine (SVM) and Extreme Gradient Boosting (XGBoost). Among all of the mentioned algorithms, XGBoost has gained much reputation due to its speed and accuracy, especially in classification problems. In recent years, there has been a massive work and research on Artificial Neural Networks (ANN) that can be used in a variety of fields from regression and classification to image and text recognition. Convolutional Neural Network (CNN) is a part of the neural network family that is mostly used for image recognition problems. A common step in all machine learning algorithms is to find the most important features that will contribute the most towards the final classification. This is something that is usually done manually by people who run algorithms like RF, SVM and XGBoost, to name but three. On the other hand, one of the beauties of the CNN is its automatic feature learning capability which is a huge advantage over all other algorithms. Since the capabilities of a single model may not be sufficient to be used for different problems, and each model has its own advantages and disadvantages, it can be a good idea to combine models and take advantage of their strengths. This paper introduces a new deep learning model called ConvXGB which is a combination of CNN and XGBoost. ConvXGB consists of a net of several stacked convolutional layers and with a XGBoost at the end of the network to do the classification part. The main difference beetween this novel model and the conventional CNN model is that ConvXGB does not have either a pooling layer or a fully connected layer which was done to add simplicity to the model. ConvXGB can be used for both image and general classification problems and according to the literature has shown promising results. Figure 1 shows the architecture of the ConvXGB model. Figure 2 displays how the output from the convolutional layers are reshaped and fed into the XGBoost part of the model.

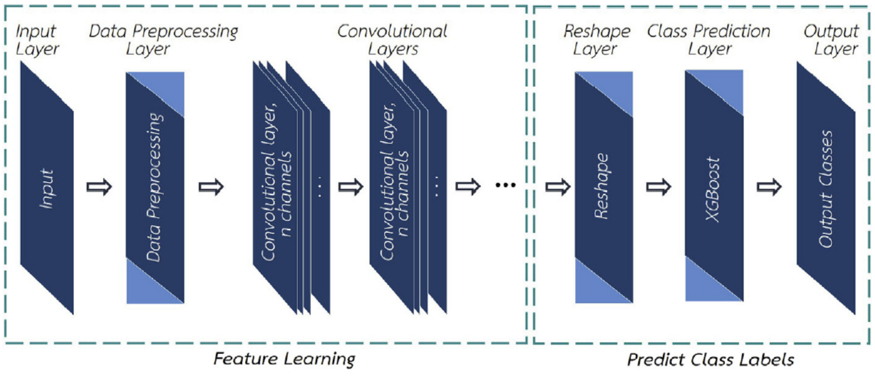


Figure 1: Arcitecture of the ConvXGB model [1]

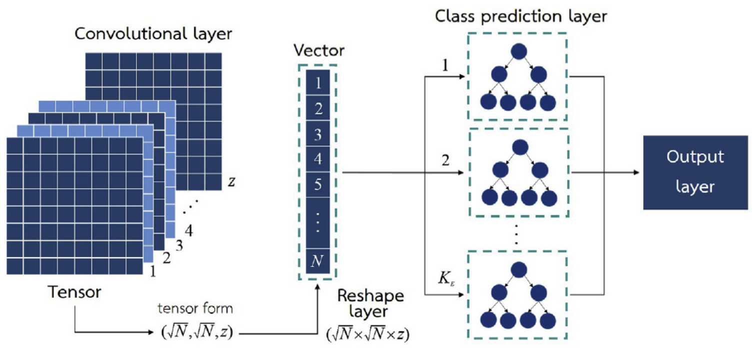


Figure 2: The process of operation in part of predict the class labels consists of reshape layer, class prediction layer and output layer [1]

A vital part of the ConvXGB model is the preprocessing part where the input data is reshaped to 2-D square matrix. If the number of features for each observation does not match a square number, zero elements will be added to the features vector until a square number is reached. Then, it will be reshaped to square matrix. This process which is called zero-padding is shown in figure 3.

