# W261 - CTR Prediction


Group 31

Prabhat Tripathi | Section 2

Suhas Gupta | Section 6

Neha Kumar | Section 6


https://github.com/UCB-w261/f19-final-project-f19-team-31

# Context

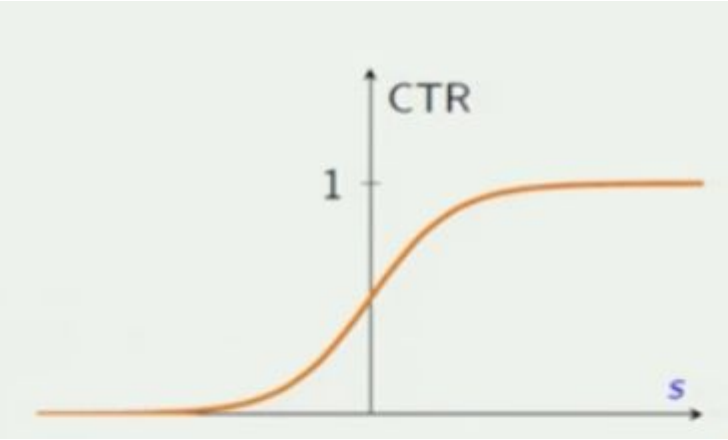
**Problem:** Predicting display ad click-through rate (CTR) probs

**Data:**

- Large dataset (over 50M+, 13GB+ train+test dataset)
  - 45M rows in training, 6M in test dataset
- 13 numeric and 26 categorical features.
  - No semantic information.
- Categorical features hashed to anonymize & regulate size of the variables
- Unbalanced labels
- Large number of distinct features -> large OHE vector ( > 33M)
  - Sparse feature matrix

# Logistic Regression with L2 Regularization

- Large-scale parallel linear learning algorithm
- Scales well to large number of features
- Sigmoid (logit) -> CTR prob
- Cost: Negative log-loss
- Gradient descent convex optimization



# Regularized Logistic Regression

### Loss Function

$$J(w) = \frac{-1}{N}\Sigma_{i=1}^{N} y_i log p_i + (1 - y_i) log(1 - p_i) + \lambda\frac{1}{2}\Sigma w^2$$

### Gradient Descent Step

$$\delta = \frac{\delta L}{\delta w} = \frac{1}{N}\Sigma_{i=1}^{N}(p_i - y_i)x_i$$

$$\delta_L = \frac{\delta L}{\delta w} = \frac{1}{N}\Sigma_{i=1}^{N}(p_i - y_i)x_i + \lambda\Sigma w$$

$$w_0 = w_0 - \alpha * \delta$$

$$w_j = w_j - \alpha * \delta_L$$

$$j = 1, 2, 3, ..$$

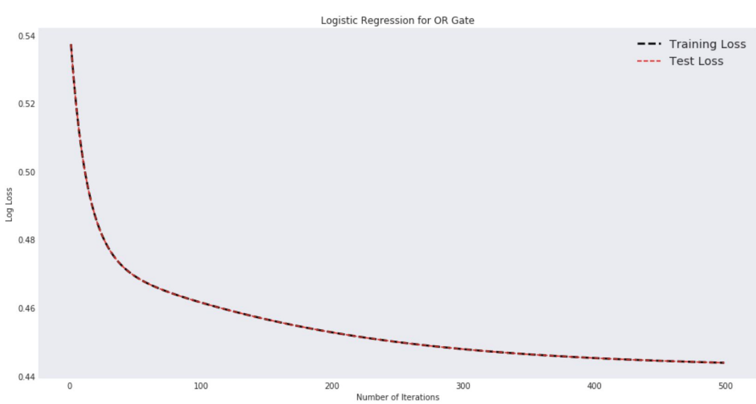
### Prediction (CTR)

$$sigmoid(t) = p_i = \frac{1}{1 + e^{-t}}$$

t : linear model

# W261 LogisticRegression

```
or_df = spark.createDataFrame(
    [(1, 1, 0, 1),
     (1, 1, 0, 0),
     (1, 1, 1, 1),
     (0, 0, 0, 0),
     (1, 0, 1, 1),
     (1, 0, 1, 0),
     (1, 0, 0, 1),
     (1, 1, 1, 0),
     ],
    ["label", "I1", "I2", "I3"])
```




[Final Notebook]()

## Homegrown Spark RDD Implementation

```
print("Homegrown Logistic Regression results")
print("\n------------------------------------------")
print("Coefficients: {}".format(models[-1][1:]))
print("Intercept: {}".format(models[-1][0]))
print("------------------------------------------")
```

```
Homegrown Logistic Regression results

------------------------------------------
Coefficients: [0.49895528 0.49922008 0.49930404]
Intercept: 1.257638686726761
------------------------------------------
```

## Spark ML

```
print("Spark ML results")
print("\n------------------------------------------")
print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))
print("\n------------------------------------------")
```

```
Spark ML results

------------------------------------------
Coefficients: [0.49353282070393717,0.4935328207039572,0.49353282070393195]
Intercept: 1.2730026935697145

------------------------------------------
```

