milestone_2_00951537

June 16, 2024

1 Preamble

See the script below for all package imports

[27]:
```
%run -i scripts/preamble.py
```

> **Note**: Due to submission size limitations, the datasets were **not** included in the repost submission. Instead, I have made them publicly downloadable from S3. Please refer to the data download instructions in the README file.

2 Introduction

The global digital advertising market is worth approximately $602 billion today. Due to the increasing rate of of online participation since the COVID-19 pandemic, this number has been rapidly increasing and is expected to reach $871 billion by the end of 2027 (eMarketer, 2024). Many of the of the major Ad platforms such as Google, Facebook and Amazon operate on a cost-per-user-engagement pricing model, which usually means that advertisers get charged for every time a user clicks on an advertisment. This means that these platforms are incentivized to make sure that the content shown to each user is as relevent as possible in order to maximize the number of clicks in the long term. Attaining accurate Click-Through Rate (CTR) prediction is a necessary first step for Ad personalization, which is why study of CTR prediction methods have been an extremely active part of Machine Learning research over the past through years.

Initially, shallow prediction methods such as XGBoost (Cite), Factorization Machines (Cite) and Field-Aware Factorization Machines (Cite) have been used for CTR prediction. However, these methods have often been shown to be unable to capture the higher order feature interactions in the sparse multy value categorical Ad Marketplace datasets (Cite). Since then, Deep Learning methods have been shown to show superior predictive ability on these datasets. The focus of my reasearch project is therefore to explore the merits of different Deep Learning architechtures for click-through rate prediction

In the following report, I explore the relevant datasets and simulations that I will be using throughout my research project. In the first section, I perform an exploratory data analysis on three widely adopted benchmark CTR prediction datasets; the KDD12 (Aden, 2012), Avazu (Wang and Cukierski, 2014) and Criteo (Tien et al, 2014) datasets. In the second section, I then explore possible ways of simulating the ad marketplace environment in order to test the reinforcement learning framework.

3 Data Analysis and Pre-processing

I begin below by first introducting the three datasets widely used as benchmarks in CTR prediction research.

3.0.1 KDD12

The **KDD12** dataset was first released for the KDD Cup 2012 competition (Cite), with the original task being to predict the number of clicks for a given number of impressions. Each line represents a training instance derived from the session logs for the advertizing marketplace. In the context of this dataset, a "session" refers to an interaction between a user and the search engine, containing the following components; the user, a list of adverts returned by the search engine and shown (impressed) to the user and zero or more adverts clicked on by the user. Each line in the training set includes:

- **Click and Impression counts**: The click counts were the original target variable when the dataset was first released for the competition. As done in (Cite Song and Others), this dataset can be adapted to CTR prediction by simply calculating the CTR for each instance by dividing the Click counts by the Impression counts.
- **Session features**: These include *session depth* (the number of ads impressed in a session) as well as the tokenized query phrase that the user entered into the search engine.
- **User features**: Encoded gender and age group for the user, if known.
- **Ad features**: Display URL, ad ID, advertiser ID and encoded title, description and purchased key words.

[2]:
```
%run -i scripts/data_analysis_and_preprocessing/retrieve_kdd12.py
```

```
Snapshot of KDD12 training data:

   Click  Impression           DisplayURL      AdID  AdvertiserID  Depth  \
0      0           1  12057878999086460853  20157098         27961      1
1      0           1  12057878999086460853  20221208         27961      2
2      0           1  12057878999086460853  20183701         27961      1
3      0           1  12057878999086460853  20183690         27961      1
4      0           1   3029113635936639912  10397010         24973      2

   Position   QueryID KeywordID TitleID DescriptionID   UserID
0         1     75606     15055   12391         13532  1350148
1         1      2977      1278    3054          4561  1350148
2         1  18594855       227     543           642  1350148
3         1   4260473     34048  175983        155050  1350148
4         2      2977      1274    2570         26091  1350148
```

3.0.2 Avazu

The **Avazu** dataset was originally released in 2014 for a CTR prediction Competition on Kaggle (Cite Avazu). The data is composed of 11 days worth mobile ad marketplace data. Much like the KDD12 dataset above, this dataset contains features ranging from user activity (clicks), user identification (device type, IP) to ad features. Notible differences to the KDD12 dataset include the fact that Avazu contains an "hour" feature (enabling the establishment of sequentiality of behaviours) and the fact that Avazu does not seem to contain query and ad texts.