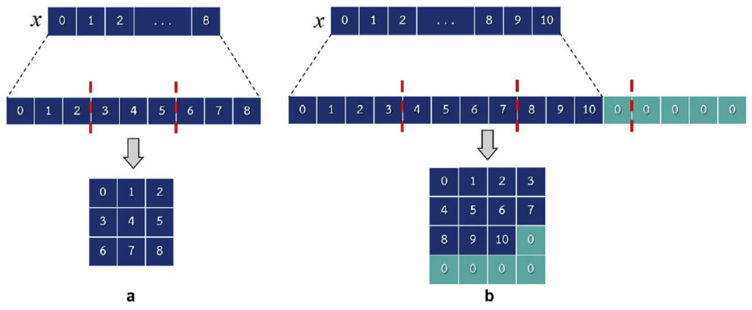


Figure 3: Example of converting data to the standard criterion: (a) Describes of a 'square' feature vector which can be simply copied whereas, in (b), zeroes are added so that the feature vector is square [1]

2 Datasets

The data is the one I will be using for my final project. It contains information of 2 different hotels, one being a resort hotel (H1) with 40,060 observations and the other one being a city hotel (H2) with 79,330 [2]. Both data sets share the same structure with 30 predictors, and the goal is to predict if a specific observation would cancel their booking or not. Data cleaning part has been already explained in my first draft, so it will not be covered here again, and I will just use the result of it. The cleaned data has 15 factor and 15 numeric columns. 80% of the data (95511 observations) was used for training, and the remaining 20% (23878 observations) was used as the test set.

3 XGBoost

Since half the features (columns) are factors, one-hot encoding needs to done before sending the data into the XGBoost algorithm. Considering different levels of those 15 columns, after one-hot encoding, both train and test sets will have 171 columns. A very small grid search of 54 combinations of 4 parameters (eta, max_depth, col_sample and sub_sample) was conducted to find the optimum value of their combination. The result is shown in table 1. For this model, 10-told cross validation was used to assess the performance of the model. Figure 4 displays the importance matrix of the model (top 10), and it shows non-refundable deposit feature as the most important feature in this classification problem.

Table 1: XGBoost tunned parameters

Parameter	Value
eta	0.1
max_depth	3.0
col_sample	0.8
sub_sample	0.8

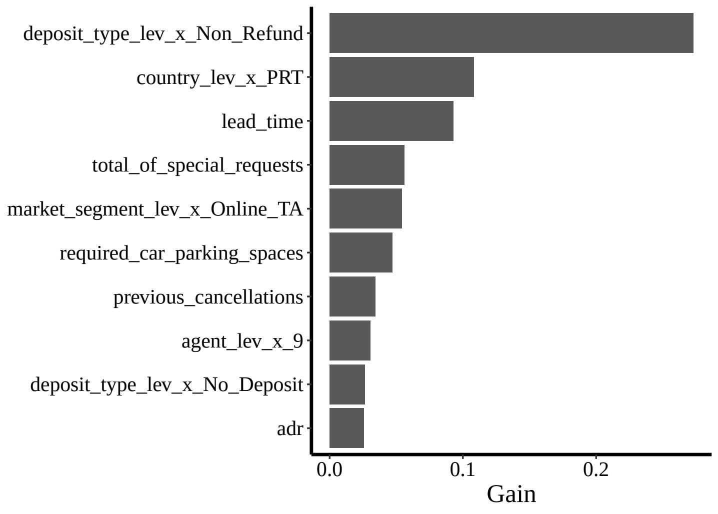


Figure 4: Importance matrix

Then, this model was evaluated on the test set, and the final accuracy was 88.04%.

