Report

Preprocess:

I started with filtering the rows consist “cost” values that are less than “cpc-bid” values due to the requirements. Then, I dropped the “device” column because it consists only one categorial value, which will not help us and give some information. The “ad_network_type_2” column was separated to two one-hot columns by one-hot encoding method. The “date” column was converted to timestamp presentation.

I choose the numeric columns to be the set consist: 'date', 'quality score', 'clicks', 'position','gdp', 'cpc_bid', 'cost', 'Google search', 'Search partners', and normalized them with z-score function, because I want the values to be negative and positive and normalizing the data can improve the learning process of the network. It’s important to normalize it with the predicted column (“cost”) because the features imply on that column.

```
df = df.drop(['device'],axis=1)
one_hot = pd.get_dummies(df['ad_network_type_2'])
df = df.drop('ad_network_type_2',axis = 1)
df = df.join(one_hot)
```

From here, I stated to handle with the categorial features. There were two types of categorial features: one is what I called, “free text”, and the other one is “categorial text”. The columns “description” and “headline” are labeled as “free text”, and the columns “keyword” and “location” are labeled as “categorial text”.

Before I will explain the workflow, I considered during my work to use NLP approaches, such as Word2Vec to embedding all the texts in the dataset, where each sentence will be converting to some vector with R^{d*l} dimension (by sum of the word2vec vectors of each one of his words). But I saw that each text, in each cell, isn’t imply on the other sentences in the other cells, so it can’t be a good training for this method, especially where for example there are only 163 unique description sentences. So, I got off this idea.

Those labels lead us to handle each one of them separately with a tailored approach to the problem- The “categorial text” were first mapped to some dictionary (for each one of them) that the key represent the word/sentence and their values was their ordinary index. Why? because when I check it, the “location” column only consists 61 unique values, and “description”, which is surprisingly mapped to indexes, consist only 163 unique sentences! so I decided to hold for each one of them a matrix which will be consider as a embedding map with dimension of <len(unique values of feature),5>. I save for each matrix an extra row (vector) for unknown words as “UNK”.

```
descriptions = {w:i for i,w in enumerate(df["description"].unique())}
for text,id in descriptions.items():
    df["description"].replace(text,id,inplace=True)
```

The “free text” I first counted the unique values for each column. For “headline” column I found that it consists more than 1000 values. Same “keyword” column, so for each one of them I counted the words appears in their sentences (i.e. in headline there were between 100+ around words) and for each word counted it times it appears. Then, I filtered all the word that appears less than 4 and stay with 96 words of “headline” and 138 words for ‘keyword”. Each one of the words (for each column) mapped to some ordinary index. Why? now I have two embedding matrices, each one for each column, i.e. I have an embedding matrix with dimension of R^{96*5} for “headline” column, where each entry is index represent a word in “headline” dictionary words. The idea is when the network will train, the whole sentence will be decomposing to embedding vectors, and I will average them to one embedding vector which will represent the sentence (i.e. sentence in “headline”).

```
from collections import Counter

keyword_words = Counter()
for __,s in enumerate(df["keyword"].unique()):
    s = s.split(" ")
    keyword_words.update(s)
keyword_words = dict(filter(lambda x: x[1] > 3, keyword_words.items()))
w2i_keyword = {w:i for i,w in enumerate(keyword_words)}
```

The neural network:

The network architecture is MLP consists two hidden layers with 20 and 10 neurons respectively , dropout layer with 0.05 probability as hyperparameter in the first hidden layer, ReLU function as the activation function all over the network, and linear function as the output function (I have tried also tanh function but it wasn't gave me better results). Also, the network consists four embedding matrices, each one for the free/categorial text features.

Another hyperparameters were chosen, such as epochs (10), learning rate for SGD optimizer (0.005).

The loss function I choose for this kind of problem, is the Mean Square Error.

The data was shuffled every epoch, even the data consists time series order, but the normalize step I did before need to fix this, and it gave me better results.

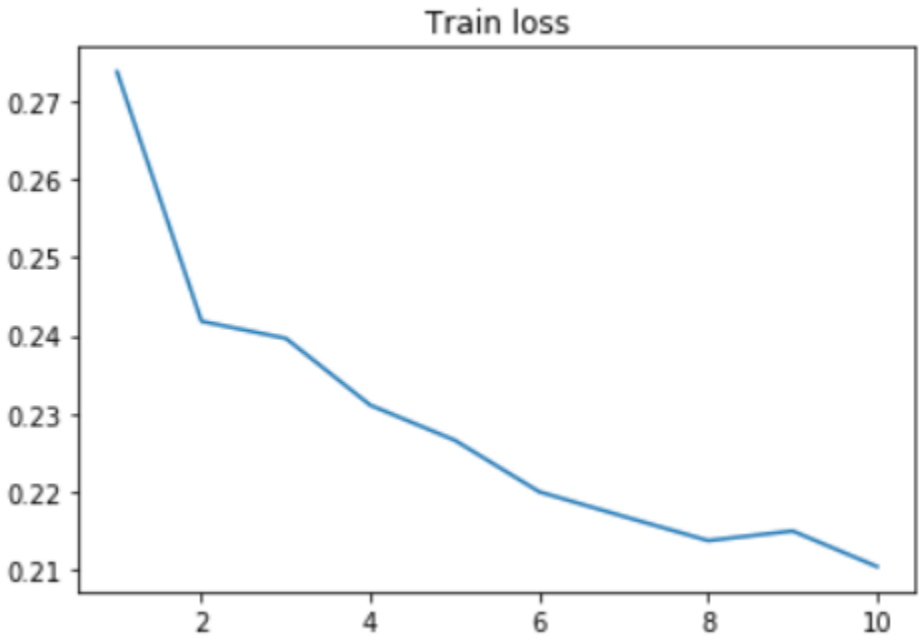
```
self.embedding1 = nn.Embedding(DESCP_SIZE+1,5) #163 + 1
self.embedding2 = nn.Embedding(KEY_SIZE+1,5) #138 + 1
self.embedding3 = nn.Embedding(HEAD_SIZE+1,5) #96 + 1
self.embedding4 = nn.Embedding(LOC_SIZE+1,5) #61 + 1
self.fc1 = nn.Linear(28,20)
self.drop1 = nn.Dropout2d(0.05)
self.relu1 = nn.ReLU()
self.fc2 = nn.Linear(20,10)
self.relu2 = nn.ReLU()
self.out = nn.Linear(10,1)
```

The output of the network, during the prediction phase, was multiplied by the standard derivate and added to the mean of the "cost" column, as the inverse action of "z-score" normalization we did during the preprocessing phase:

$$predicted_{price} = predicted_{price} * \ std_{price} + mean_{price}$$

Results:

I checked the train loss accuracy of the model, and as you can see the loss range between 0.28-0.20 which consider low, and the model trained very good.



Conclusions:

I think about some possible problems that have occurred during the training and before it, for example, one of them is the normalization processing, when I normalized certain columns, and possibly this normalization didn't work well.

I have tried to play with the hyperparameters, and I haven't achieved any significant improvement for my model.