# Homegrown Vs Spark ML Model

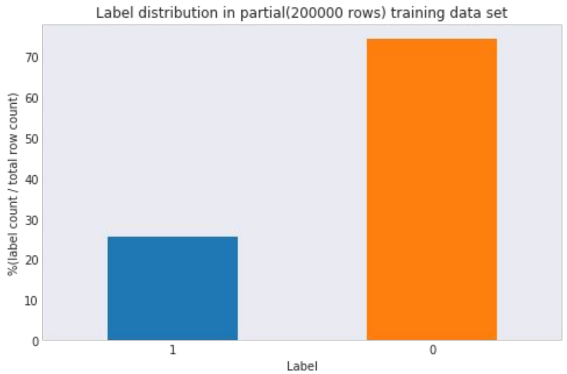
- Functionally equivalent
- Spark ML -> Better learning throughput
    - Dataframe Vs rdd?

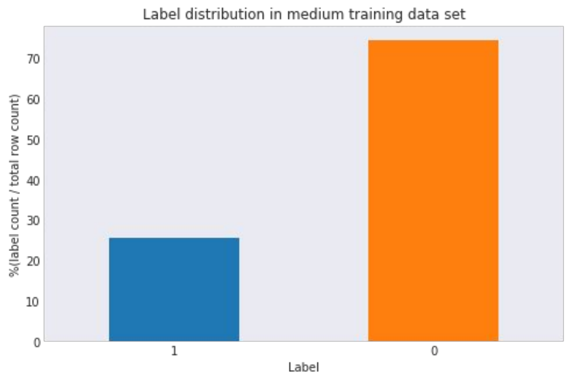
**Takeway: We will use Spark ML model for large dataset in this project**

# Click Through Rate

- Click through rate (Label) is the outcome variable we are trying to predict
- The overall click through rate is **25.62%**
  - Most of the web ad links are not clicked by surfers
  - **Smaller sampled datasets have the same Label=0 to Label=1 ratio**


*Distribution in percentage of total rows (45M) in the dataset*



# Numerical Feature Scale

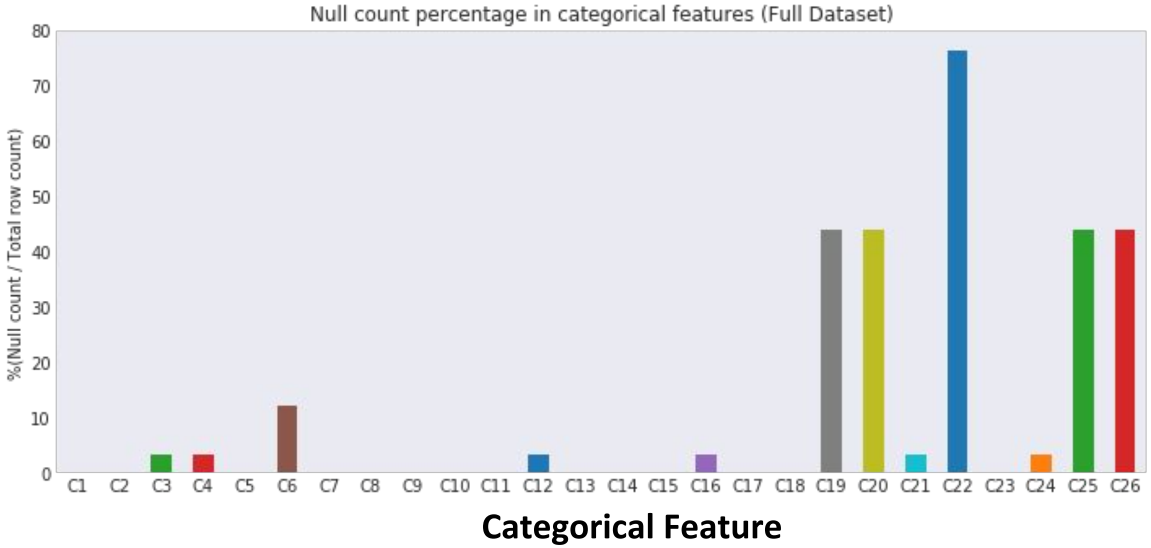
- For numerical features min-max scaling was used
  - **Allows gradient descent to converge**

| | I1 | I2 | I3 | I4 | I5 | I6 | I7 | I8 | I9 | I10 | I11 | I12 | I13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Min** | 0 | -3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Max** | 5775 | 257675 | 65535 | 969 | 23159456 | 431037 | 5631 1 | 6047 | 29019 | 11 | 231 | 4008 | 7393 |