4 Convolutional Neural Network

For convolutional neural network, we do not need to do zero-padding. In fact, we will use the exact number of columns that are generated after one-hot encoding which is 171. So, the matrix is reshaped to a 4-D array of size train-test $\times$ 19 $\times$ 9 $\times$ 1 since this is the way a CNN works ($19 \times 9 = 171$). Then, this 4-D array was fed into the network. For this model, three convolutional layers were used each of which had 32, 64 and 128 filters successively with the kernel size of 3 $\times$ 3. After the first 2 convolutional layers, there is a pooling layer with a 2 $\times$ 2 kernel size. The Rectified Linear Unit (ReLU) was used as the activation function for all layers, the "padding" option was set to "same" in order to keep the size of the matrix constant through all the convolutional layers. Finally, there is a flattened layer which is connected to a dense layer with 256 neurons. This dense layer is connected to a dropout layer with 25% rate to prevent overfitting. At the end, there is a dense layer with only 1 neuron and sigmoid activation function. This model and network are shown in figures 5 and 6. In a CNN model, there can be thousands or even millions of parameters that the network learns throughout its process. But, where do these parameters come from? In a CCN model, all parameters come from filters and kernels. For example, let us consider the network in figure 6. The last column of this figure shows the number of parameters in each layer. For the 2^{nd} convolutional layer, we can see the number of parameters is 18496. This number comes from the filters in the first and second layers, the kernel size (here 3 $\times$ 3) and the bias for the second layer(for each filter there is a single bias). Therefore we have $(32 \times 64 \times 9) + 64 = 18496$ parameters. The accuracy of this model was 87.72%.

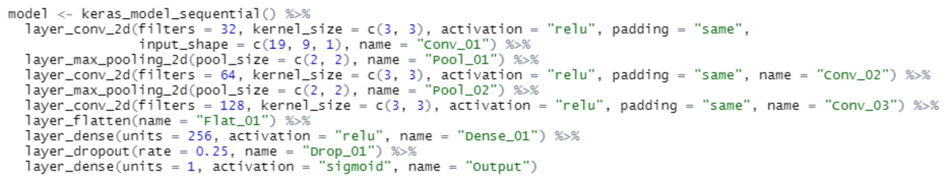
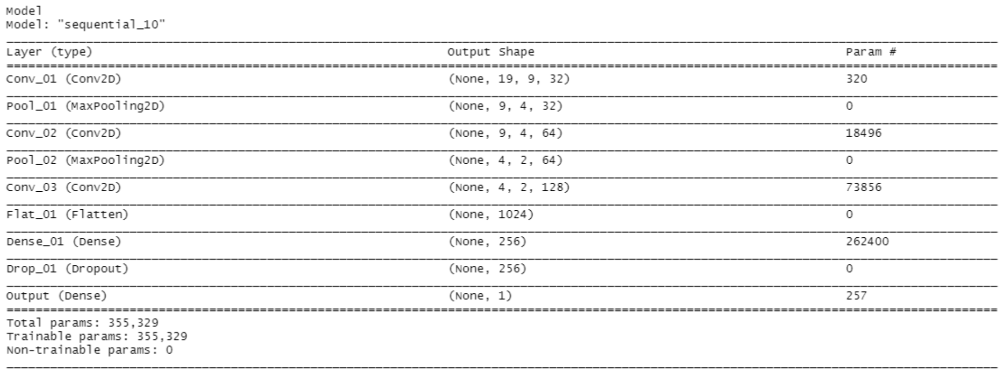

```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu", padding = "same",
                input_shape = c(19, 9, 1), name = "Conv_01") %>%
  layer_max_pooling_2d(pool_size = c(2, 2), name = "Pool_01") %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu", padding = "same", name = "Conv_02") %>%
  layer_max_pooling_2d(pool_size = c(2, 2), name = "Pool_02") %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu", padding = "same", name = "Conv_03") %>%
  layer_flatten(name = "Flat_01") %>%
  layer_dense(units = 256, activation = "relu", name = "Dense_01") %>%
  layer_dropout(rate = 0.25, name = "Drop_01") %>%
  layer_dense(units = 1, activation = "sigmoid", name = "Output")
```

