Applied Deep Learning
Fall'2018

Project Report

Click-Through Rate (CTR) Prediction

Presented By:

ROHIT DALAL

HARISH VISWESWARAN

Objective:

- In online advertising, click-through rate (CTR) is a very important metric for evaluating ad performance. As a result, click prediction systems are essential and widely used for sponsored search and real-time bidding.

- Goal is to build a model to predict whether an ad will be clicked or not.

- We chose this over the class project due to the fact it represents majority of the industry scenarios where teams have structured data at their disposal compared to images or other unstructured data. Application of Deep Learning Models on structured data is the most prominent business use case.

Approach

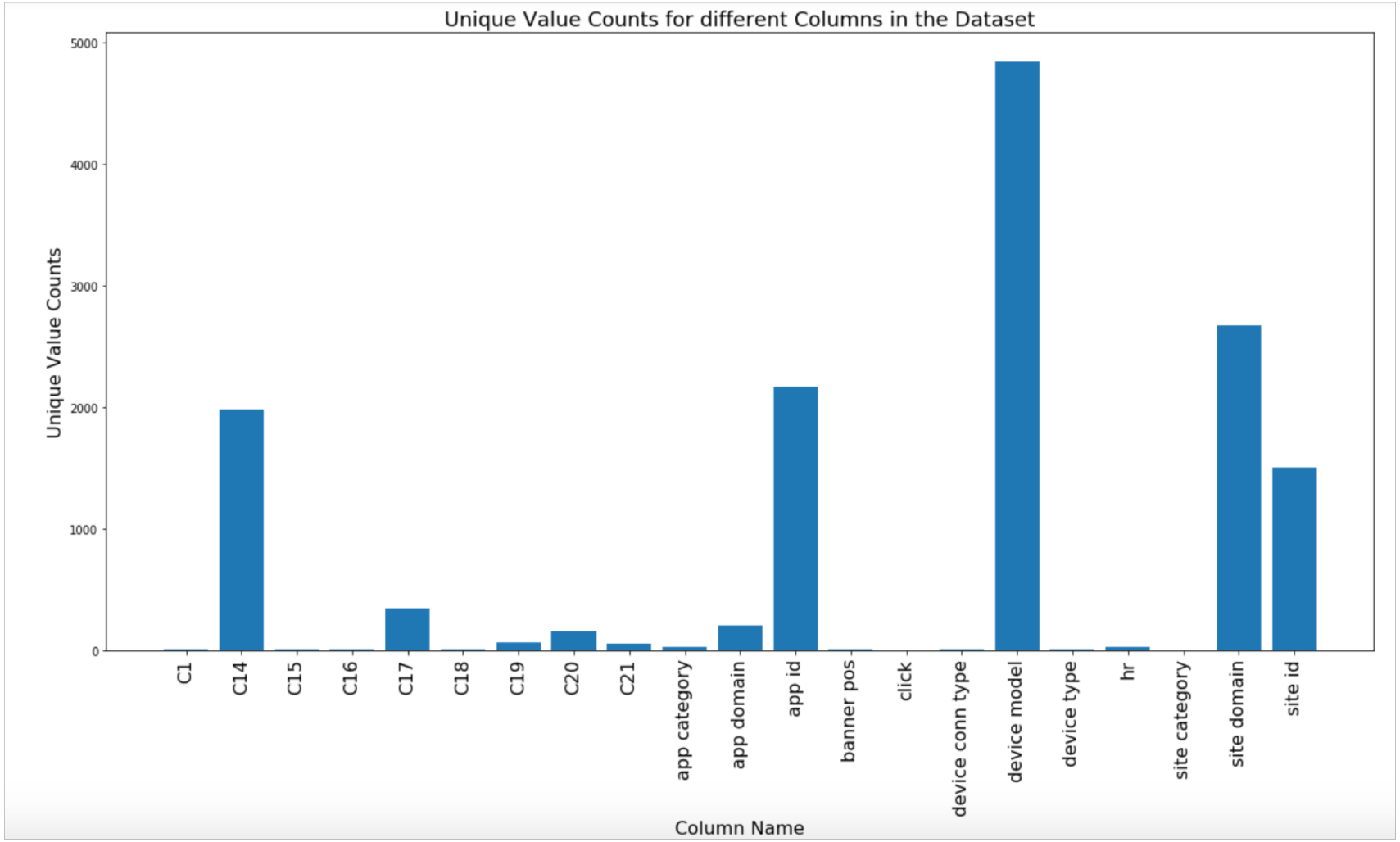
- Implement both standard ML Classification Models and Deep Learning Models to compare
    a) Performance
    b) When to use one v/s other

- **Models To Consider:**
    a) Random Forests
    b) DNN
    c) Wide and Deep

Features

- Id: Ad Identifier
- Hour: Format is YYMMDDHH
- C1: Anonymized Categorical Variable
- Banner_Pos
- Site_Id
- Site_Domain
- Site_Category
- App_Id
- App_Domain
- App_Category
- Device_Id
- Device_Ip
- Device_Model
- Device_Type
- Device_Conn_Type
- C14 - C21: Anonymized Categorical Variables
- Click: 0/1 for non-click/click

Data Statistics and Metrics



- Dataset has many features with high cardinality

Data Statistics and Metrics

- Imbalanced Dataset
  - 86% non-clicks (0s) and 14% clicks (1s)

- Metrics
  - Accuracy – Not the best measure for this dataset as it is highly imbalanced
  - Precision – Useful measure of model performance
  - Recall – Another useful measure of model performance. Typically a balance between precision and recall
  - F1 Score – Measure to combine precision and recall into one metric
  - Log Loss – Standard loss reported for model comparison and performance. Very useful from a model standpoint but not as useful from a business standpoint

Data Preparation

- Create hour of the day field
- Drop Device ID and IP fields
  - For this run of the model, we did not want to consider user identifier. This is to avoid the scenario of having to score for new users
- For high cardinality features, create ‘Other’ bucket for values that occur infrequently
- One hot-encode features resulting in 178 features for the model (most of them one-hot encoded categorical variables after bucketing)
- When using embeddings, we do not create the ‘Other’ bucket

Random Forests - Methodology

- 80/20 split (stratified by target variable) – run twice to create Train, Validation and Test
- 5 Fold Cross Validation with Grid Search
  - Grid Search Space: Max Depth: [15, 20, 25, 40] | # of Estimators: [50, 100] | Min Samples Split: [5, 10]
- Find best parameters based on F1 Score
- Consider adjusting probability threshold for classifying 1s if higher recall is required
- Report performance on Test Dataset

Random Forests - Results

- Given the imbalanced dataset, if we use a random prediction (for each ad, predict click/not click randomly with 0.5 probability) as a baseline we get:
  - Precision: 0.137
- Best model parameters based on Grid Search
  - Max Depth: 40 | # of Estimators: 50 | Min Samples Split: 5
  - General Observation – Larger Tree with lower threshold split performing best based on F1 score
- Metrics based on 0.5 probability threshold
  - Accuracy: 0.856
  - Precision: 0.362
  - Recall: 0.070
  - F1 Score: 0.117
  - Log Loss: 0.4992