$$Rescaled(e_i) = \frac{e_i - E_{min}}{E_{max} - E_{min}} * (max - min) + min$$

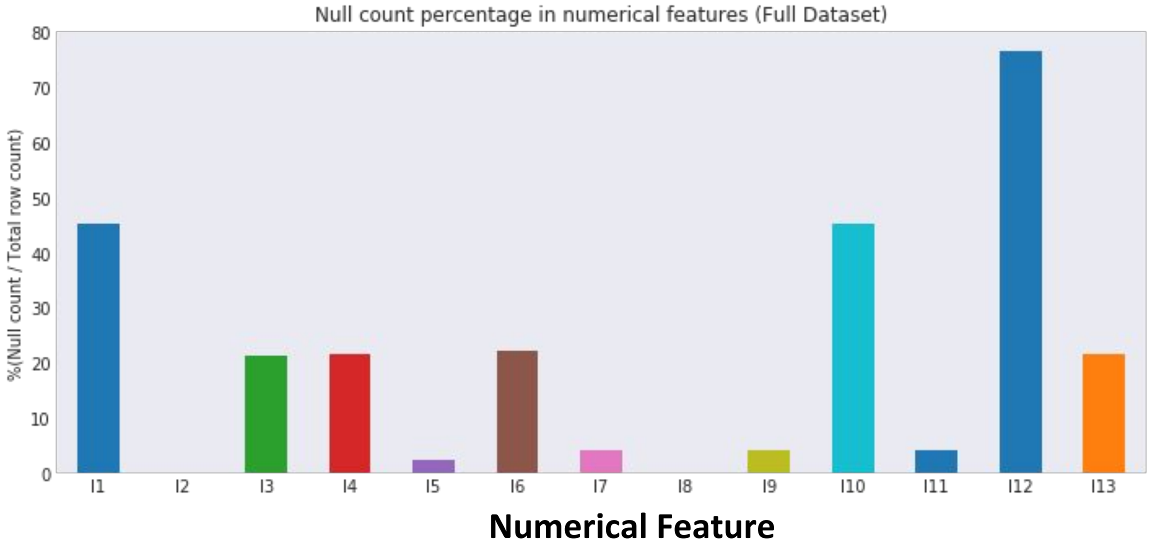
Spark ML Implementation used for the MinMaxScaler

# Handling NULL Values




Replace NULL with 0xFFFFFFFF

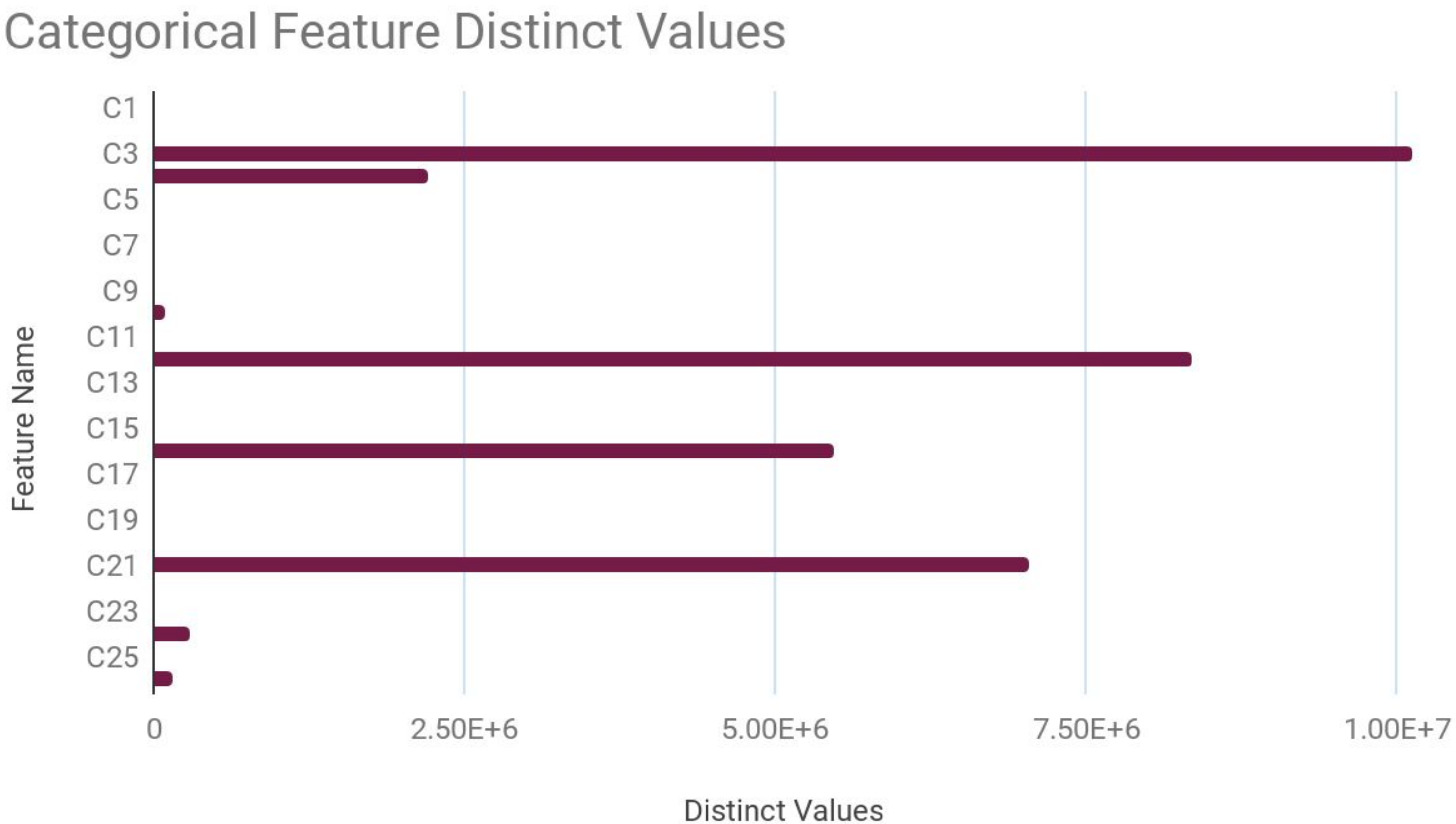
| Feature | C1 | C2 | C3 |
|---|---|---|---|
| Before | “” | “” | 0x7e0ccccf |
| After | 0xffffffff | 0xffffffff | 0x7e0ccccf |