[3]:
```
%run -i scripts/data_analysis_and_preprocessing/retrieve_avazu.py
```

```
Snapshot of Avazu training data:

                     id  click      hour    c1  banner_pos   site_id  \
0  15674134821169810910      1  14102300  1005           0  85f751fd
1  15674278914362889244      0  14102300  1005           0  85f751fd
2   1567455966106046075      0  14102300  1005           0  26fa1946
3  15674616734887926359      0  14102300  1005           0  85f751fd
4  15674670592044781339      0  14102300  1005           0  85f751fd

  site_domain site_category    app_id app_domain  … device_type  \
0    c4e18dd6      50e219e0  e71aba61   2347f47a  …           1
1    c4e18dd6      50e219e0  6f8bcb0f   2347f47a  …           1
2    e2a5dc06      3e814130  ecad2386   7801e8d9  …           1
3    c4e18dd6      50e219e0  53de0284   d9b5648e  …           1
4    c4e18dd6      50e219e0  a0fc55e5   2347f47a  …           1

  device_conn_type    c14  c15  c16   c17  c18  c19     c20  c21
0                0  21676  320   50  2495    2  167      -1   23
1                0  20476  320   50  2348    3  427  100005   61
2                0  20362  320   50  2333    0   39      -1  157
3                0  21611  320   50  2480    3  297  100111   61
4                0  20361  300  250  2333    0   39      -1  157

[5 rows x 24 columns]
```

3.0.3 Criteo

Finally, the Criteo dataset is another benchmark CTR prediction dataset that was originally released on Kaggle for a CTR prediction compitition. The original dataset is made up of 45 Million user's click activity, and contains the click/no-click target along with 26 categorical feature fields and 13 numerical feature fields. Unlike the other two datasets however, the semintic significance of these fields is not given - they are simply labelled as "Categorical 1-26" and "Numerical 1-13" respectively.

[4]:
```
%run -i scripts/data_analysis_and_preprocessing/retrieve_criteo.py
```

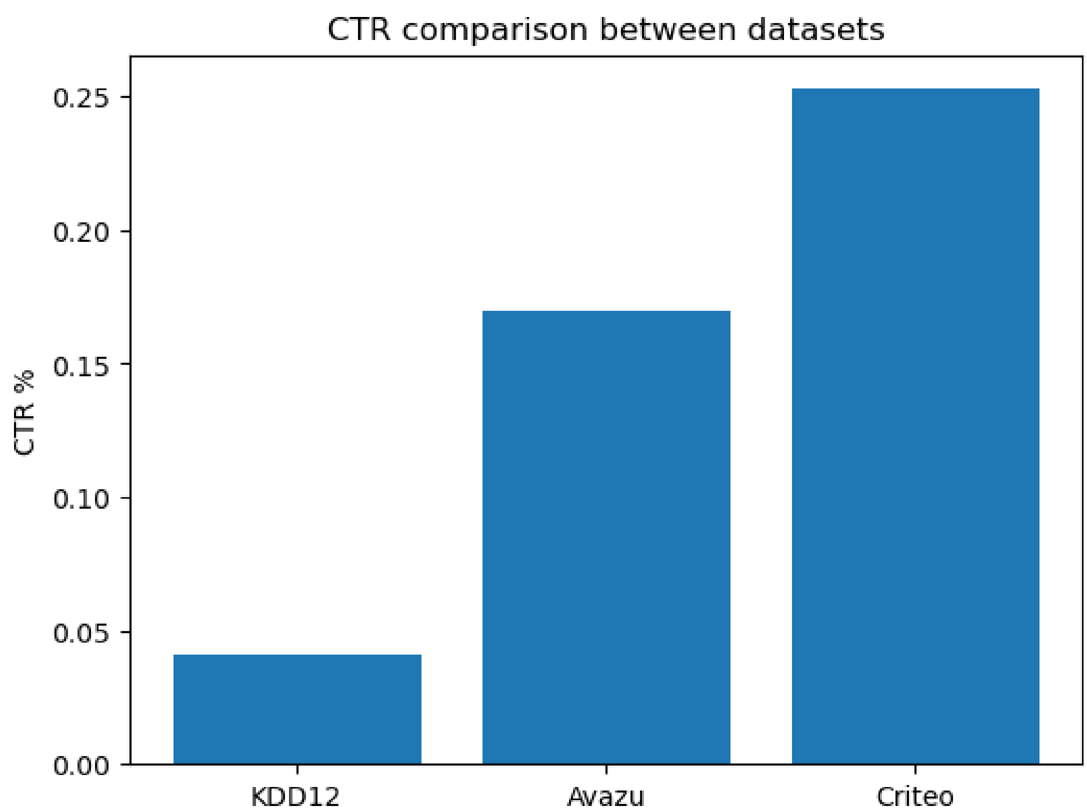
```
Snapshot of Criteo training data:

   click  int_1  int_2  int_3  int_4    int_5  int_6  int_7  int_8  int_9  \
0      0    NaN      1    2.0    5.0  27586.0   32.0    2.0   14.0   21.0
1      1   14.0      1    1.0    8.0    276.0   14.0   41.0    9.0   10.0
2      0    NaN      1   27.0   25.0      NaN    NaN    0.0   54.0   55.0
3      0    0.0    442    1.0    1.0   3029.0   58.0    2.0   13.0   44.0
4      0    0.0     -1    2.0    1.0   1167.0   88.0   23.0   19.0  673.0
```

```
   …    cat_17    cat_18    cat_19    cat_20    cat_21 cat_22    cat_23  \
0  …  07c540c4  bdc06043       NaN       NaN  6dfd157c    NaN  32c7478e
1  …  e5ba7672  87c6f83c       NaN       NaN  0429f84b    NaN  be7c41b4
2  …  2005abd1  87c6f83c       NaN       NaN  15fce809    NaN  be7c41b4
3  …  d4bb7bd8  cdfa8259       NaN       NaN  20062612    NaN  dbb486d7
4  …  27c07bd6  5bb2ec8e  49b8041f  b1252a9d  bff87997    NaN  32c7478e

     cat_24    cat_25    cat_26
0  ef089725       NaN       NaN
1  c0d61a5c       NaN       NaN
2  f96a556f       NaN       NaN
3  1b256e61       NaN       NaN
4  3fdb382b  f0f449dd  49d68486

[5 rows x 40 columns]
```

