IN-CLASS DATA SCIENCE COMPETITION REPORT (5TH PLACE SOLUTION)

Rishabh Sehgal rs56738, EE460J; Kaggle: NeuRish95(sorry made the account long back)

1. ACKNOWLEDGEMENT

The author would like to acknowledge the teaching team of EE460J for organizing this in-class Kaggle competition to practice our machine learning and data-science skills acquired in the class. The author is deeply grateful to Prof. Alex Dimakis for his constant support through amazing lectures, well-thought out lab assignments and constant availability in office-hours. He is also thankful of all the TA's of the course, Ethan and George, for their continuous support and help during their office hours which cleared a lot of confusions and doubts regarding certain concepts and techniques.

2. EXECUTIVE SUMMARY

As per the in-class announcement the data is majorly of categorical nature and since a binary classification is the objective, the author examined and studied couple of other Kaggle competitions to develop intuitions and "feel" for the challenge. The EE460J competition data comprises HIPPA compliant medical data where the patient's details are anonymized by assigning them to numeric identifiers.

The final solution is based on an approach where multiple models were trained on 3 different kinds of datasets. The models used were XGBoostClassifier, LightGBM, Random Forests and CatBoost. CatBoost was specifically used for its very lucrative features with respect to the competition data, and its state-of-the-art libraries and fast-GPU training. CatBoost provides much better categorical features support and has much better overfitting handling as compared to a finely tuned XGBoost model. These models were further stacked with a XGBoost as a meta-regressor. Finally, the stacked model was trained and blended with the other trained models to ensure the robustness of the solution designed when the model sees an unknown dataset portion in the private LB score calculation.

3 different kinds of datasets were created after input data processing and those different datasets worked well with different models providing the author with insights. Eg. The CatBoost stacked model and the standalone CatBoost model worked very well when only frequency encoding was performed on the 24 features. The model actually dropped in performance slightly when one-hot encoding was performed before fitting and Cross Validation. This was completely opposite of XGBoost which absolutely required One hot encoding to perform decently on the dataset. Later more transforms and supervised/unsupervised categorical encodings were applied on the dataset to make sure that XGBoost gives cross-val AUC scores near low 90s. (i.e. 0.91- 0.93). The final solution was a blend of these three models, a) CatBoost trained on the dataset encoded with SVD and frequency encoding (unsupervised); b) XGBoost tuned and trained on dataset encoded with both unsupervised and supervised encoding techniques such as Target Encoding with Smoothing and also adding noise within the encoding. The noise part is important for the powerful classifiers such as LGBM and XGBM as without it, the model simply learns/memorizes the features and drops a lot when repeated stratified KFold Cross Validation is performed on it. The detailed report follows in the subsequent sections.

Final model is a **Stacked version** of **CatBoost + XGBoost + LGBM with XGBoost** as the meta-classifier (Stack_model). The **final prediction was a blend** of 0.35 Stack_model, 0.5 CatBoost model and 0.15 XGBoost

model. This whole combination performed slightly better than a standalone CatBoost trained on the Feature encoded dataset.

3. EDA And Feature Engineering

Understanding the data:

Data was loaded and analyzed, first by visual inspection and then by some standard Exploratory Data Analysis techniques. The main data was saved in a folder in the base path and that data was immutable meaning that any data processing and feature engineering was applied on the copied version of those dataFrames so as to preserve the original data from changing.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
import gc
import os

import matplotlib.pyplot as plt
from scipy.stats import skew
from scipy.stats.stats import pearsonr