Replace NULL with column median

| Feature | I1 | I2 | I3 |
|---|---|---|---|
| Before | null | 5 | 3 |
| After | 1 | 5 | 3 |

# One Hot Encoding: A Scalability Challenge

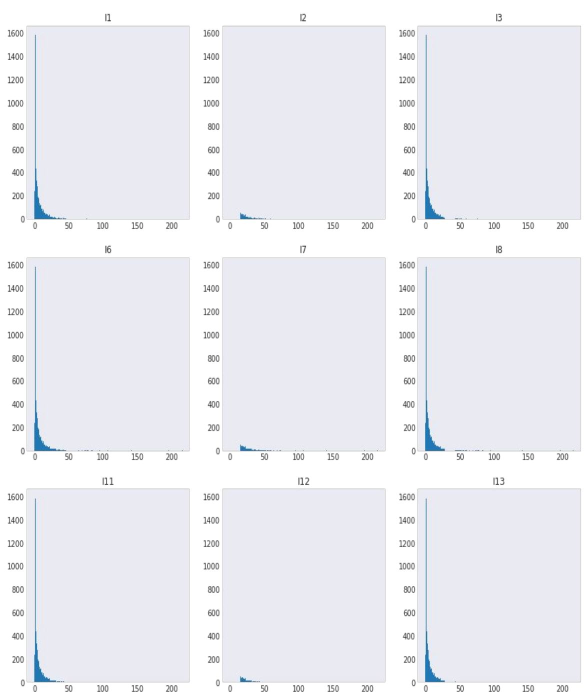



| Count of one-hot encoded features | |
|---|---|
| Pre-Binning | Post-Binning |
| 33,762,577 | 19,889 |

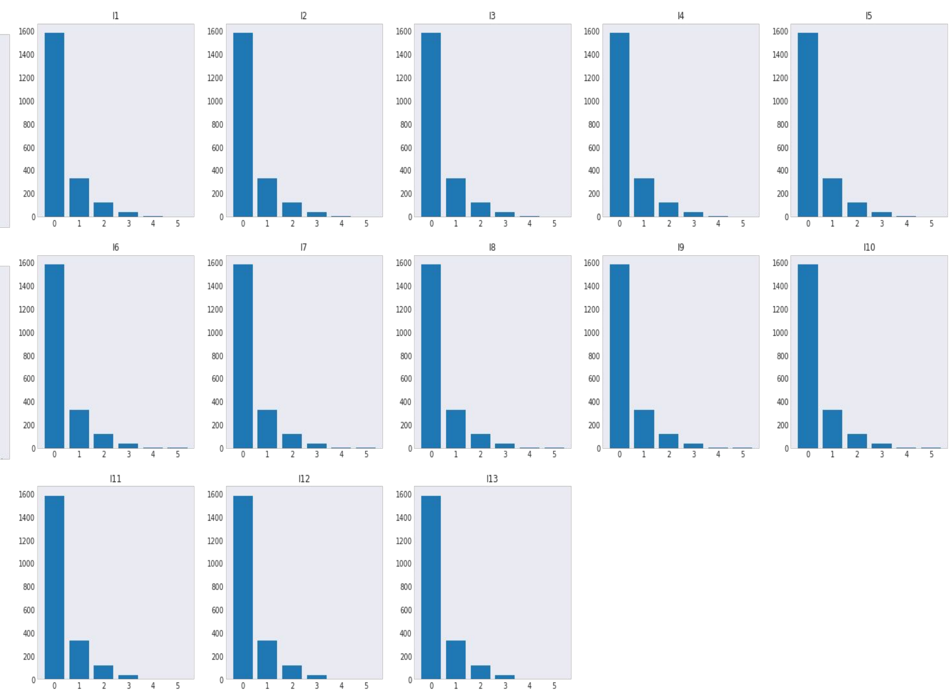
More than 1500x reduction!