3.0.4 Target Variable Analysis

The figure below shows that the three datasets have vastly different average Click Through Rates per instance. The average CTR for the KDD12 dataset is only 3.4%, whereas the Criteo dataset is 25.6%.

[5]:
```
%run -i scripts/data_analysis_and_preprocessing/ctr_bar_charts.py
```

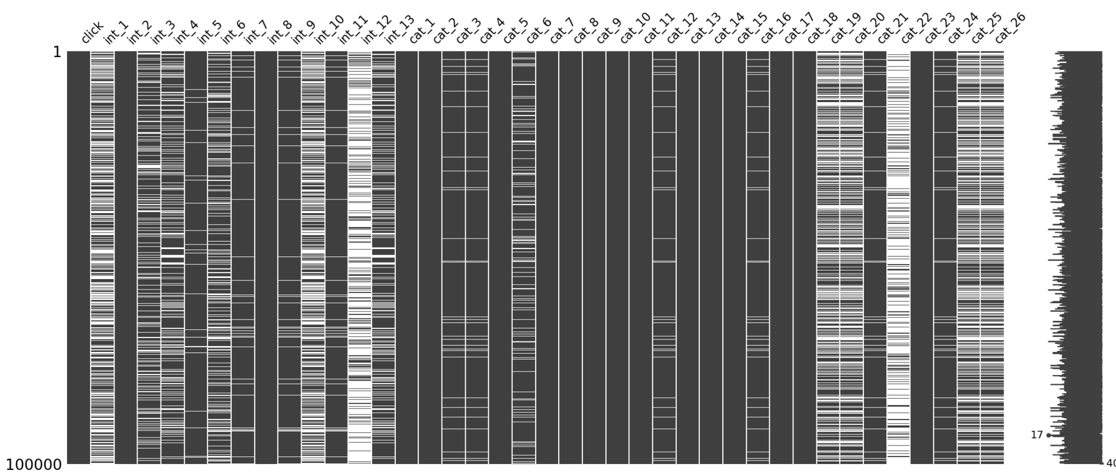


3.0.5 Missingness and Data Imputation

Below, I take a look at whether or not our dataset has any missing values.

[6]:
```
# Show missingness matrix for criteo dataset
print("Missingness matrix for Criteo dataset:")
msno.matrix(criteo)
plt.show()
```

```
Missingness matrix for Criteo dataset:
```



Above we see that Criteo has some missing values. Below I proceed by imputing the missing values using Sklearn's KNN Imputer. The code for these imputations does not get executed here.