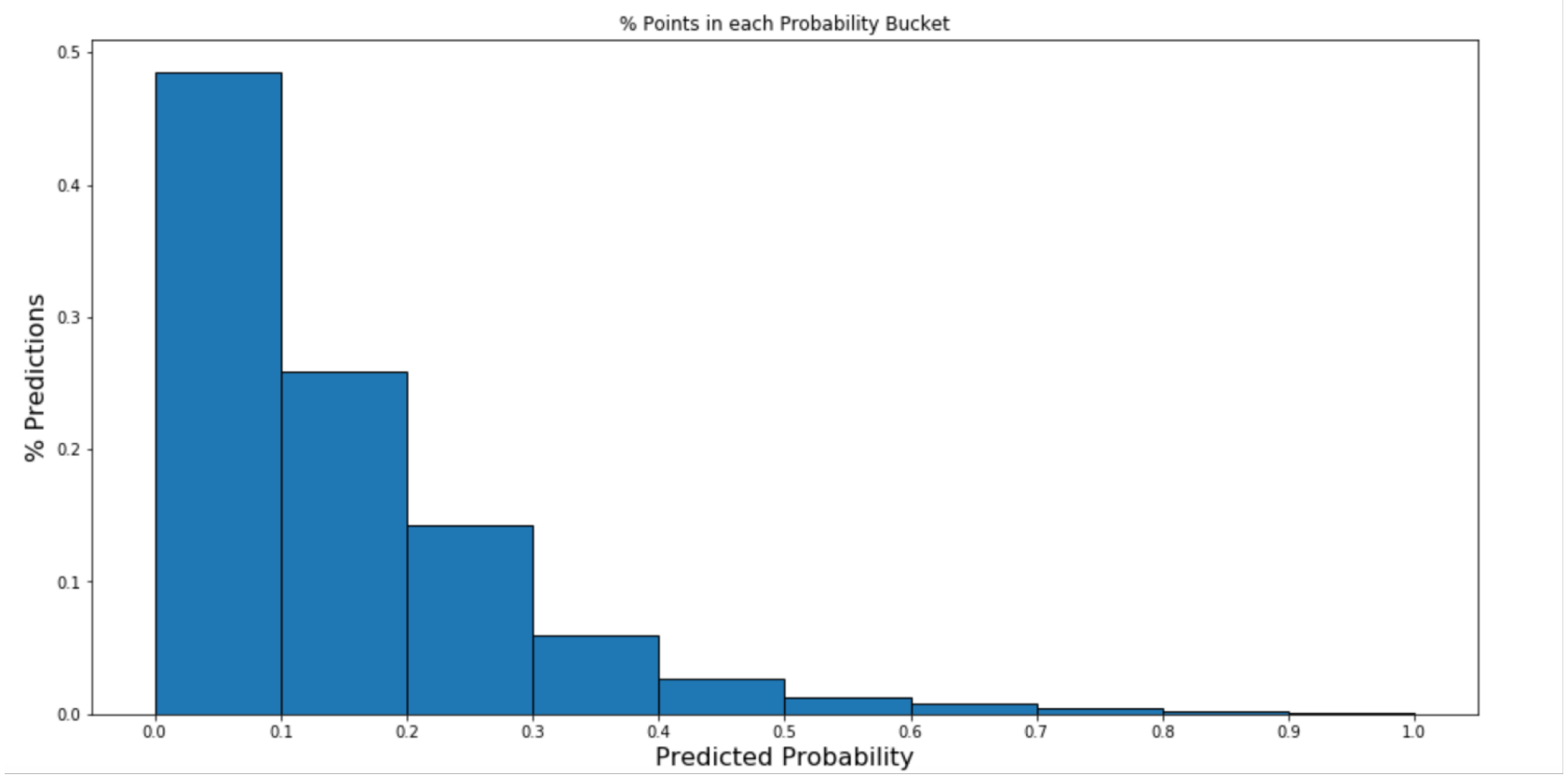
Random Forests – Thresholds

- We may be interested in sacrificing some precision to get higher recall. Testing out different thresholds for classification

Threshold	Accuracy	F1	Precision	Recall
0.1	0.541	0.300	0.189	0.720
0.2	0.728	0.309	0.237	0.446
0.3	0.811	0.250	0.272	0.231
0.4	0.843	0.171	0.309	0.119
0.5	0.856	0.117	0.362	0.070
0.6	0.861	0.079	0.418	0.044
0.7	0.863	0.050	0.487	0.026
0.8	0.864	0.030	0.576	0.016
0.9	0.864	0.009	0.552	0.005

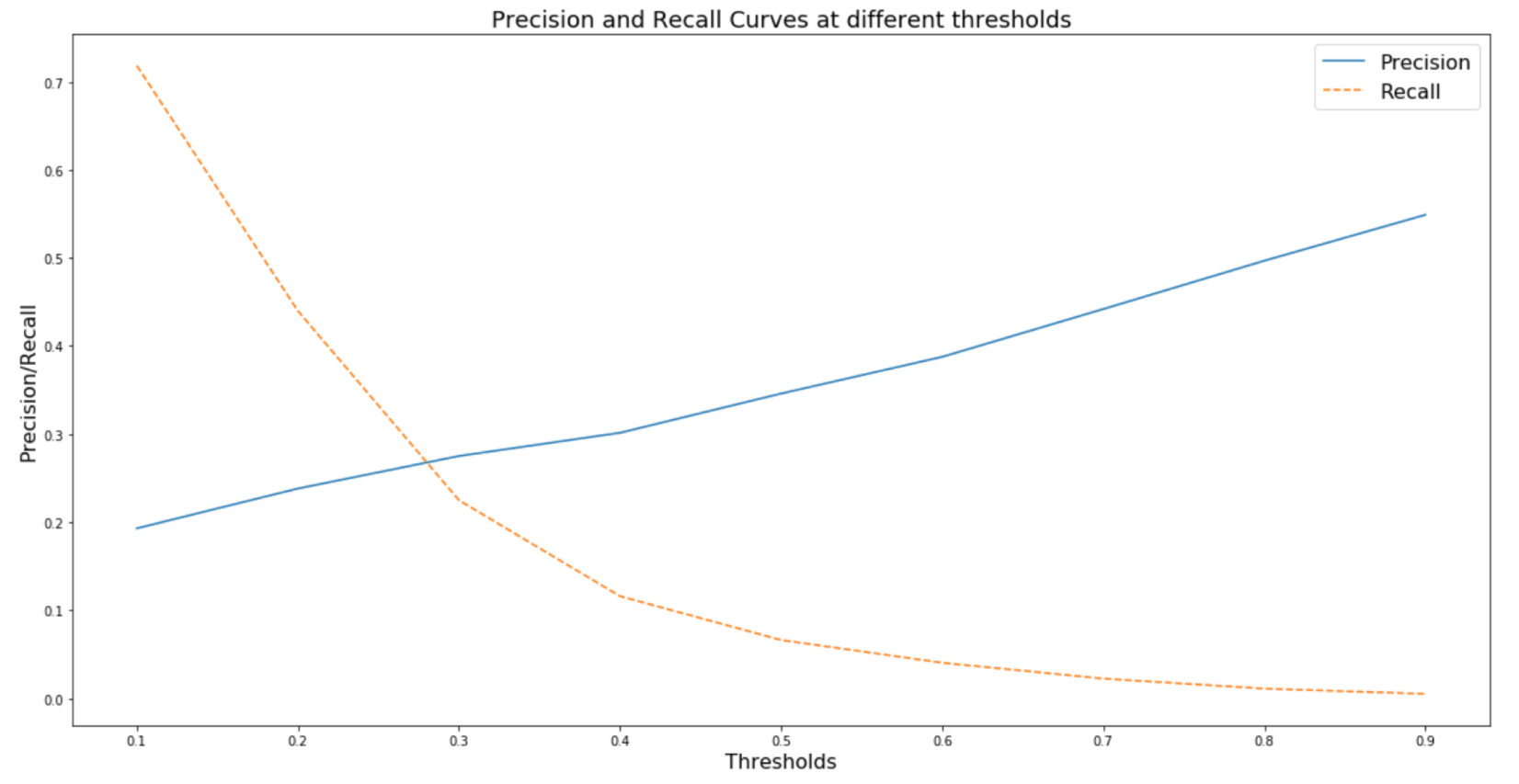
Random Forests – Precision/Recall

- Model generally predicts a very small number of 1s resulting a low recall rate
  - Default threshold of 0.5 is being used here – only 2.65% of ads are predicted to be clicked
  - The imbalance in the data is one of the reasons for this



Random Forests – Precision/Recall

- A threshold of around 0.32 seems to provide a good balance between precision and recall rates. Given this is an imbalanced dataset with ~14% of 1s, a good balance for a useful model seems to be to get about ~30% precision and ~20% recall