%config InlineBackend.figure_format = 'png' #set 'png' here when working on notebook
%matplotlib inline
n_jobs=-1
```

In [2]:
```
base_path = '/home/local/lambda/rishabhs/ML/data_science_lab/kagglecomp'
train = pd.read_csv("data/data-science-comp-f2020/train_final.csv")
test = pd.read_csv("data/data-science-comp-f2020/test_final.csv")
```

In [3]:
```
train.head()
```

Out[3]:

	Id	Y	f1	f2	f3	f4	f5	f6	f7	f8	...	f15	f16	f17	f18	f19	f20	f21	f22	f23	f24
0	1	1	25884	1	33.63	118596	1	0	118595	125738	...	1945	118450	119184	1	121372	1	1	1	2	1
1	2	1	34346	1	10.62	118041	1	0	117902	130913	...	15385	117945	292795	1	259173	1	1	1	1	1
2	3	1	34923	1	1.77	118327	1	0	117961	124402	...	7547	118933	290919	1	118784	1	1	1	1	1
3	4	1	80926	1	30.09	118300	1	0	117961	301218	...	4933	118458	118331	1	307024	1	1	1	2	1
4	5	1	4674	1	1.77	119921	1	0	119920	302830	...	13836	142145	4673	1	128230	1	1	1	620	1

5 rows × 26 columns

In [4]:
```
print(train.shape)
```

```
(16383, 26)
```

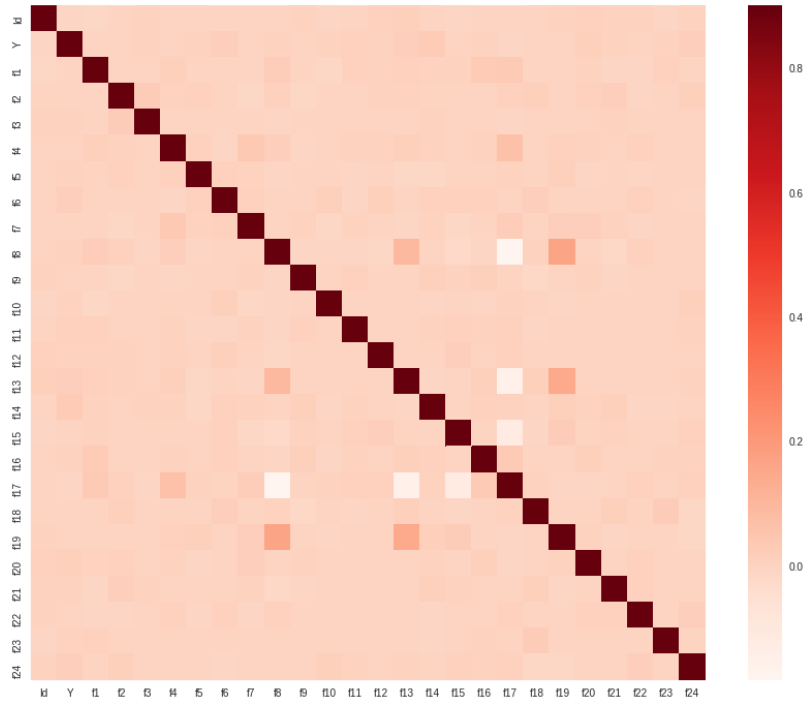
It was observed the test and training data are equal in size (num of samples) with the only change being that the training data contains the target value “Y”, which is the representative of final classes. The very first step required was to remove the column containing IDs from the training and testing data set. this was important to ensure that no feature containing ID is learned by the model.

A. Correlation Plot, Unique Values and Missing Data

The following correlation plot was obtained from the training data which showed the correlation between the features and features and the target value. As seen from the graph below features like after f13, f19, f8 are quite correlated to each other. But apart from these basic intuitions the correlation graph was not enough to provide any more insight into the data by itself. Since from the visual inspection it was clear that a lot of features are categorical data a list of the total number of unique values in each feature was calculated and certain inferences were drawn from that list. The list is also shown in the graph below.

From the below list it looks like apart from F-14 all other features do not have numbers of unique values in proportion to the number of total samples. Hence apart from F-14 all other features were considered to

be categorical features and treated accordingly. A basic check on missing data was performed which showed that none of the values in the features were missing hence no special treatment was required.

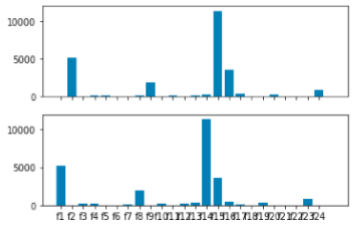