Imputation steps taken were:

1. Factorize categorical values in the dataset, converting them to integers. This was done because sklearn's imputers only work with numerical data.
2. Use the sklearn's IterativeImputer with HistGradientBoostingRegressor to impute the missing values. This was recommended in this github discussion for imputing data with categorical values.
3. Concatinate missingness indicators to the dataset as additional features, as recommended by Van Buuren (2018)

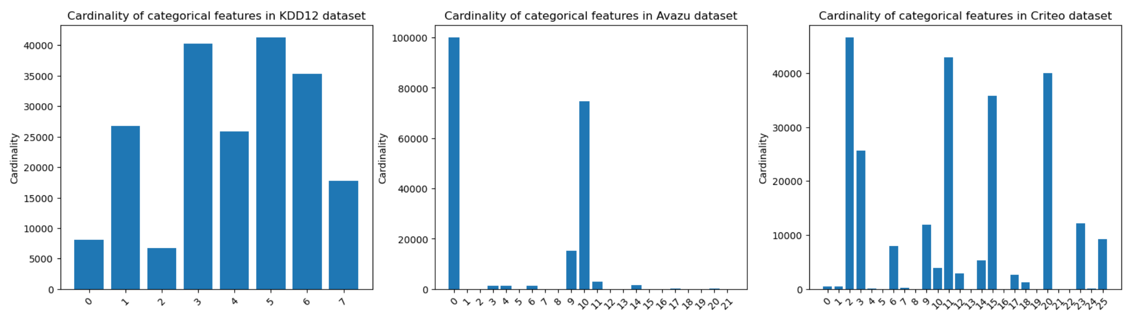
The script for the above is in scripts/data_analysis_and_processing/impute_criteo_nulls.py.

[7]:
```
# Pick up the imputed dataset
criteo_imputed_inds = pd.read_csv('./data/criteo/criteo_train_imputed.csv').
 ↪astype('int')
criteo_imputed_inds[criteo.columns[criteo.dtypes == 'category']] =␣
 ↪criteo_imputed_inds[criteo.columns[criteo.dtypes == 'category']].
 ↪clip(lower=0,upper=None)
```

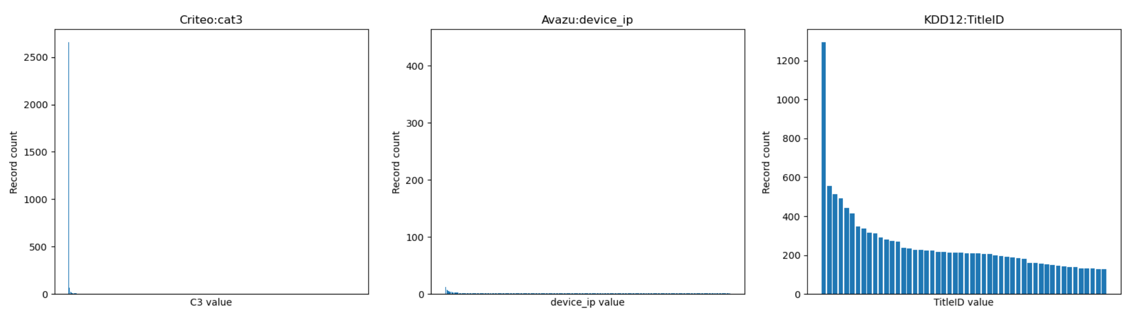
3.0.6 Sparse Multi-Value Categorical Features

As already mentioned above, ad marketplace data often contains sparse categorical features, which make signal detection extremely difficult in shallow modelling frameworks. Below I show examples from each dataset

[11]:
```
%run -i scripts/data_analysis_and_preprocessing/plot_cardinalities.py
```



[12]:
```
%run -i scripts/data_analysis_and_preprocessing/
↪plot_high_cardinality_record_counts.py
```



A common remidy to the above issue is to *bin* the categorical feature values before one-hot encoding or embedding, according to some given threshold (Cite Song, Others). This essentially means that for a given threshold t, we retain only the values for the multi-value categorical features that have more than t occurances in the dataset. (Cite Song) Reccomends usign, setting $t = 10, 5, 10$ for Criteo, KDD12 and Avazu respectively. Due to computational limitations, this was multiplied by a factor of 100

[13]:
```
%run -i scripts/data_analysis_and_preprocessing/binned_OH_encoding.py
```