# Numerical Feature Normality

- Applying CLT requires *approximately* normal distribution of linear independent variables
- **Log transformation of numerical features removes extreme skew**




Before Log Transformation

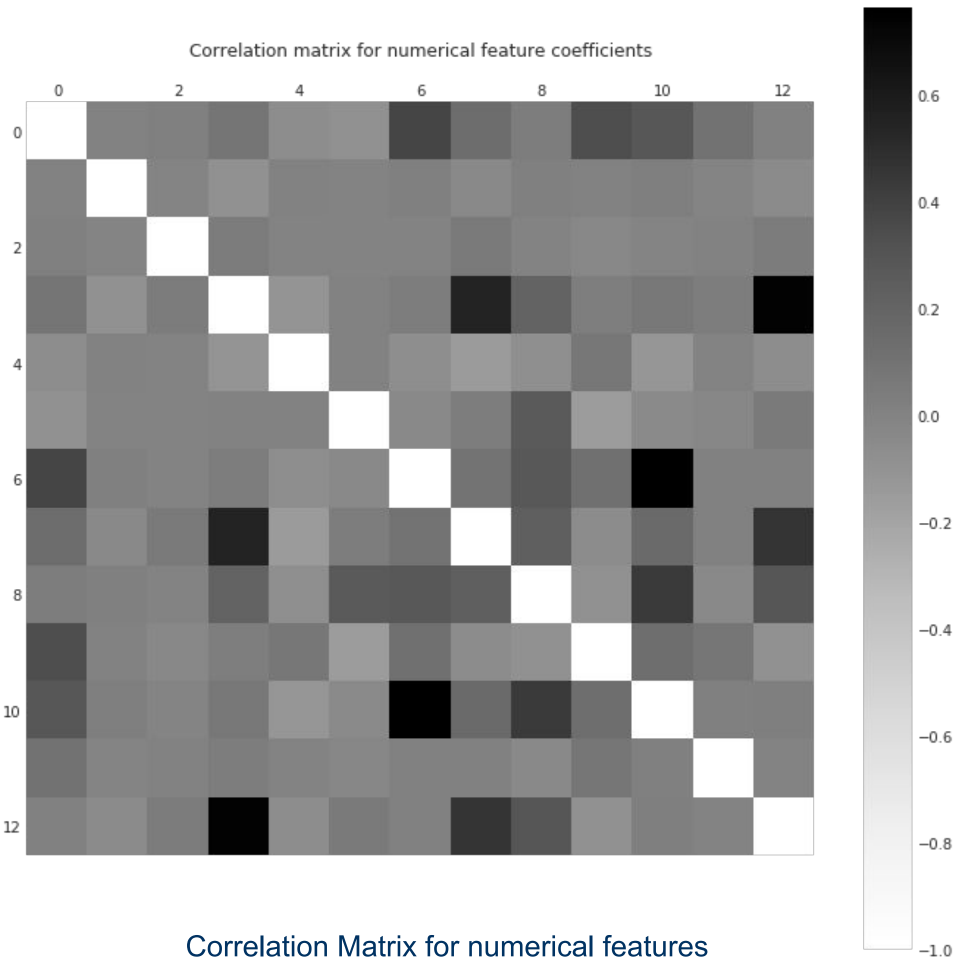



After Log Transformation

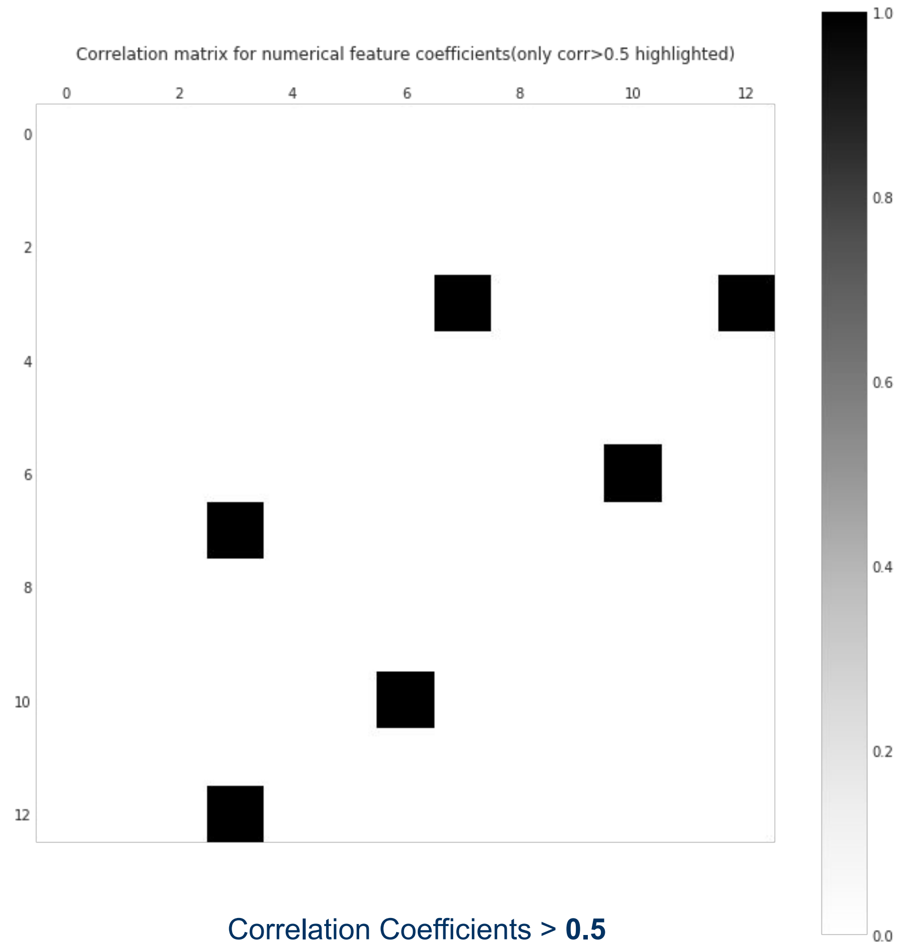
**Numerical Feature Value Distribution**