```
train.nunique()
#trainu = train.nunique()
```

```
Id     16383
Y          2
f1      5170
f2         7
f3       168
f4       162
f5         8
f6         9
f7       118
f8      1851
f9         7
f10      182
f11        7
f12      157
f13      322
f14    11349
f15     3555
f16      432
f17       67
f18        9
f19      322
f20        7
f21        8
f22        7
f23      906
f24        7
dtype: int64
```

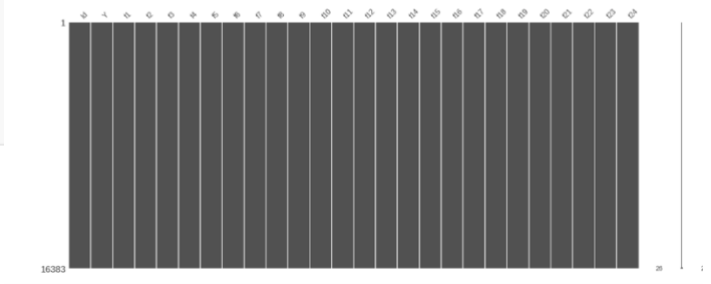
```
In [5]:
# Compare number of Unique Categorical labels for train and test

unique_train= pd.DataFrame([(col,train[col].nunique()) for col in train.columns],
                           columns=['Columns', 'Unique categories'])
unique_test=pd.DataFrame([(col,test[col].nunique()) for col in test.columns],
                         columns=['Columns', 'Unique categories'])
unique_train=unique_train[1:]
unique_test=unique_test[1:]

fig, ax = plt.subplots(2, 1, sharex=True, sharey=True)
ax[0].bar(unique_train.Columns, unique_train['Unique categories'])
ax[1].bar(unique_test.Columns, unique_test['Unique categories'])
#plt.xticks(rotation=90)
```