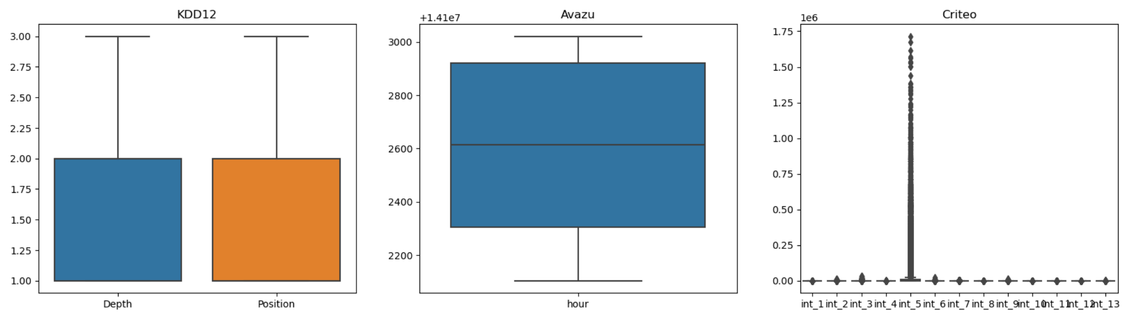
```
Before one-hot encoding:
KDD12 shape: (100000, 12)
Avazu shape: (100000, 24)
Criteo shape: (100000, 64)
```

```
After one-hot encoding:
KDD12 shape: (100000, 12)
Avazu shape: (100000, 24)
Criteo shape: (100000, 64)
Sparse output:
KDD12 shape: (100000, 1478)
Avazu shape: (100000, 924)
Criteo shape: (100000, 2003)
```

3.0.7 High Variance Numerical outliers

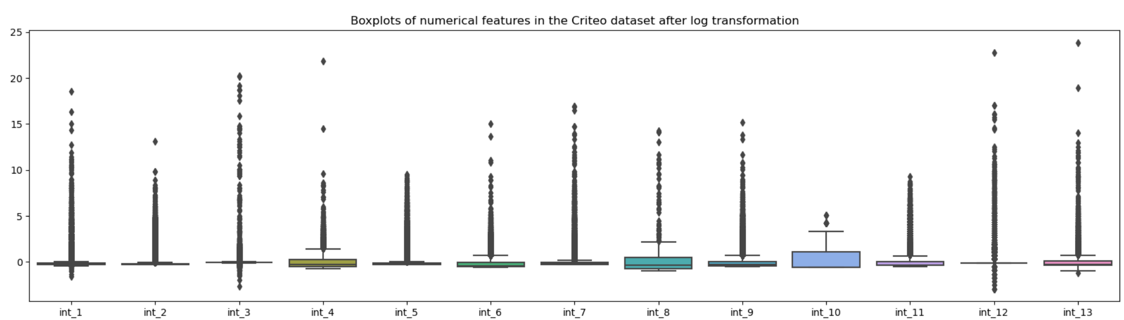
Below I check the distributions of the numerical features in the datasets

[15]:
```
%run -i scripts/data_analysis_and_preprocessing/plot_numerical_distributions.py
```



Due to the high variance of numerical features in the Criteo dataset, it is necessary to transform these variable in order to ease the training of deep NN's. As done be (Cite Song and Wang, and the winner of the Criteo Competition), we will proceed by applying the transform $\log^2(z)$ if $z > 2$, and where z is the standardized numerical value.

[16]:
```
%run -i scripts/data_analysis_and_preprocessing/numerical_standardization.py
```