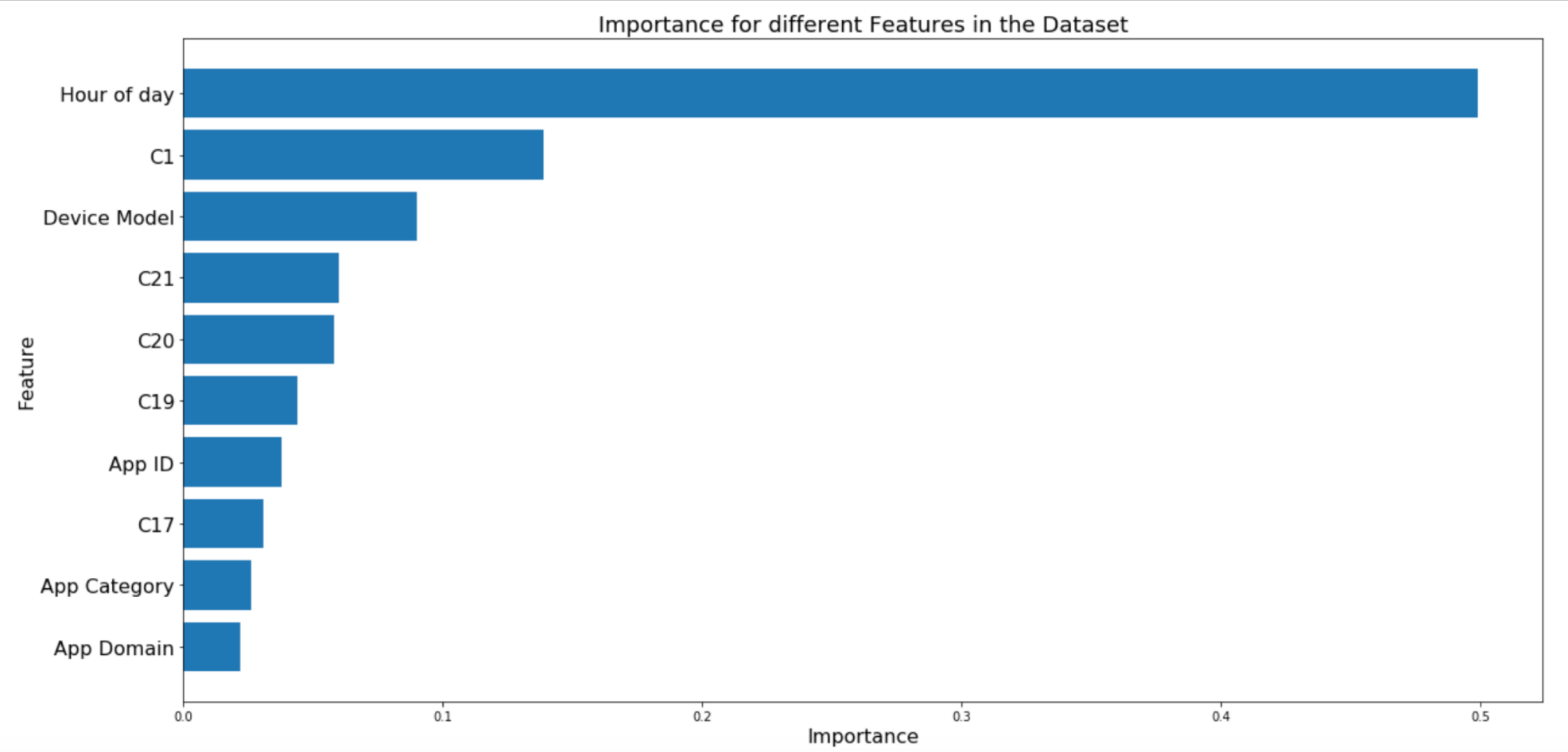
Random Forests – Feature Importance

- One of the advantages of using traditional ML methods (as opposed to DL methods) is that the feature importance can be obtained very easily
- Since we have used one-hot encoding of features after basic bucketing, we have some features with multiple values from the same category in our feature importance
- We also combined the feature importance of the different one-hot encodings to create overall importance for the input variables as this is more understandable

Feature	Importance
Hour of day	0.499
Device Model: Other Bucket	0.018
C20: Value -1.0	0.016
C21: Value 15	0.013
C20: Other Bucket	0.012
Device Connection Type: Value 0	0.009
App ID: Other Bucket	0.008
App Category: Value 0f2161f8	0.008
C18: Value 2	0.007
C17: Other Bucket	0.007

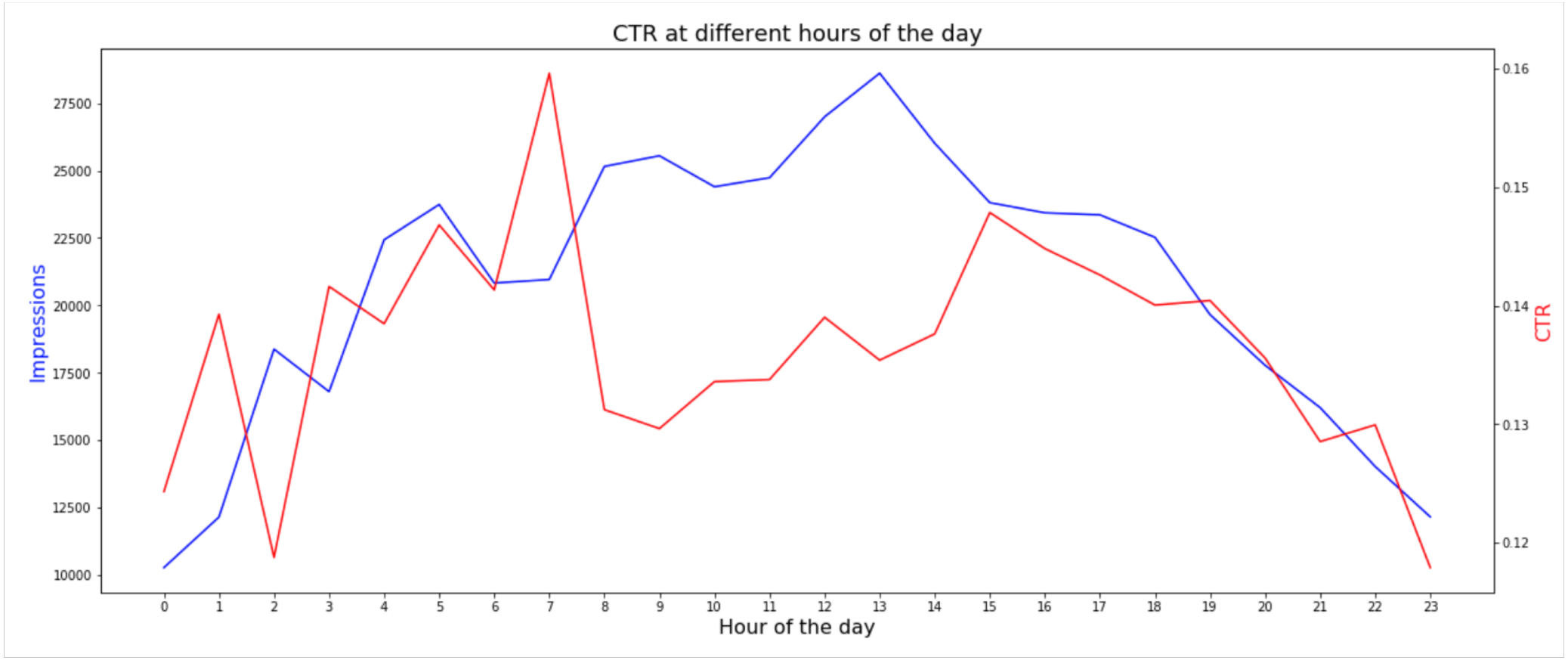
Feature	Importance
Hour of day	0.499
C1	0.139
Device Model	0.090
C21	0.060
C20	0.058
C19	0.044
App ID	0.038
C17	0.031
App Category	0.026
App Domain	0.022

Random Forests – Feature Importance



Random Forests – Feature Importance

- We clearly notice that there is a difference in CTR at different hours of the day



Random Forests – Balanced Sample

- We also tried an alternate sampling method. Training dataset was sampled to be balanced (50% clicks and 50% non-clicks) while validation and test datasets were maintained to have the original ratio of ~86% clicks and ~14% non-clicks. The performance on the validation and test datasets improve a little but not too much.