```
Out[5]: <BarContainer object of 24 artists>
```



```
[11]:
import missingno as msno
msno.matrix(train)
```

```
[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7ac06710d0>
```

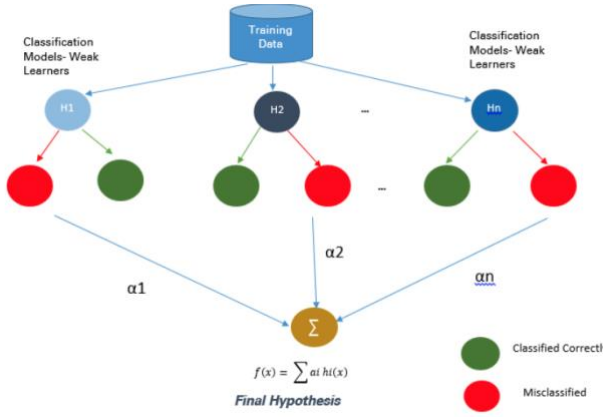


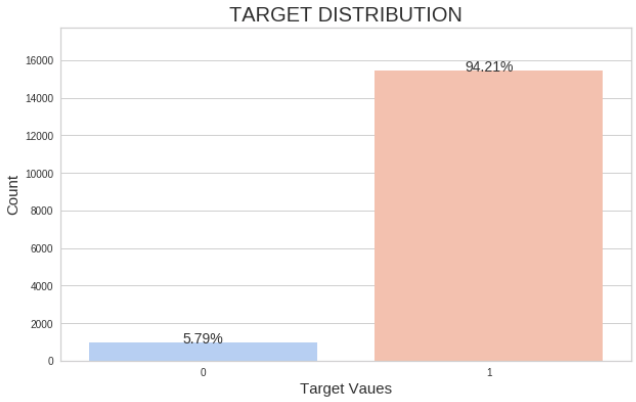
It was found out that only f3 and f14 have floating data and all other features have integer data further confirming that all other features are categorical in nature. A scatter plot of all the features with respect to “Y” prediction labels was also were also plotted but they yielded no solid intuition other than the fact that the number of classes samples with classification one what predominantly present as compared to the samples with final classification 0.

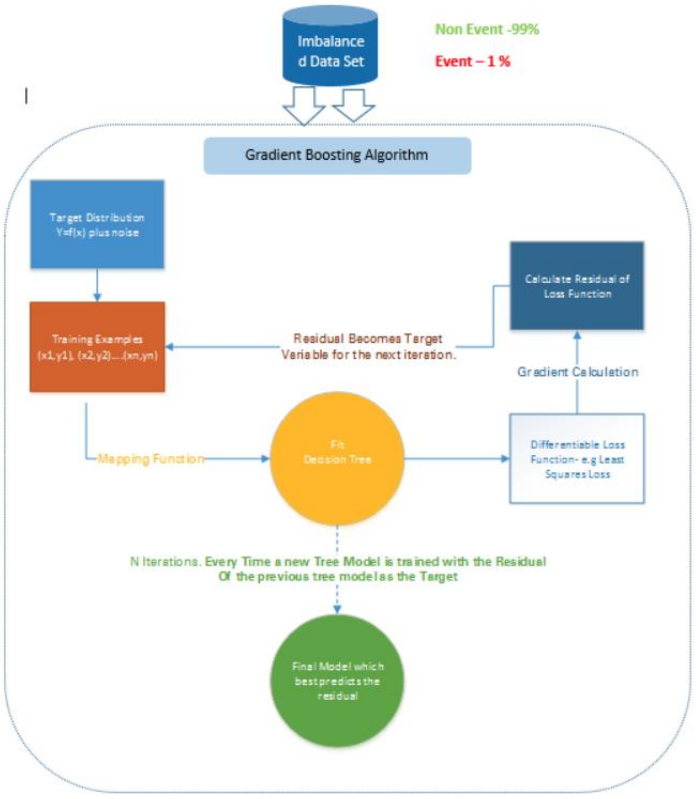
B. Target Distribution

As seen from the target distribution plots below, it can be easily inferred that the number of True samples far outweigh the number of false samples in the data. This kind of data is called an unbalanced data. The conventional model evaluation methods do not accurately measure model performance when faced with imbalanced datasets hence evaluation metrics such as AUC are used to properly evaluate the model performance whenever the data set of these kinds are used. It is imperative that while working with

imbalanced data we use ensembling methods so that multiple classifiers are trained separately, and their combined vote gives a final strong classifier. If we don't use the previously mentioned approach, we could also use techniques such as cluster-based oversampling all random oversampling to make sure that the imbalanced nature of data does not negatively impact the learning of the model. The author chose the ensembling technique to mitigate this issue. Also, one of the main reasons for using Boosting and Bagging based techniques was this imbalanced data, since gradient boosting itself is an ensemble technique where multiple weak learners are combined to create a strong learner for making accurate predictions.







Some final thoughts on EDA: It was also found out that there are many high cardinality features in the dataset and hence the different types of encodings were used for the high cardinality and the low cardinality features.

C. Feature Engineering and different types of encoding used.

Initially while working with the data no feature engineering techniques were used and an extra boost model was fitted and hyperparameter tuned and an AUC score of 0.89 was obtained with the said model. This was the case even when multiple models were stacked when the next re boost meta regressor and when they were trained, the model could not break the barrier of AUC score 0.92 in the public leaderboard. apart from the usage of cat boosting techniques one thing that helped the most in getting better score was using appropriate encoding and transformation techniques in the feature engineering for data set provided. The various techniques for feature engineering fall under two heads, Unsupervised and Supervised. As mentioned in the executive summary of this report, different models work well with different types of Encoding techniques.

Unsupervised Techniques: It was found out that certain unsupervised techniques such as the one hot encoding of the categorical data and frequency encoding go along way to improve the AUC scores on XGBoost and catBoost based models. The same technique was also applied on LGBM based models and it also helped their score a little bit. While one-hot encoding is easy to visualize, frequency encoding encodes the categorical features based on their counts. Techniques used were the following:

1. **Frequency Encoding**: The counting of how many times the unique value is present in the dataset, is performed. It is easy to implement and understand and it cannot be used if the categorical values are balanced, which is clearly not the case in our data set.
2. **SVD Encoding**: In this embedding we try to incorporate information about data interaction for example we take two columns say A and B and present them as 2D matrix where rows are unique values of column A and columns are unique values of column B. These techniques are very frequently used in the text-based data sets such as one of the datasets in our previous lab assignments. We can also apply IDF (inverse document transform) in order to assign bigger values rare pairs. Doing this causes a huge increase in number of features and hence dimensionality reduction techniques need to be applied otherwise the complexity and dimensionality of model will increase. The author has applied truncated SVD and the matrix is reduced into a vector. This vector is the embedding of column A based on column B. This process is repeated for all the columns in data set. The major advantage of this technique is that it provides rich and multi-dimensional representations. Though it's not very easy to understand and it's very unintuitive. it's easy to make mistakes if SVD is applied wrongly.

Supervised Techniques: The supervised techniques use the information about the target we are trying to predict in order to build our categorical embeddings. following supervised techniques were employed:

1. **Target Encoding with Smoothening**: Encode each value by target mean and for the unseen values use data set target mean. This technique is straightforward to implement but it introduces leakage. Leakage means use of extra information that the model would not have during prediction. This means that there will be drop in AUC score when unknown values of data set are seen by the model. This happens because the extra information overestimates the model's utility.

In [35]:

```
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

def extract_col_interaction(dataset, col1, col2, tfidf = True):
    data = dataset.groupby([col1])[col2].agg(lambda x: " ".join(list([str(y) for y in x])))
    if tfidf:
        vectorizer = TfidfVectorizer(tokenizer=lambda x: x.split(" "))
    else:
        vectorizer = CountVectorizer(tokenizer=lambda x: x.split(" "))

    data_X = vectorizer.fit_transform(data)
    dim_red = TruncatedSVD(n_components=1, random_state = 5115)
    data_X = dim_red.fit_transform(data_X)

    result = pd.DataFrame()
    result[col1] = data.index.values
    result[col1+"_{}_svd".format(col2)] = data_X.ravel()
    return result

import itertools
def get_col_interactions_svd(dataset, tfidf = True):
    new_dataset = pd.DataFrame()
    for col1,col2 in itertools.permutations(dataset.columns, 2):
        data = extract_col_interaction(dataset, col1,col2, tfidf)
        col_name = [x for x in data.columns if "svd" in x][0]
        new_dataset[col_name] = dataset[[col1]].merge(data, on = col1, how = 'left')[col_name]
    return new_dataset
```

In [38]:

```
new_train, new_test = transform_dataset(
    train[cat_cols], test[cat_cols],
    get_col_interactions_svd
)
print(new_train.shape, new_test.shape)
new_train.head(5)
```