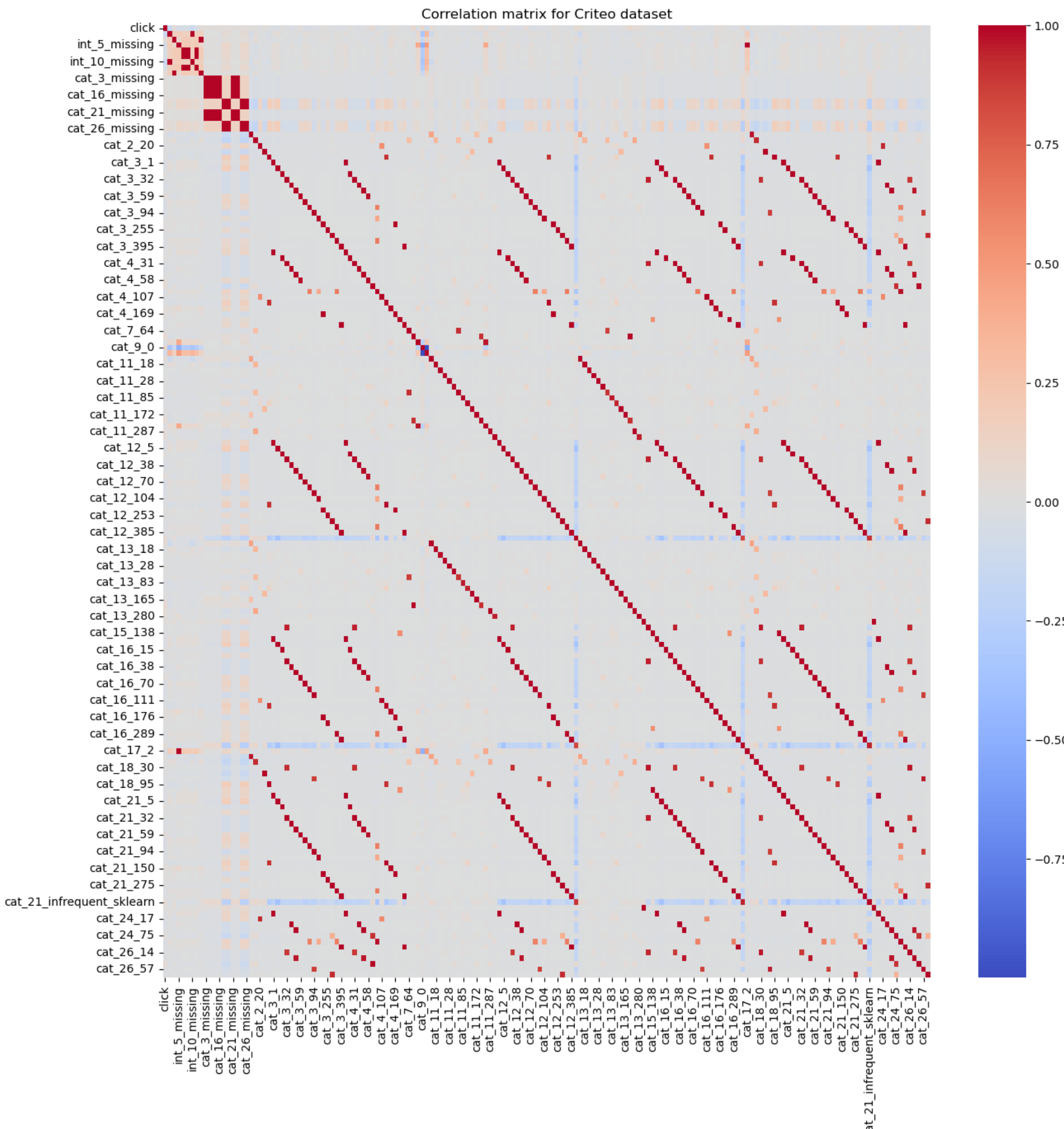
[17]:
```
%run -i scripts/data_analysis_and_preprocessing/export_preprocessed.py
```

3.0.8 Correlation Analysis

In the following script, conduct a correlation analysis of the features to the Click-Through rate.

[20]:
```
Image(filename = './figures/criteo_corr_matrix.png')
```

[20]:



Some very high correlations between some of the features across fields in all three datasets. This possibly points to there being potential for dimensionality reduction across this feature set.

Unfortunately, from the correlation heatmaps, there appears to be little to no correlation between the first-order features and the target click variable.

4 Modelling

In this section I will compare the perfomance of two shallow modelling approaches (Logistic Regression and Factorization Machines) to a naive DNN for CTR prediction. As with (Cite Song and Wang), I will use the **Area Under the ROC Curve** and **Logloss** measures to compare the performance of the different modelling approaches on the test set.

[21]:
```
%run -i scripts/modelling/load_and_prep_data.py
```

4.1 Logistic Regression

[22]:
```
%run -i scripts/modelling/fit_lr_models.py
```

[23]:
```
%run -i scripts/modelling/score_lr_models.py
```

```
KDD12:
Log loss: 0.1626051997902124
ROC AUC: 0.6991046240417678
Accuracy: 0.95825

Avazu:
Log loss: 0.4122042376810601
ROC AUC: 0.718712375456128
Accuracy: 0.8321

Criteo:
Log loss: 0.4934800596857564
ROC AUC: 0.7450121989735492
Accuracy: 0.7671
```

[24]:
```
%run -i scripts/modelling/save_lr_models.py
```

4.2 Factorization Machine

Below I proceed by applying the SGD solver, as shown in the relevant tutorial for FastFM (Cite).