- Performance on the test dataset after training on a balanced dataset:
  - Accuracy: 0.828
  - Precision: 0.287
  - Recall: 0.211
  - F1 Score: 0.243

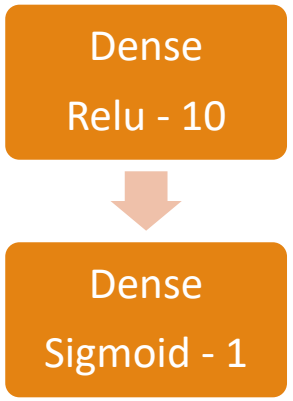
DNN- Methodology

- 80/20 split (stratified by target variable) – run twice to create Train, Validation and Test
- Use feature column layer with Dense layers on the top
- Run and compare Small/Medium/Large/Large (with Dropout) DNNs with:
  - Indicator Columns (One Hot Encoding) FC layer
  - Embedding Columns FC Layer
- Use different combinations of Activation Functions/Optimizers/Learning Rate/Epochs
- Find best network architecture based on minimizing Loss and Overfitting
- Consider adjusting probability threshold for classifying 1s if higher recall is required
- Report performance on Test Dataset

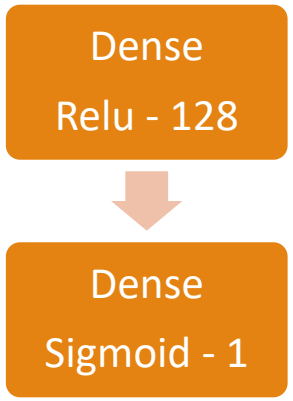
Deep Neural Network – Architectures

- Multiple network architectures on the data – small, medium, large and large with dropout

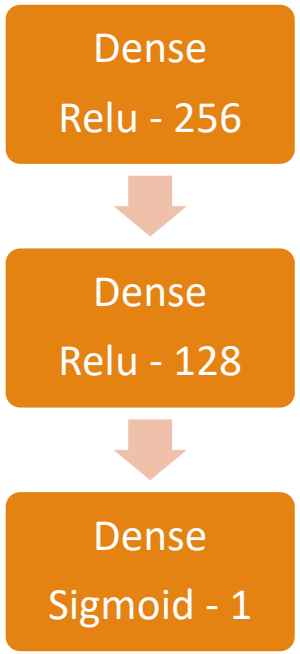
Small Network



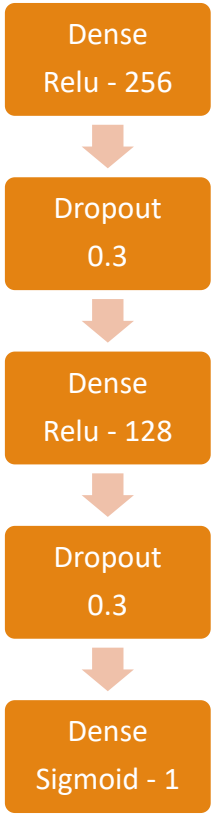
Medium Network



Large Network

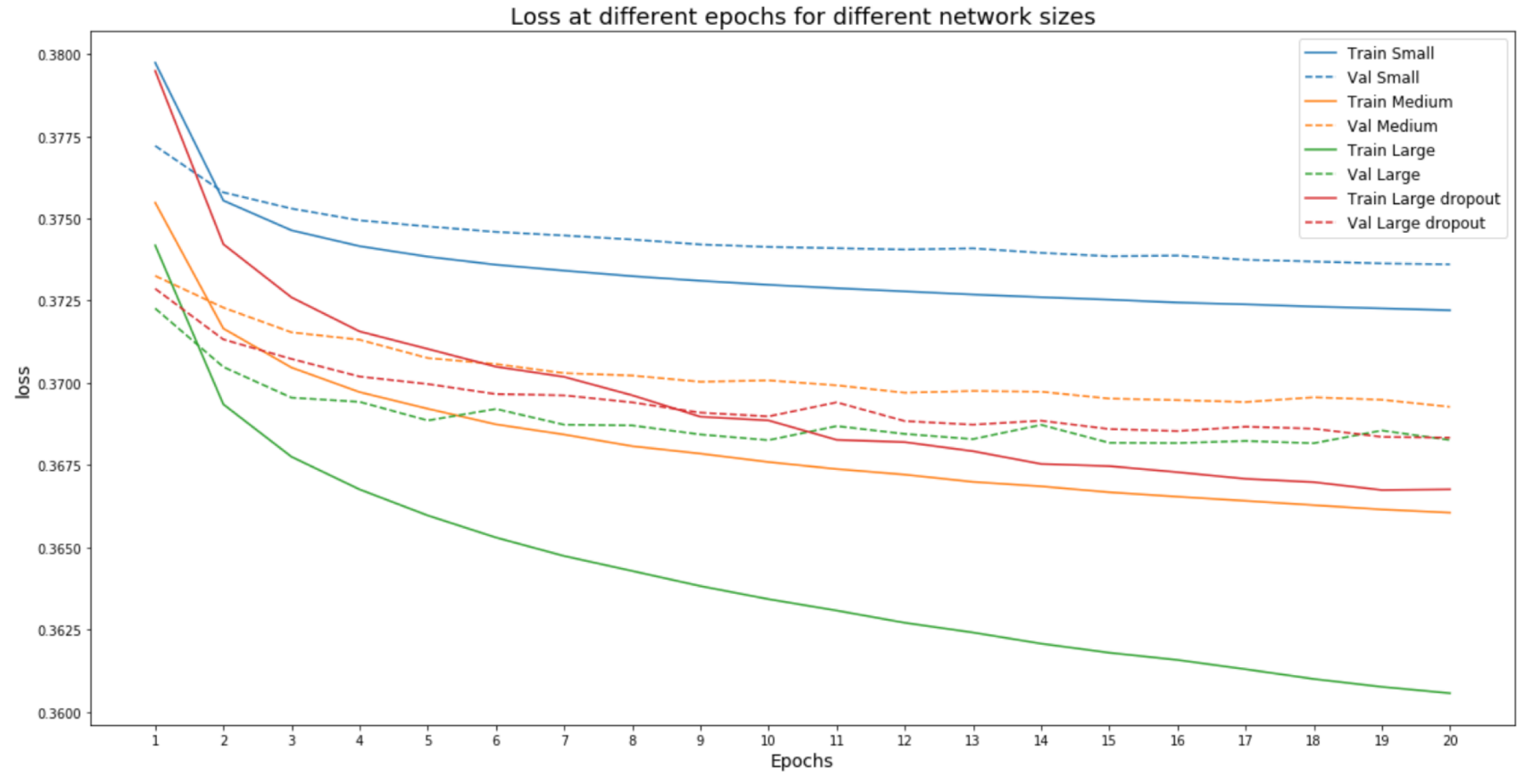


Large Network with Dropout



Deep Neural Network – Loss Curve

- From the below plot, we can see that the large network with dropout has good performance on the data and also does not overfit unlike the large network without dropout



DNN - Results

- Best model parameters
  - Activation Function: Relu| Optimizer: Adam| Loss: Binary Crossentropy
- General Observation
  - Larger network with dropout performs better
  - Embedding FC layer takes more time to train compared to one hot encoding
- Metrics based on the default 0.5 probability threshold
  - Accuracy: 0.862
  - Precision: 0.666
  - Recall: 0.022
  - F1 Score: 0.042