# Removing Multicollinearity

- **No perfect multicollinearity is a required assumption for OLS coefficients to be BLUE**
- Threshold for dropping collinear features is a hyperparameter in our model



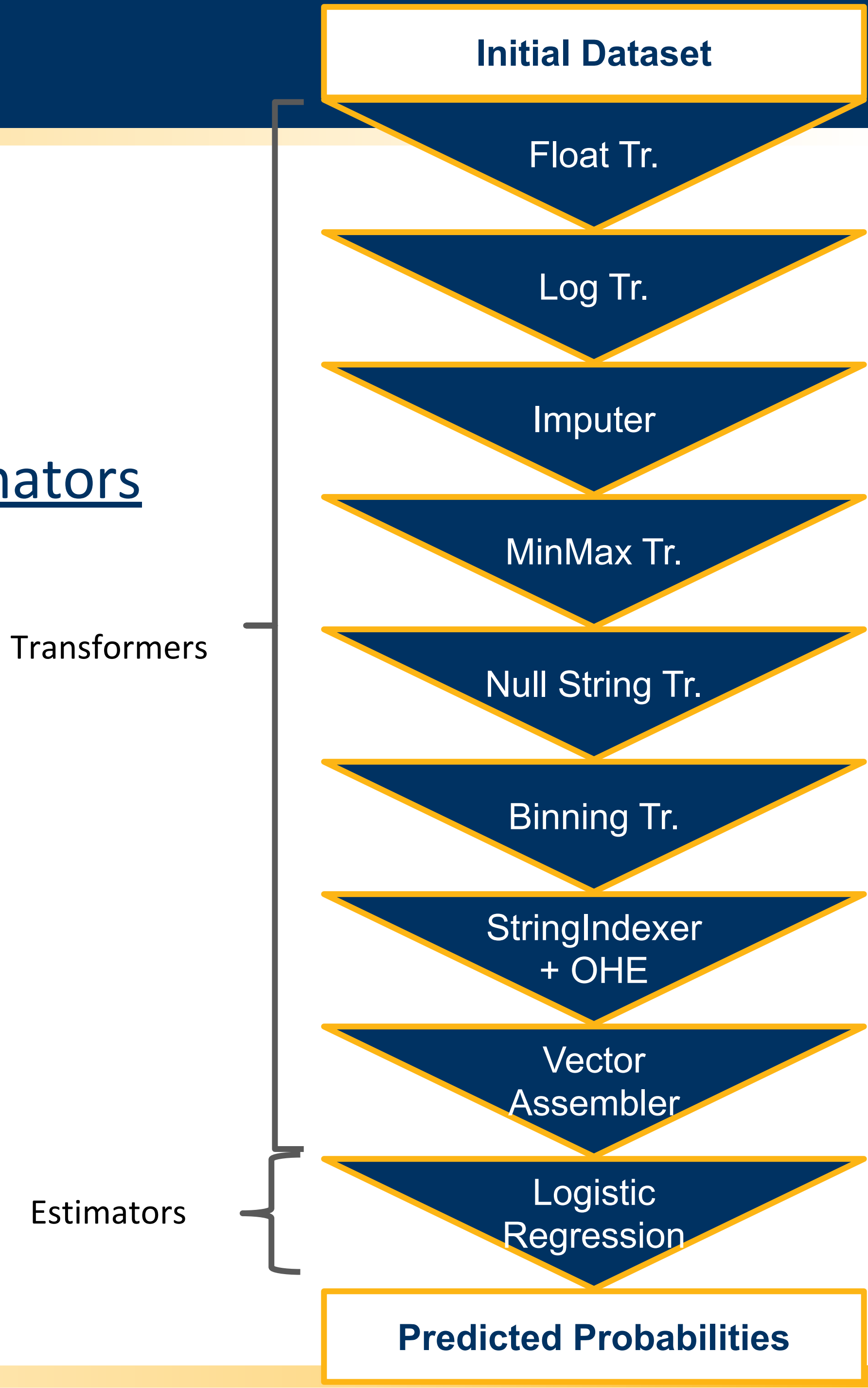

Correlation Matrix for numerical features




Correlation Coefficients > **0.5**

# Pipeline

- Spark ML pipelines facilitate easy iteration
- Consist of transformers and estimators