```
(16383, 506) (16385, 506)
```

Out[38]:

	f1_f2_svd	f1_f3_svd	f1_f4_svd	f1_f5_svd	f1_f6_svd	f1_f7_svd	f1_f8_svd	f1_f9_svd	f1_f10_svd	f1_f11_svd	...	f24
0	0.978162	0.691295	0.504234	0.899624	0.785948	0.491524	0.002927	0.977465	0.833553	0.999653	...	
1	0.999674	0.020508	0.005027	0.999668	0.999633	0.015864	0.000132	0.999690	0.165411	0.999653	...	
2	0.999985	0.955762	0.119602	0.996644	0.989458	0.971643	0.824147	0.998050	0.924635	0.992672	...	
3	0.999674	0.212666	0.920981	0.999668	0.999633	0.999748	0.000483	0.999690	0.981989	0.999653	...	
4	0.995974	0.939793	0.580522	0.991605	0.997983	0.988238	0.917563	0.995034	0.949541	0.997082	...	

5 rows × 506 columns

To make sure that target encoding is more robust to leakage we address the main problem which causes leakage in the first place, low frequency values. The weighted sum of 2 means data set mean and level mean are taken 2 smoothen the target encoding. Level mean is the mean of a particular value in the feature set. This is calculated by the following formula. This does add to complexity of the data set.

Weighted sum:

$$f(n) * mean(level) + (1 - f(n)) * mean(dataset)$$

Weighting function:

$$f(x) = \frac{1}{1+exp(\frac{-(x-k)}{f})}$$

where,

k - inflection point, that's the point where $f(x)$ is equal 0.5

f - steepness, a value which controls how step is our function.

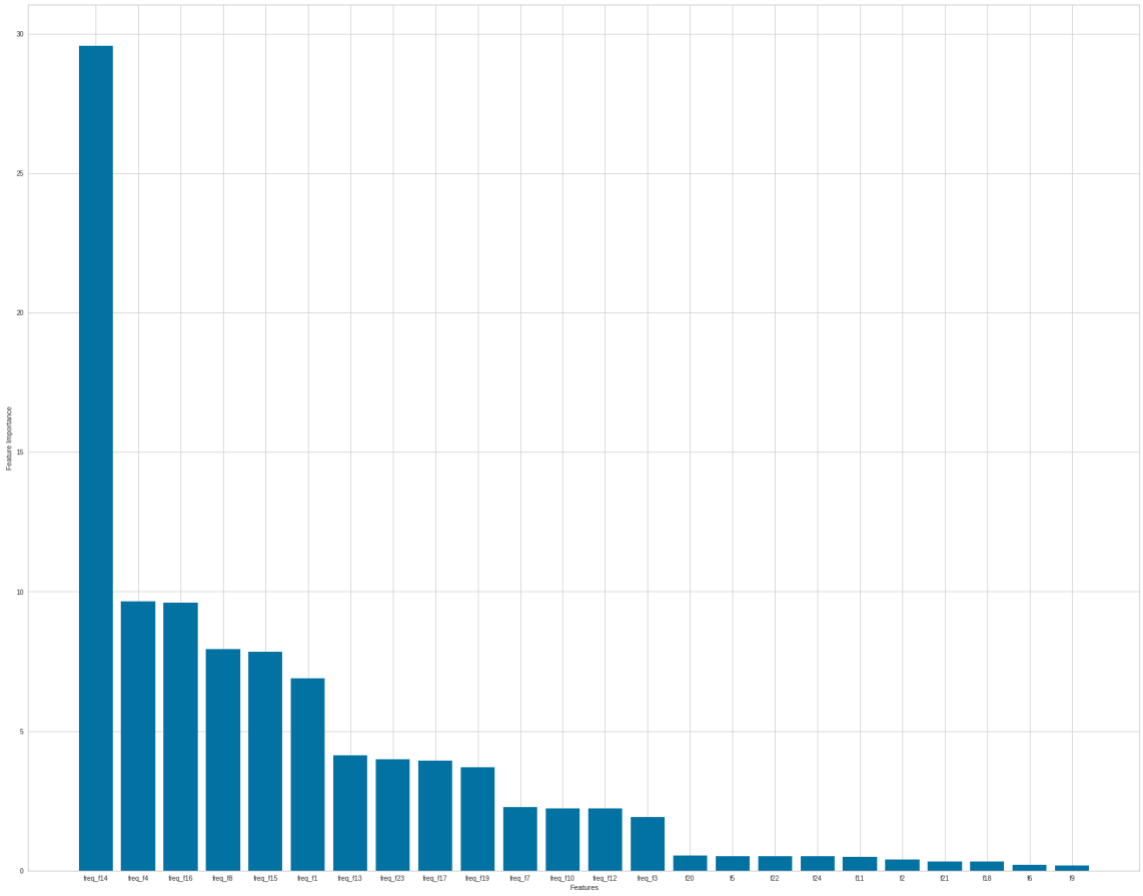
2. **Addition of noise using Expanding Mean:** One of the main motivations of adding noise in the embedded data is that the powerful models such as light GBM and XG boost should not be able to memorize the data and instead should be able to generalize and not cause overfitting. This technique was used by the author in the feature engineering. There are many different ways to implement this in code, the way that author followed was very intuitive and the code was readily available in Kaggle as it was used by a Kaggle master in one of their modelling/encoding efforts. This approach uses the streaming technique to introduce noise. For each new row it uses all the previously seen rows to calculate the new row mean. For the very first row the mean is data set mean and for the 2nd row the mean of only first row is used.

In [18]:
```
class TargetEncodingExpandingMean(BaseEstimator, TransformerMixin):
    def __init__(self, columns_names):
        self.columns_names = columns_names
        self.learned_values = {}
        self.dataset_mean = np.nan
    def fit(self, X, y, **fit_params):
        X_ = X.copy()
        self.learned_values = {}
        self.dataset_mean = np.mean(y)
        X_["__target__"] = y
        for c in [x for x in X_.columns if x in self.columns_names]:
            stats = (X_[[c,"__target__"]]
                     .groupby(c)["__target__"]
                     .agg(['mean', 'size'])) #
            stats["__target__"] = stats["mean"]
            stats = (stats
                     .drop([x for x in stats.columns if x not in ["__target__",c]], axis = 1)