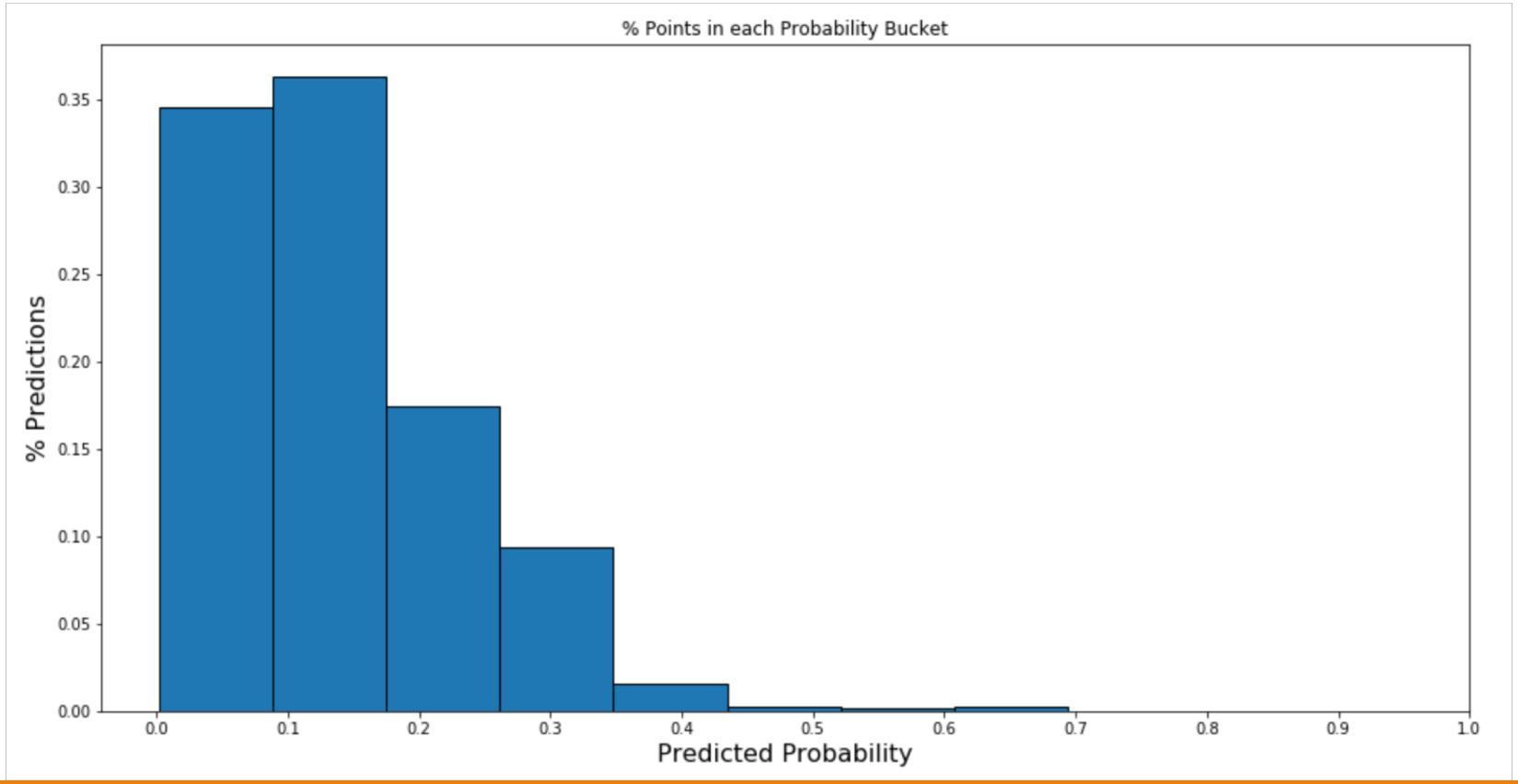
DNN – Thresholds(One Hot Encoding)

- We may be interested in sacrificing some precision to get higher recall. Testing out different thresholds for classification

Threshold	Accuracy	F1	Precision	Recall
0.1	0.495	0.314	0.194	0.831
0.2	0.756	0.342	0.274	0.455
0.3	0.842	0.224	0.354	0.164
0.4	0.861	0.067	0.535	0.036
0.5	0.862	0.042	0.666	0.022
0.6	0.862	0.027	0.692	0.014
0.7	0.861	0.005	0.733	0.002
0.8	0.861	0.001	1.000	0.001
0.9	0.861	0.000	0.000	0.000

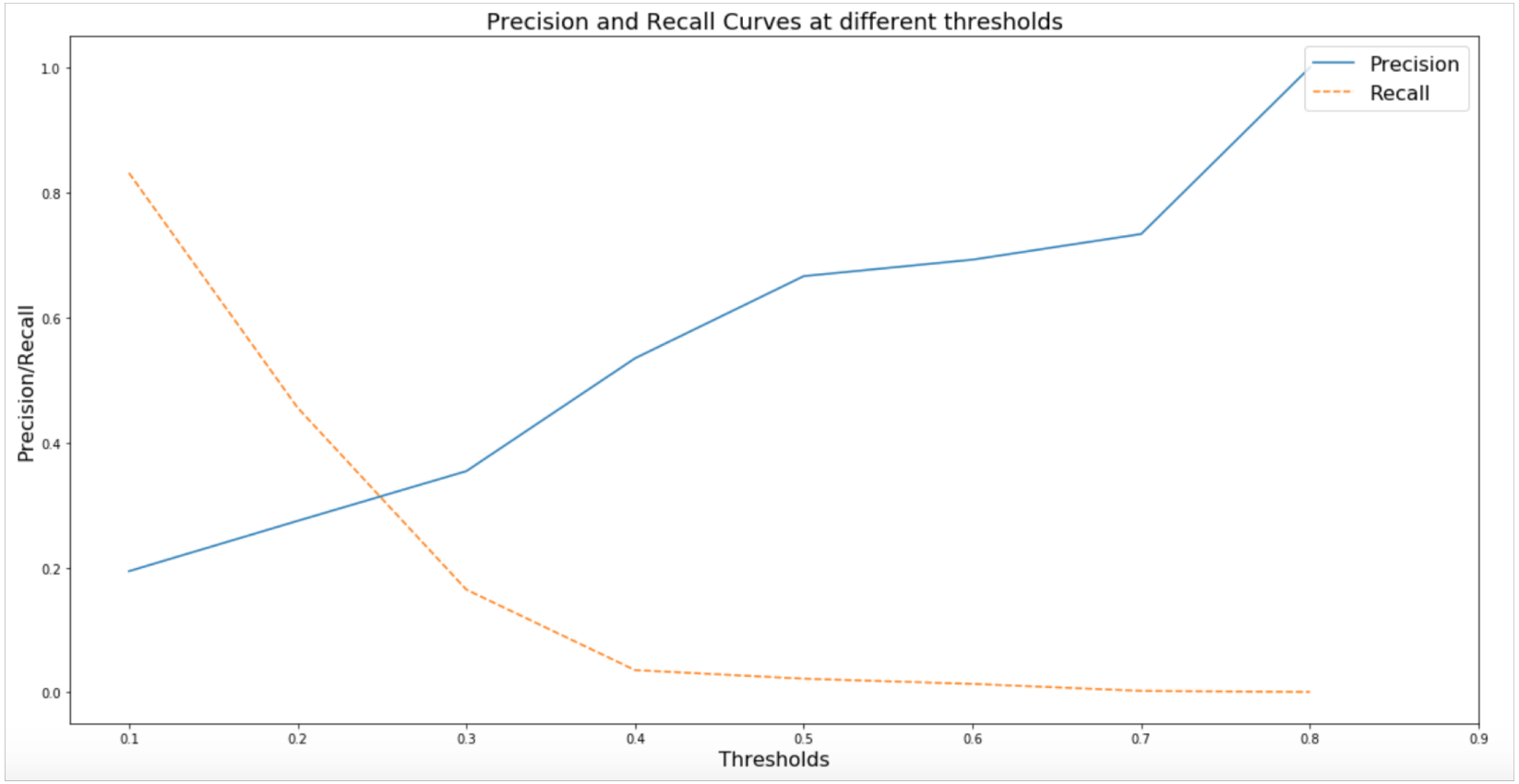
DNN (One Hot Encoding) – Precision/Recall

- Similar to the RF model, the DNN model also predicts a very small number of 1s resulting a low recall rate
- When the default threshold of 0.5 is used, only 0.4% of ads are predicted to be clicked



DNN (One Hot Encoding) – Precision/Recall

- A threshold of around 0.27 seems to provide a good balance between precision and recall rates. Given this is an imbalanced dataset with ~14% of 1s, a good balance for a useful model seems to be to get about ~33% precision and ~23% recall and the metrics are very comparable to the Random Forest Model



Note: The threshold of 0.9 results in no prediction of 1s and hence precision and recall are 0. These points have been removed from the chart

DNN – Balanced Sample

- We also tried the above DNN model after using an alternate sampling method. Training dataset was sampled to be balanced (50% clicks and 50% non-clicks) while validation and test datasets were maintained to have the original ratio of ~86% clicks and ~14% non-clicks. The performance on the validation and test datasets did not improve significantly.