Figure 5: Model generated in keras for CNN

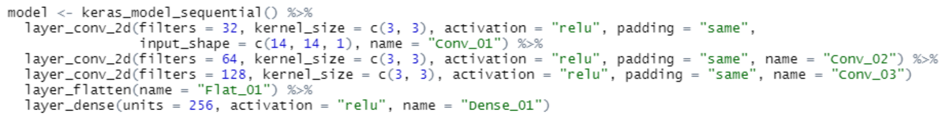


```
Model
Model: "sequential_10"
________________________________________________________________________________________________________________________________
Layer (type)                                   Output Shape                                Param #
================================================================================================================================
Conv_01 (Conv2D)                               (None, 19, 9, 32)                           320
________________________________________________________________________________________________________________________________
Pool_01 (MaxPooling2D)                         (None, 9, 4, 32)                            0
________________________________________________________________________________________________________________________________
Conv_02 (Conv2D)                               (None, 9, 4, 64)                            18496
________________________________________________________________________________________________________________________________
Pool_02 (MaxPooling2D)                         (None, 4, 2, 64)                            0
________________________________________________________________________________________________________________________________
Conv_03 (Conv2D)                               (None, 4, 2, 128)                           73856
________________________________________________________________________________________________________________________________
Flat_01 (Flatten)                              (None, 1024)                                0
________________________________________________________________________________________________________________________________
Dense_01 (Dense)                               (None, 256)                                 262400
________________________________________________________________________________________________________________________________
Drop_01 (Dropout)                              (None, 256)                                 0
________________________________________________________________________________________________________________________________
Output (Dense)                                 (None, 1)                                   257
================================================================================================================================
Total params: 355,329
Trainable params: 355,329
Non-trainable params: 0
________________________________________________________________________________________________________________________________
```

Figure 6: CCN network

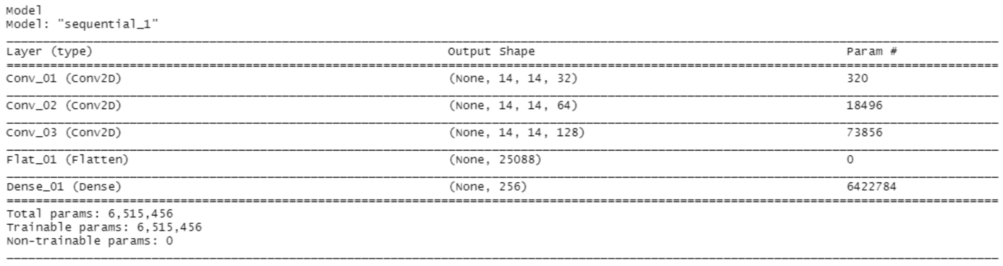
5 ConvXGB

As mentioned in the XGBoost section, after one-hot encoding, the number of columns will be 171. We know the first step in this approach is zero-padding if necessary. Since 171 is not a square number, 25 zero columns were added to the matrix to raise the number of columns to 196. Then, the 2-D matrix was reshaped to a 4-D array of size train-test $\times$ 14 $\times$ 14 $\times$ 1. Three convolutional layers were used each of which had 32, 64 and 128 filters successively with the kernel size of 3 $\times$ 3. The ReLU was used as the activation function for all layers, the "padding" option was set to "same" in order to keep the size of the matrix constant through all the layers (as the paper suggests). However, in this case, since the number of columns would be 25088, it had to be reduced so the system could handle it. Then, the last layer was flattened and connected to a dense layer with 256 neurons. The information of this layer was extracted and fed into the XGBoost model (a model with 256 columns). The same parameters as the pure XGBoost model was used here in order to be coherent. Also, a 10-fold cross validation was used to assess the performance of the model. The accuracy of this hybrid model was 88.26%. Figures 7 and 8 show the model as well as the network.



```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu", padding = "same",
                input_shape = c(14, 14, 1), name = "Conv_01") %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu", padding = "same", name = "Conv_02") %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu", padding = "same", name = "Conv_03")
  layer_flatten(name = "Flat_01") %>%
  layer_dense(units = 256, activation = "relu", name = "Dense_01")
```