# Pipeline

- Logistic Regression requires all the features to be stored as a vector in a single column
  - Use: Vector Assembler to collapse columns
  - Omitting features = do not include them in the vector assembler call

| A | B | C |
|---|---|---|
| 1 | 4 | 9 |
| 2 | 3 | 0 |
| 3 | 3 | 1 |

| features |
|---|
| [1, 4, 9] |
| [2, 3, 0] |
| [3, 3, 1] |

# Pipeline

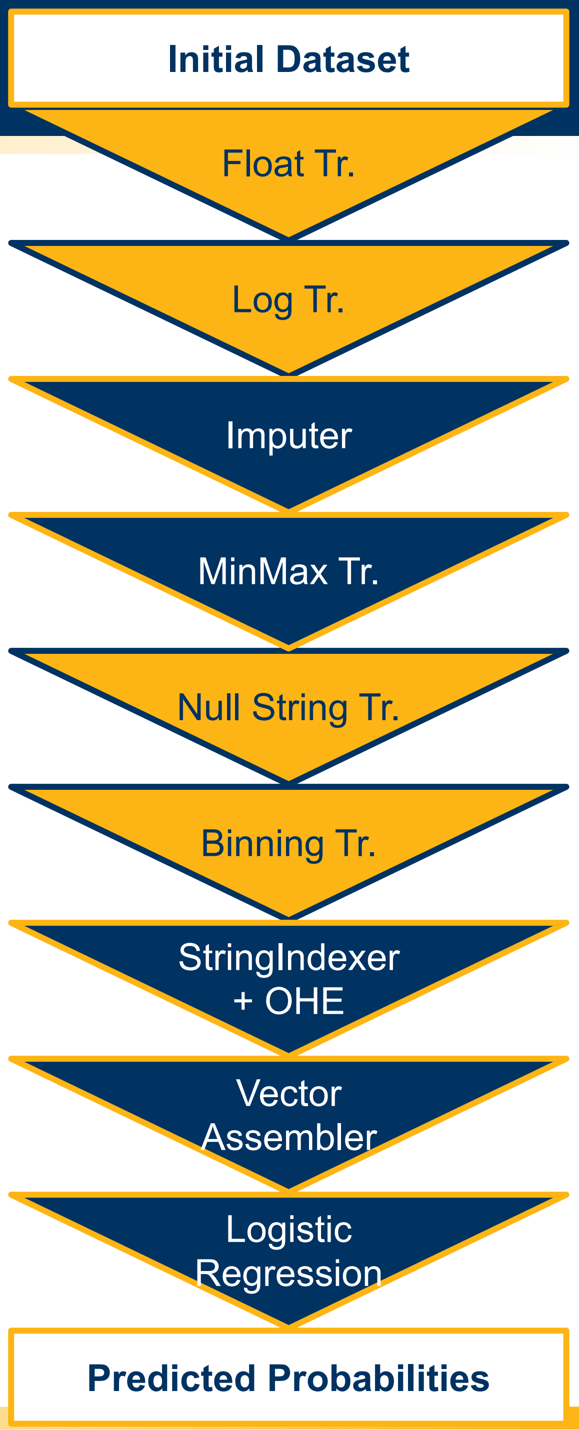
Other Built in methods utilized

- Imputer (fill in null values for numeric features with the median)
- MinMax scalar (rescale all the values from 0 to 1)
- StringIndexer + OneHotEncoder
    - HandleInvalid = ‘keep’ to bin unseen values

# Pipeline

Created custom classifiers

- Float Transformer: needed to use downstream transformers
- Log Transformer: Take log of numerical features to reduce skew
- Null String Transformer: Replacing blank strings with ‘fffffffff’ for easier viewing
- Binning Transformer: Bin categorical values by average y value



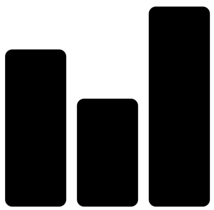
# More on Our Binning Approach

**For each categorical feature:**

**Phase A: Create Binning Lookup Tables**

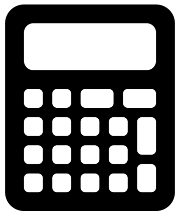
### Step 1

Count the number of distinct values per feature and determine which ones should be binned