- Performance on the test dataset after training on a balanced dataset:
  - Accuracy: 0.829
  - Precision: 0.331
  - Recall: 0.228
  - F1 Score: 0.270

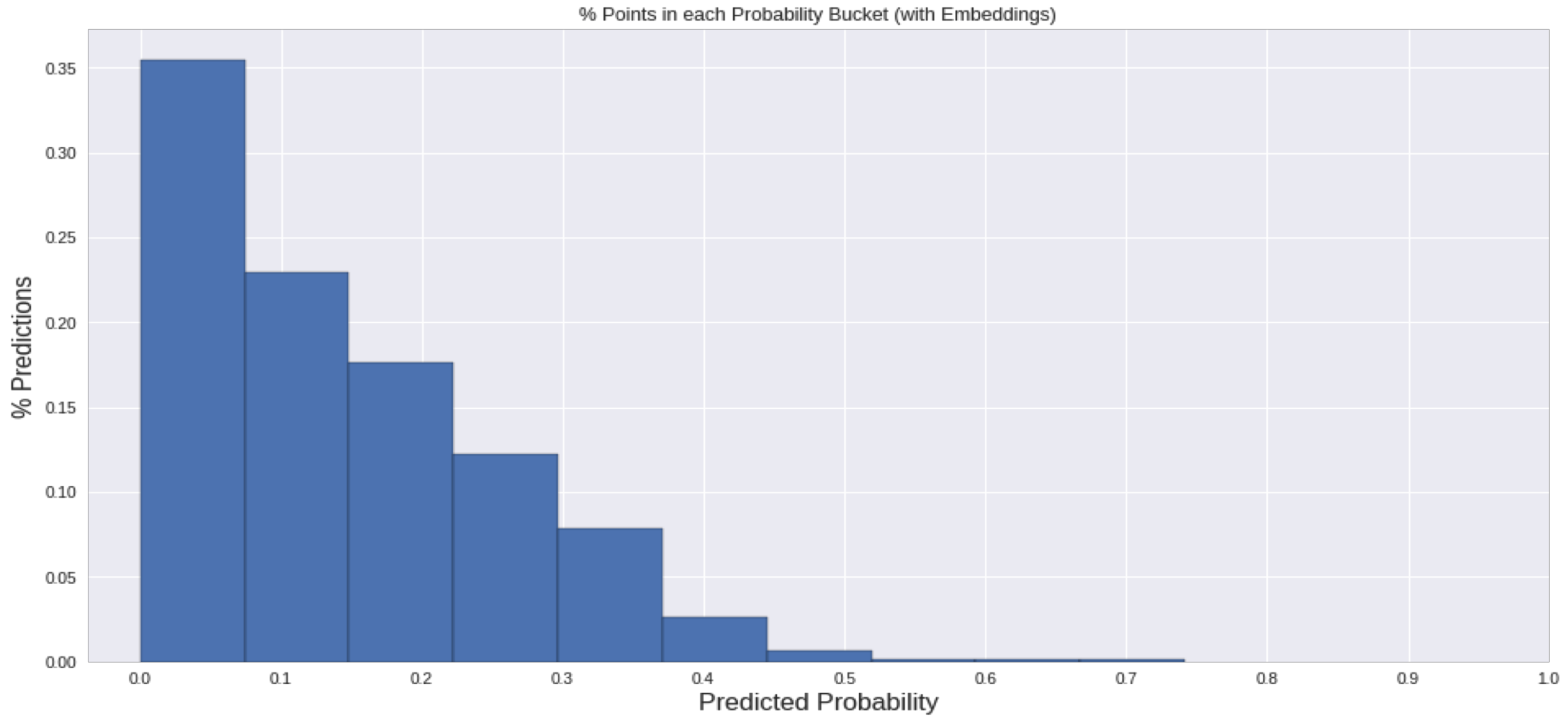
DNN – Thresholds(Embedding)

- We may be interested in sacrificing some precision to get higher recall. Testing out different thresholds for classification

Threshold	Accuracy	F1	Precision	Recall
0.1	0.524	0.312	0.194	0.794
0.2	0.719	0.340	0.250	0.529
0.3	0.824	0.293	0.323	0.268
0.4	0.862	0.135	0.473	0.079
0.5	0.866	0.054	0.673	0.028
0.6	0.865	0.034	0.750	0.018
0.7	0.864	0.007	0.833	0.004
0.8	0.864	0.000	0.000	0.000
0.9	0.864	0.000	0.000	0.000

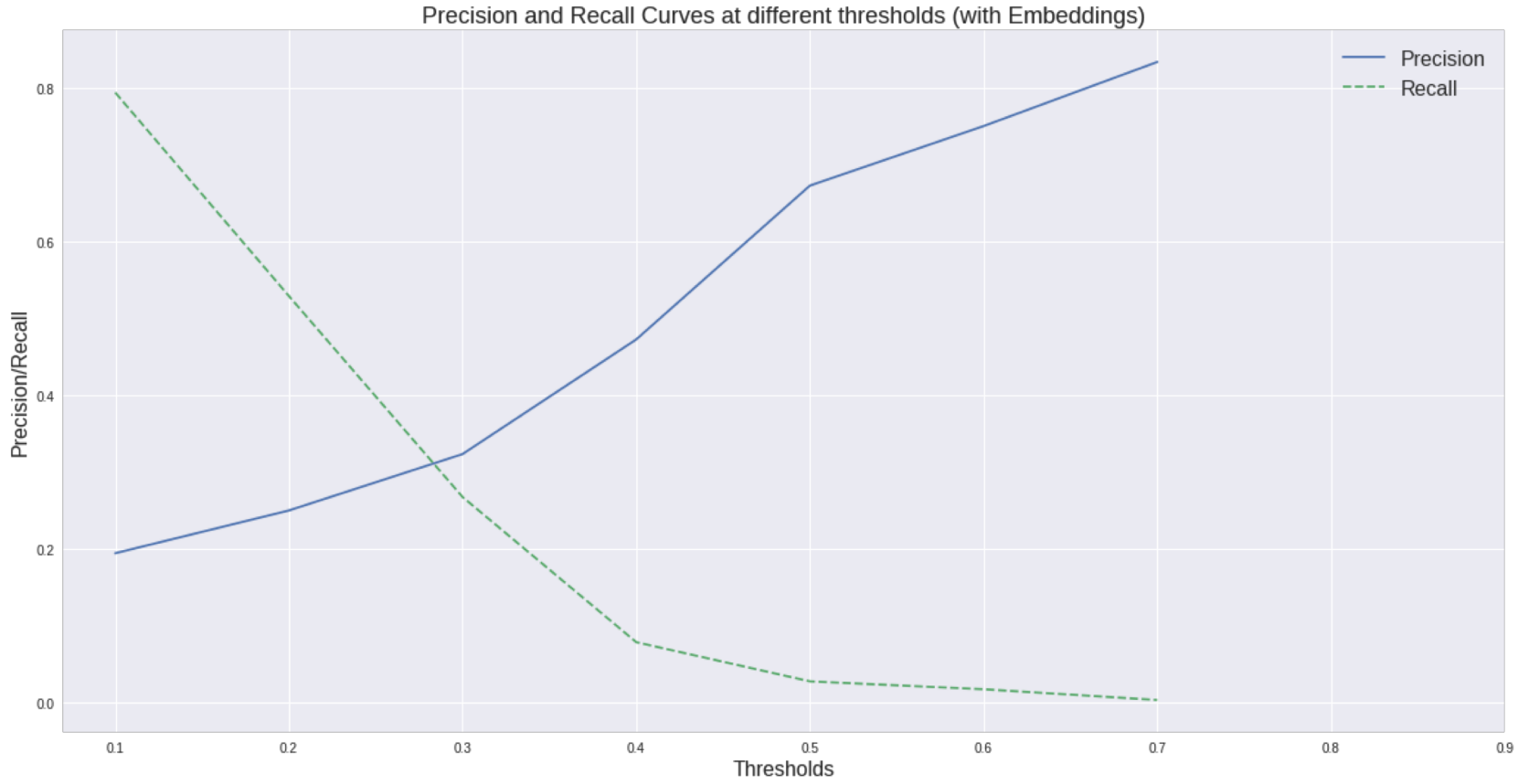
DNN (Embedding) – Precision/Recall

- Similar to the RF model, the DNN model also predicts a very small number of 1s resulting a low recall rate