> **Note**: Unfortunately, the fastFM library is currently only compatible with Linux and iOS. Since I have a Windows PC, I ran the training script below on an AWS Sagemaker Instance.

[ ]:
```
%run -i scripts/modelling/fit_fm_models.py
```

[48]:
```
%run -i scripts/modelling/score_fm_models.py
```

```
KDD12:
Log loss: 0.31685987446578695
ROC AUC: 0.5336269490760196
Accuracy: 0.95825

Avazu:
```

```
Log loss: 10.974548402513749
ROC AUC: 0.5212335152327499
Accuracy: 0.66185

Criteo:
Log loss: 15.774504905747122
ROC AUC: 0.47794380839584155
Accuracy: 0.56235
```

[49]:
```
%run -i scripts/modelling/save_fm_models.py
```

4.3 MLP

In this section, I implement a simple 3 layer MLP for CTR prediction.

[25]:
```
%run -i scripts/modelling/load_tf_datasets.py
```

[32]:
```
%run -i scripts/modelling/fit_kdd12_mlp.py
%run -i scripts/modelling/fit_avazu_mlp.py
%run -i scripts/modelling/fit_criteo_mlp.py
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 128)               189056

 batch_normalization_4 (Batc  (None, 128)              512
 hNormalization)

 dropout_3 (Dropout)         (None, 128)               0

 dense_6 (Dense)             (None, 64)                8256

 batch_normalization_5 (Batc  (None, 64)               256
 hNormalization)

 dropout_4 (Dropout)         (None, 64)                0

 dense_7 (Dense)             (None, 32)                2080

 batch_normalization_6 (Batc  (None, 32)               128
 hNormalization)

 dropout_5 (Dropout)         (None, 32)                0

 dense_8 (Dense)             (None, 1)                 33

=================================================================
```

```
Total params: 200,321
Trainable params: 199,873
Non-trainable params: 448

_________________________________________________________________
None
```

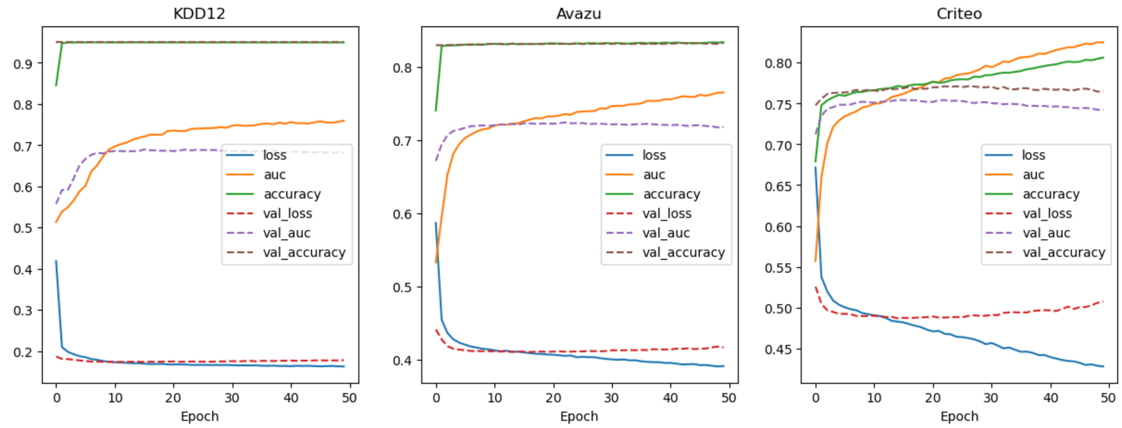
```
The Kernel crashed while executing code in the current cell or a previous cell.


Please review the code in the cell(s) to identify a possible cause of the␣
 ↪failure.


Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for more info.


View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further details.
```