Figure 7: CNN model generated in keras for ConvXGB



```
Model
Model: "sequential_1"
________________________________________________________________________________________________________________________________
Layer (type)                                   Output Shape                                Param #
================================================================================================================================
Conv_01 (Conv2D)                               (None, 14, 14, 32)                          320
________________________________________________________________________________________________________________________________
Conv_02 (Conv2D)                               (None, 14, 14, 64)                          18496
________________________________________________________________________________________________________________________________
Conv_03 (Conv2D)                               (None, 14, 14, 128)                         73856
________________________________________________________________________________________________________________________________
Flat_01 (Flatten)                              (None, 25088)                               0
________________________________________________________________________________________________________________________________
Dense_01 (Dense)                               (None, 256)                                 6422784
================================================================================================================================
Total params: 6,515,456
Trainable params: 6,515,456
Non-trainable params: 0
________________________________________________________________________________________________________________________________
```

Figure 8: CCN network used in ConvXGB

6 Comparison

Table 2 shows the summary of these three models. The first thing to note is that, they all are performing in a close range where the difference between the best and worst model is only 0.54%. Among the three models, the best performance belongs to the ConvXGB model which is slightly better than a pure XGBoost model, and after these two, there is the CNN model.

Table 2: Models accuracy comparison

Model	Accuracy (%)
XGBoost	88.04
CNN	87.72
ConvXGB	88.26

7 Summary

This paper introduces a novel approach to address classification problems which is called ConvXGB. This method is a combination of CNN and XGBoost to take advantage of the strengths of both and can be applied to both image and non-image classification problems. Something unique that this approach has is that it does not include maximum pooling or fully connected layers. Also, its the pre-processing part makes sure that every input is a square matrix. However, as we saw in section 4, the matrix does not have to be a square matrix, and as long as it can be divided into two integers, the matrix can be shaped. The reason paper insists on a square matrix is to hold a standard shape for all different data types. Also, as shown in section 5, unlike what the paper suggests, fully connected and maximum pooling layers can actually be used. In fact, sometimes it is necessary to use them since the size of the final matrix might be too big for the computer to handle. CNN takes into account spatial structure of the data, and that is why it works perfectly on image data. However, it also can be used for non-image data like a normal data frame. The reason it will work is that there might be some correlation among different columns, and a kernel might capture that correlation. I, personally, think CNN should be cautiously used for non-image data (or generally for data sets that are not spatially sensitive). The reason is that if we swap some of the columns (e.g. the first column and the last column), the result should change since they will be moved to different parts of the next convolutional layer. Having said that, other types of machine learning algorithms (e.g. decision trees) do not face this problem since they are not sensitive to the location of each column, and if we swap columns, they will always give back the same result. It should also be noted that filters in a CNN are something that a network learns by going through different convolutional layers where they become more and more complicated. Finally, since XGBoost was introduced in 2014, it has won many competitions and is very popular due to its accuracy and speed. One of the advantages that Random Forest (RF) has over XGBoost is that it does not need data pre-processing like one-hot encoding that XGBoost does. So, it is possible to use other classifiers as the last layer of ConvXGB instead of XGBoost, but their performance should be evaluated. All in all, based on the results in section 6, it seems that if CNN is not that good for prediction, XGBoost will not boost the result a lot.

References

1. Thongsuwan, S., Jaiyen, S., Padcharoen, A., & Agarwal, P. (2020). ConvXGB: A new deep learning model for classification problems based on CNN and XGBoost. Nuclear Engineering and Technology.
2. Antonio, N., de Almeida, A., & Nunes, L. (2019). Hotel booking demand datasets. Data in brief, 22, 41-49.