DNN (Embedding) – Precision/Recall

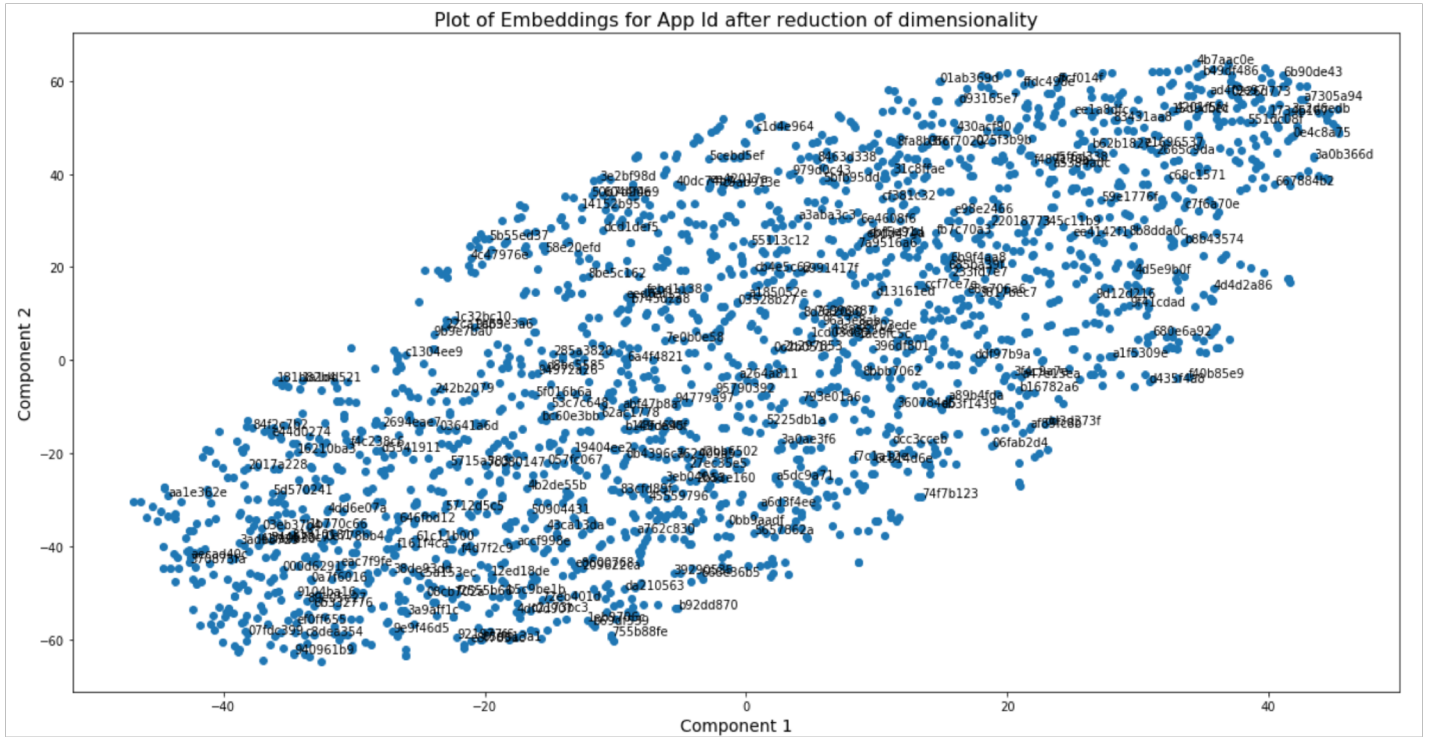
- A threshold of around 0.3 seems to provide a good balance between precision and recall rates. Given this is an imbalanced dataset with ~14% of 1s, a good balance for a useful model seems to be to get about ~33% precision and ~27% recall and the metrics are slightly better than the model without Embedding



Note: The threshold of 0.8 & 0.9 results in no prediction of 1s and hence precision and recall are 0. These points have been removed from the chart

DNN (Embedding) – Visualization

- The App ID was embedded into an 8 dimensional vector by the Embedding Layer
- A TSNE model was used to reduce dimensionality to 2d for easy visualization
- We can see that the embeddings have a certain pattern to them. However, since the app ids are all just alphanumeric identifiers and we don't have the actual names, it's hard to understand the pattern better
- It will be interesting to understand the embedding patterns when we do have the mapping information



Note:
We have the embedding weights for multiple features but we have plotted just one feature here

Random points have been labeled with app ids in order to avoid clutter

Wide and Deep Model- Methodology

- 80/20 Train Test Split
- Use combination of Linear (Wide) and Deep models
- Wide Model : Memorize interactions with data. Features are implemented as is. One hot encoding and embeddings are not used
- Deep Model : Generalize learned interactions on new data. Features implemented either using one hot encoding or categorical embeddings
- Use different combinations of Activation Functions/Optimizers/Learning Rate/Epochs
- Find best network architecture based on minimizing Loss and Overfitting
- Report performance on Test Dataset

Wide and Deep – Estimator

```
wide_deep_model = tf.estimator.DNNLinearCombinedClassifier(
    # wide settings
    linear_feature_columns = linear_cols,
    linear_optimizer=tf.train.FtrlOptimizer(learning_rate=0.1,
                                            l1_regularization_strength=10.0,
                                            l2_regularization_strength=0.0),
    # deep settings
    dnn_feature_columns = deep_cols,
    dnn_hidden_units=[256, 256],
    dnn_dropout=0.5,
    dnn_activation_fn = tf.nn.relu,
    dnn_optimizer=tf.train.AdamOptimizer
  )
```

Wide and Deep - Results

- Best model parameters
  - Activation Function: Relu| Optimizer: Adam| Loss: Binary Crossentropy
- General Observation
  - Gives comparable results to DNN
  - To truly leverage wide and deep capabilities, it needs high-cardinality features, where each feature has millions/billions of unique possible values
- Metrics based on the default 0.5 probability threshold
  - Accuracy: 0.865
  - Precision: 0.638
  - Recall: 0.033