[ ]:

```

```

[ ]:

```
%run -i scripts/modelling/plot_mlp_training.py
```



[ ]:

```
# Load the models
kdd12_mlp = tf.keras.models.load_model('./models/mlp/kdd12_mlp.keras')
avazu_mlp = tf.keras.models.load_model('./models/mlp/avazu_mlp.keras')
criteo_mlp = tf.keras.models.load_model('./models/mlp/criteo_mlp.keras')

# Evaluate MLP models
print("KDD12:")
kdd12_mlp.evaluate(kdd12_val_tfd)
print("\nAvazu:")
avazu_mlp.evaluate(avazu_val_tfd)
print("\nCriteo:")
```

```
criteo_mlp.evaluate(criteo_val_tfd)
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
Cell In[31], line 3
      1 # Load the models
      2 kdd12_mlp = tf.keras.models.load_model('./models/mlp/kdd12_mlp.keras')
----> 3 avazu_mlp = tf.keras.models.load_model('./models/mlp/avazu_mlp.keras')
      4 criteo_mlp = tf.keras.models.load_model('./models/mlp/criteo_mlp.keras'
      6 # Evaluate MLP models

File c:
↪\Users\marti\anaconda3\envs\deep_learning\lib\site-packages\keras\utils\traceback_utils.
↪py:70, in filter_traceback.<locals>.error_handler(*args, **kwargs)
     67     filtered_tb = _process_traceback_frames(e.__traceback__)
     68     # To get the full stack trace, call:
     69     # `tf.debugging.disable_traceback_filtering()`
---> 70     raise e.with_traceback(filtered_tb) from None
     71 finally:
     72     del filtered_tb

File c:
↪\Users\marti\anaconda3\envs\deep_learning\lib\site-packages\h5py\_hl\files.py
↪567, in File.__init__(self, name, mode, driver, libver, userblock_size, swmr,
↪rdcc_nslots, rdcc_nbytes, rdcc_w0, track_order, fs_strategy, fs_persist,
↪fs_threshold, fs_page_size, page_buf_size, min_meta_keep, min_raw_keep,
↪locking, alignment_threshold, alignment_interval, meta_block_size, **kwds)
    558     fapl = make_fapl(driver, libver, rdcc_nslots, rdcc_nbytes, rdcc_w0,
    559                      locking, page_buf_size, min_meta_keep, min_raw_keep,
    560                      alignment_threshold=alignment_threshold,
    561                      alignment_interval=alignment_interval,
    562                      meta_block_size=meta_block_size,
    563                      **kwds)
    564     fcpl = make_fcpl(track_order=track_order, fs_strategy=fs_strategy,
    565                      fs_persist=fs_persist, fs_threshold=fs_threshold,
    566                      fs_page_size=fs_page_size)
--> 567     fid = make_fid(name, mode, userblock_size, fapl, fcpl, swmr=swmr)
    569 if isinstance(libver, tuple):
    570     self._libver = libver

File c:
↪\Users\marti\anaconda3\envs\deep_learning\lib\site-packages\h5py\_hl\files.py
↪231, in make_fid(name, mode, userblock_size, fapl, fcpl, swmr)
    229     if swmr and swmr_support:
    230         flags |= h5f.ACC_SWMR_READ
--> 231     fid = h5f.open(name, flags, fapl=fapl)
    232 elif mode == 'r+':
    233     fid = h5f.open(name, h5f.ACC_RDWR, fapl=fapl)
```

```
File h5py\_objects.pyx:54, in h5py._objects.with_phil.wrapper()

File h5py\_objects.pyx:55, in h5py._objects.with_phil.wrapper()

File h5py\h5f.pyx:106, in h5py.h5f.open()

OSError: Unable to open file (file signature not found)
```

5 Summary of findings

6 Suggested Future Research

7 References

- eMarketer. (2023). Digital advertising spending worldwide from 2021 to 2027 (in billion U.S. dollars) . Statista. Statista Inc.. Accessed: June 09, 2024. https://www-statista-com.iclibezp1.cc.ic.ac.uk/statistics/237974/online-advertising-spending-worldwide/
- Aden, Yi Wang. (2012). KDD Cup 2012, Track 2. Kaggle. https://kaggle.com/competitions/kddcup2012-track2
- Steve Wang, Will Cukierski. (2014). Click-Through Rate Prediction. Kaggle. https://kaggle.com/competitions/avazu-ctr-prediction
- Jean-Baptiste Tien, joycenv, Olivier Chapelle. (2014). Display Advertising Challenge. Kaggle. https://kaggle.com/competitions/criteo-display-ad-challenge
- Van Buuren, S. (2018). Flexible imputation of missing data. CRC press.
- Song, W., Shi, C., Xiao, Z., Duan, Z., Xu, Y., Zhang, M., & Tang, J. (2019, November). Autoint: Automatic feature interaction learning via self-attentive neural networks. In Proceedings of the 28th ACM international conference on information and knowledge management (pp. 1161-1170).
- Wang, F., Gu, H., Li, D., Lu, T., Zhang, P., & Gu, N. (2023, October). Towards Deeper, Lighter and Interpretable Cross Network for CTR Prediction. In Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (pp. 2523-2533).