Eluvio Challenge

Define the problem

The problem I studied is to predict 'up_votes' based on 'date_created', 'author' and 'title' columns.

One difficulty was, 'up_votes' ranges from 0 to 21253 and the distribution is pretty sparse for large up_votes values. To deal with this outlier issue, I defined one classification problem and one regression problem, and trained them separately.

a) The binary classification problem

Define a binary class 'category' based on: 'category' 0 if 'up_votes' <= 5 else 'category' 1. There is no specific reason of choosing threshold 'up_votes' = 5, but naively because in this way 'category' = 0 and 'category' = 1 have about equal data amounts. This makes it a balanced binary classification problem.

b) The regression problem

For regression problem, one has to deal with 'up_votes' outliers. The way I did is applying transformation 'logvotes' = ln(1 + 'up_votes') and use the 'logvotes' as y_label, where 'logvotes' ranges from 0 to 9.96.

How to deal with large dataset

For EDA and data processing purpose, one has to iterate over the whole '.csv' file by tools like Python generator. I used pd.read_csv with chunksize = 10,000. To do DEA, I created some hashmaps to cache the information I need when iterating over 51 chunks.

For storing trainable np.array's and feeding them into nn model, I referred to the following blog:

https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly

Let me state the brief idea. For each data, give it an ID. In our problem, the data in the i-th row has a natural ID i. When iterating over chunks, save the encoded trainable np.array of data i as a single file '_{i}_.npy' into the hard drive, also create a hashmap 'labels' to cache {i: y_label of data i}.

Create train_list, val_list, test_list, storing ID's for each of training set, validation set and testing set. In our problem, there're in total 509,236 rows in the dataset. I splitted it into 400,000 training data, 100,000 validation data and 9,236 testing data, randomly and fixing random.seed(0).

Next, implement a DataGenerator class, build nn model and use 'fit_generator' in Keras to do training. DataGenerator can shuffle all data, load a batch_size of data from saved '.npy' files and feed them into the model.

The benifits for doing so:

It always only accesses a batch_size of data, so won't cause RAM overload;

Compare to accessing '.csv' file by pd.read_csv with chunksize, saving each data in a single file allows shuffling during training.

Instead of saving each row of the original '.csv' as a single file, only saving encoded trainable np.array reduces hard drive memory cost.

Jobs of each '.ipynb' file

a) 'EDA_data_processing.ipynb'

It does EDA and saves the following files to "D:\eluvio\data_\" for training use:

"modified_embedding_matrix.npy" : word embedding matrix mapping word token to word embedding vector. I loaded pretrained 'glove.6B.50d' for word embedding. There are 13741 missing words in 'title' text. I tried SpellChecker but it works awful. Then I manually corrected the 40 most frequent words and set word embedding of the rest missing words be zero vectors.

*"_pad_seq{i}.npy" for i in range(*509236*)* : size 50 np.array, padded word token for each title text.

*"_feature{i}.npy" for i in range(*509236*)* : size 8 np.array, features from 'date_created' and 'author': ['author_norm_ave_logvotes', 'author_norm_pub_count', 'norm_year', 'month_cosine', 'month_sine', 'weekday_cosine', 'weekday_sine', 'author_ave_category']. To avoid cheating, the author-related features are calculated only for authors in training dataset. For those authors appear in validation or testing dataset but not in training dataset, author-related features are set to be zero.

"logvotes_labels.npy", *"category_labels.npy"* : y_labels

b) 'LSTM_GloVe_feature_regression'

Regression model predicting logvotes.

c) 'LSTM_GloVe_feature_classification'

Classification model predicting category.

Output

Please refer to 'LSTM_GloVe_feature_regression' and 'LSTM_GloVe_feature_classification'.

Those two model files should have explained themselves well.