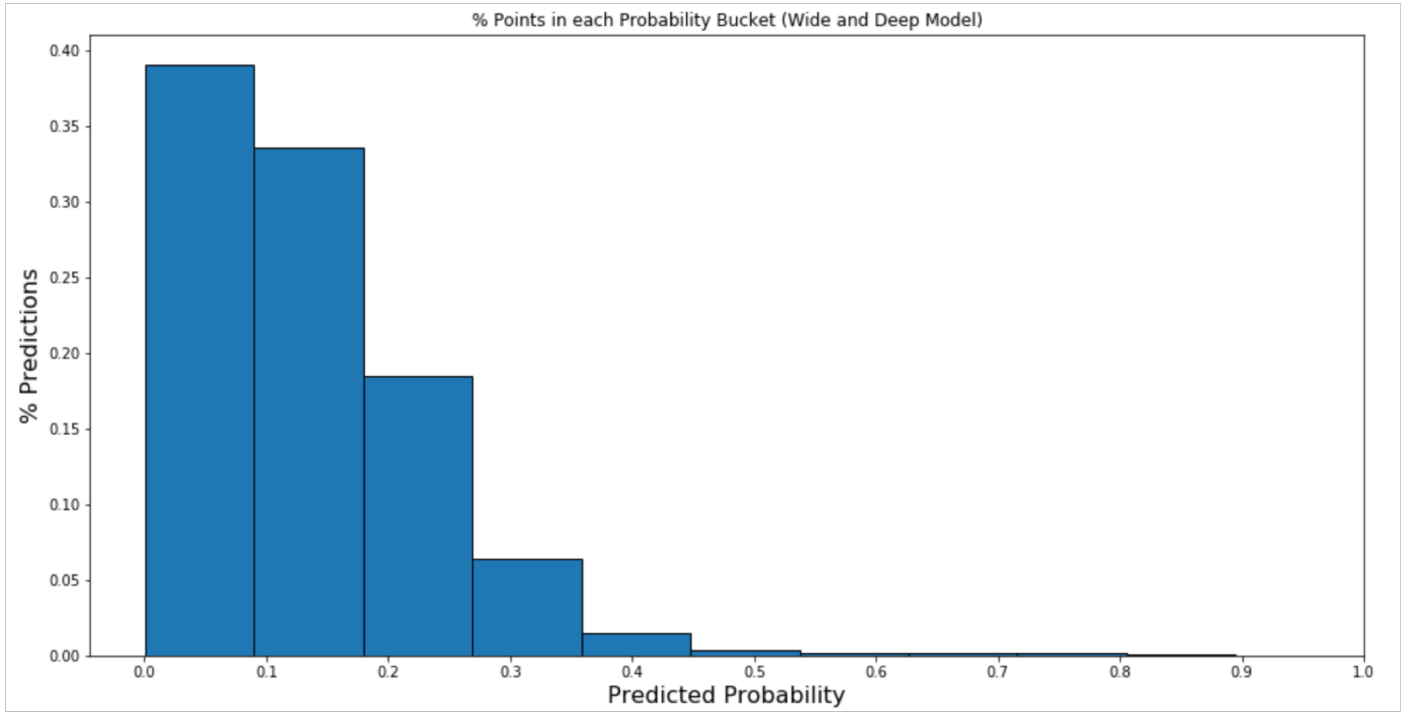
Wide and Deep – Thresholds

- We may be interested in sacrificing some precision to get higher recall. Testing out different thresholds for classification

Threshold	Accuracy	F1	Precision	Recall
0.1	0.526	0.323	0.201	0.827
0.2	0.770	0.357	0.289	0.467
0.3	0.850	0.234	0.388	0.167
0.4	0.864	0.103	0.534	0.057
0.5	0.865	0.063	0.638	0.033
0.6	0.865	0.047	0.713	0.025
0.7	0.865	0.028	0.771	0.014
0.8	0.864	0.010	0.855	0.005
0.9	0.863	0.000	0.000	0.000

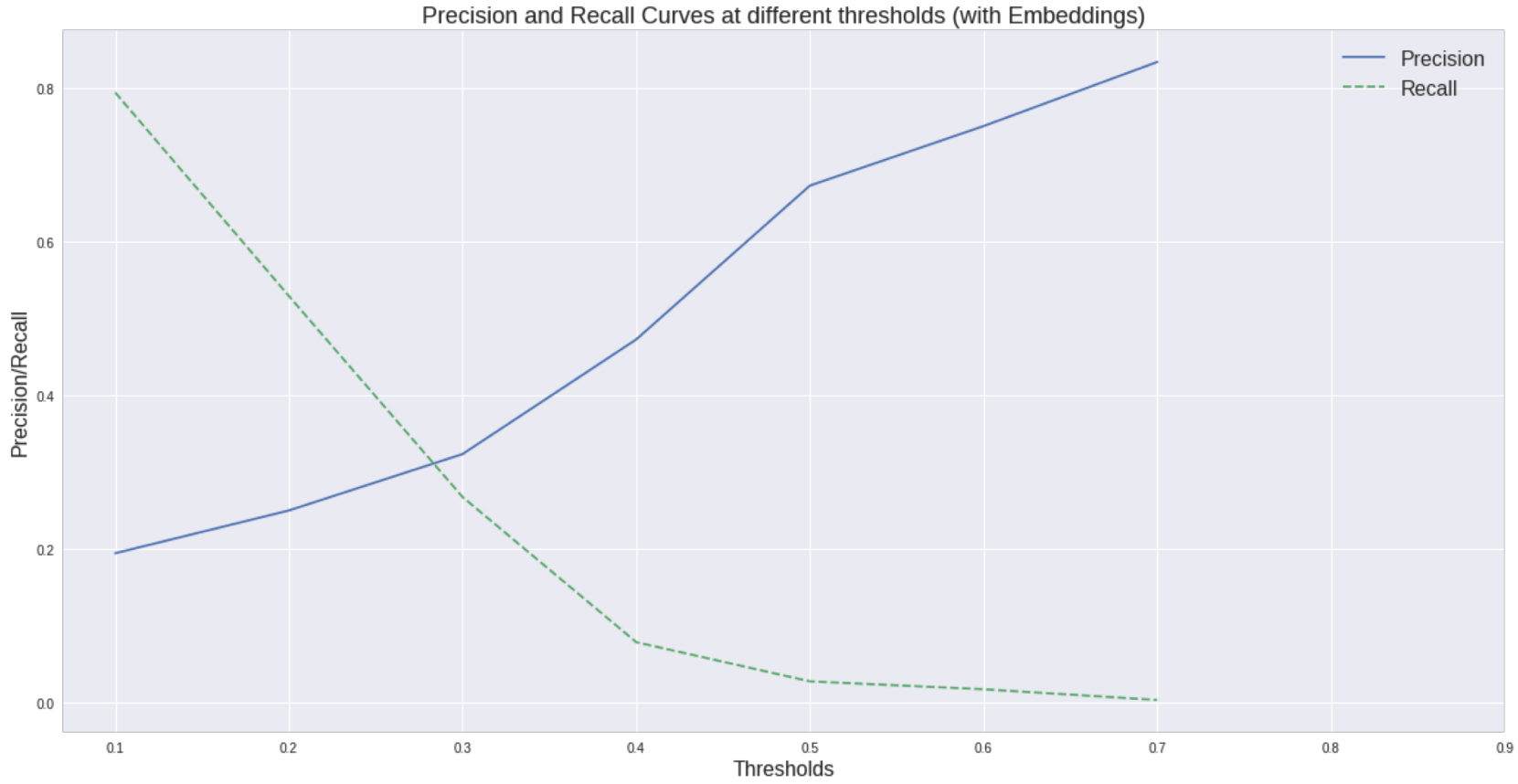
Wide and Deep – Precision/Recall

- Similar to the RF model, the Wide And Deep model also predicts a very small number of 1s resulting a low recall rate



Wide and Deep – Precision/Recall

- A threshold of around 0.26 seems to provide a good balance between precision and recall rates. Given this is an imbalanced dataset with ~14% of 1s, a good balance for a useful model seems to be to get about ~35% precision and ~27% recall and the metrics are slightly better than the other DNN models we tried



Note: The threshold of 0.8 & 0.9 results in no prediction of 1s and hence precision and recall are 0. These points have been removed from the chart

Model Comparison

Metric for the chosen model	Random Forest	DNN	Wide and Deep
Precision	~30%	~33%	~35%
Recall	~20%	~27%	~27%
Feature Importance	Standard Method	No standard method	No standard method
Data Preparation Time	High	Low	Low

- Random Forests has the added advantage of being interpretable
  - Feature importance is readily available via scikit-learn's API
  - There are methods to understand Feature Importance with DL models but they are not straight forward. One approach is to scramble the values of one feature and check increase in loss and rank features by the increase in loss when each variable is scrambled
- In our dataset, DL models performed better than RF models. We can spend more time on data preparation for RF models to eke out better performance but DL models do not need a lot of data preparation to get comparable performance.
- Overall, the wide and deep model had the best performance. But the comparison might be interesting if we focus more on the data preparation for the RF model

Notebook Execution and Future Exploration

- Data File (avazu_sample.gz) has been uploaded on Courseworks. This file needs to be available on the Colab server for the notebook to run end-to-end
- Random Forest Saved Model File is 170MB in size – so we have not uploaded it to Courseworks. But the model runs end to end with just the source file

Future Exploration:

- We want to try to use the embeddings as features in traditional ML algorithms (Random Forests or Boosted Trees) to check performance
- We want to spend more time on feature creation to see if it boosts performance in any of the models we tried above
- We want to try Deep Factorization Machines on this dataset to take advantage of Recommender System Methods along with Deep Learning

Acknowledgements

- We would like to express our sincere gratitude to Prof. Joshua Gordon for the guidance on areas to explore during our project
- We also want to thank the Professor and the TAs for helping through the semester
- We learned a lot of the application of DL for structured data during out project and we are sure it will come in handy at work in real world applications