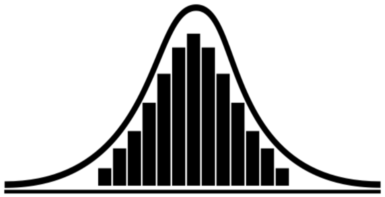
### Step 2

Take the average y value per distinct value



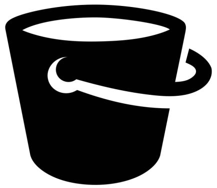
### Step 3

Log transform any results from Step 2 if the median is far below the mean



### Step 4

Round results to the tenths place. These are our new values



Images from the Noun Project

# More on Our Binning Approach

**For each categorical feature:**

## Phase A: Create Binning Lookup Tables

**Step 1**

Count the number of distinct values per feature and determine which ones should be binned

**Step 2**

Take the average y value per distinct value

**Step 3**

Log transform any results from Step 2 if the median is far below the mean

**Step 4**

Round results to the tenths place. These are our new values

## Phase B: Join Back to Original Dataset

**Step 5**

Create a list that contains all the lookup tables to loop over

**Step 6**

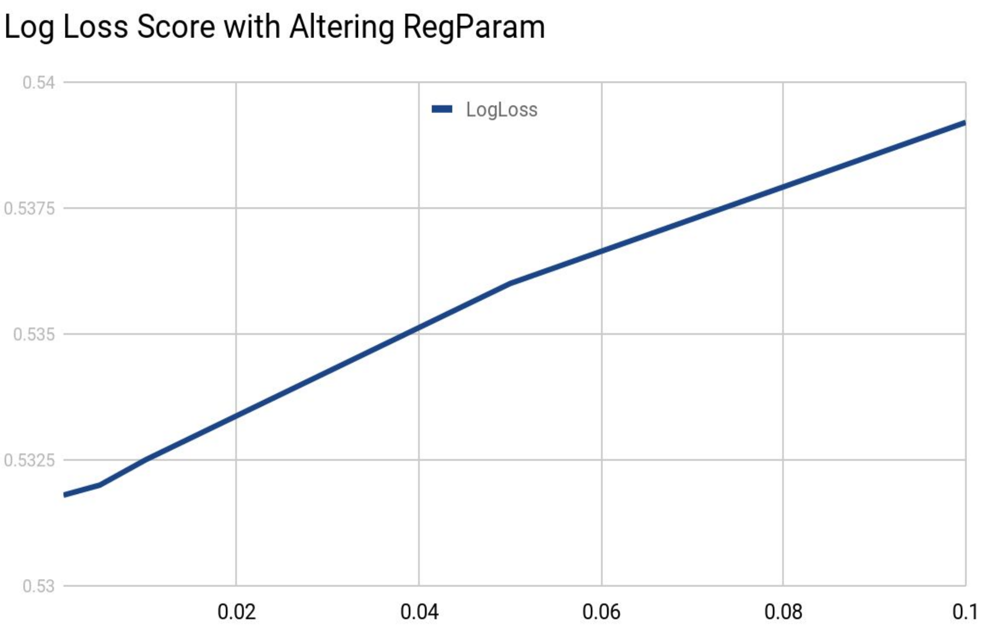
Run a mapper-side join, holding the binning lookup table in memory across all mappers

**Step 7**

Drop the old column with the original categorical values

# Conclusion / Final Model

- Finding the best regParam [0.001, 0.005, 0.01, 0.05, 0.1]
- Cutoff for dropping categorical variables and binning them instead (10k distinct values versus 100k)
- Logloss on training dataset = 0.4804
- Obtained probabilities for each record on the testing dataset for the purposes of this project
- Next step would be to upload to Kaggle to get the logloss score of the test dataset




**14 minutes 24 seconds**
(8 nodes: n1-standard-8)

# Confusion Matrix & Accuracy Measurement

| | TP | TN |
|---|---|---|
| Predicted Positive | 3246587 | 1848263 |
| Predicted Negative | 8498851 | 32246916 |

| Measure | Value |
|---|---|
| **Sensitivity** | 0.2764 |
| **Specificity** | 0.9458 |
| **Precision** | 0.6372 |
| **Negative Predictive Value** | 0.7914 |
| **False Positive Rate** | 0.0542 |
| **False Discovery Rate** | 0.3628 |
| **False Negative Rate** | 0.7236 |
| **Accuracy** | 0.7743 |
| **F1 Score** | 0.3856 |
| **Matthews Correlation Coefficient** | 0.3086 |

Generated from
http://onlineconfusionmatrix.com/

# Future Work

1. Prevent Overfitting
    a. Cross Validation
    b. Using a Dev dataset
2. Branching out to other classifiers
    a. Decision Trees (No linearly separable assumption)
3. More Feature Engineering
    a. Adding a row number feature to reflect that the data was chronologically ordered

# Thank you!

# Appendix

# Scalable Object Oriented Infrastructure

- Two classes define the functionality for all data processing, EDA & feature engineering
  - Allows for flexible usage of methods throughout the project
  - Ensures column oriented database is used in EDA and feature engineering