                     .reset_index())
            self.learned_values[c] = stats
        return self
    def transform(self, X, **fit_params):
        transformed_X = X[self.columns_names].copy()
        for c in transformed_X.columns:
            transformed_X[c] = (transformed_X[[c]]
                                .merge(self.learned_values[c], on = c, how = 'left')
                               )["__target__"]
        transformed_X = transformed_X.fillna(self.dataset_mean)
        return transformed_X
```

D. Addition of more features.

Combinations of feature pairs are used to create new set of categorical features. Naive techniques such as taking pair of existing features and concatenating them together is used as starting point. This increases the feature count from 24 to 278. When the author performed extra boost modeling initially in the one hot encoded data it was found out that **F 14 being not a categorical feature has considerably higher feature importance as opposed to the other features.** This fact was exploited by the author to add **certain transformations of feature 14** so that the importance of feature 14 in the decision trees increases and hence hoping that the AOC score would increase with it. It was seen that it was indeed true to a certain extent and **polynomial transformations** of feature 14 such as square and cubic transformation helped in increasing AUC score by 0.1-0.15. The number of unique values for each feature generated was evaluated and it was found out that a lot of them were high cardinality categorical features. The previously explained functions such as target encoding expanding mean and target encoding smoothing what used with cross validation to create embeddings of these high cardinality features.

Final thoughts: Feature transformations and feature engineering were extremely helpful in getting a boost of .2 two .3 in the final public leaderboard AUC scores. The newly added features were especially helpful for cat boost-based model as it gave the model more categorical data to consume and train on.



4. Models/Techniques Used and Usage of Stacking and Ensembling

Various kinds of models were extensively used two train on this data and get predictions with a decent AUC score.

Initial attempts:

At the very onset models such as **XG boost, LGBM, random forests and logistic regression** were individually trained on the data and they were stacked together with a logistic regression as meta regressor. It was quickly found out that logistic regulation as a meta regressor does not work well in the stacked models. Two overcome this shortcoming XT boost was used as the meta regressor of the stacked models and predictions were made based on this model. This alone was sufficient in taking the author to a score of 0.89-0.90 in the public leaderboard without any feature transformation and feature engineering.

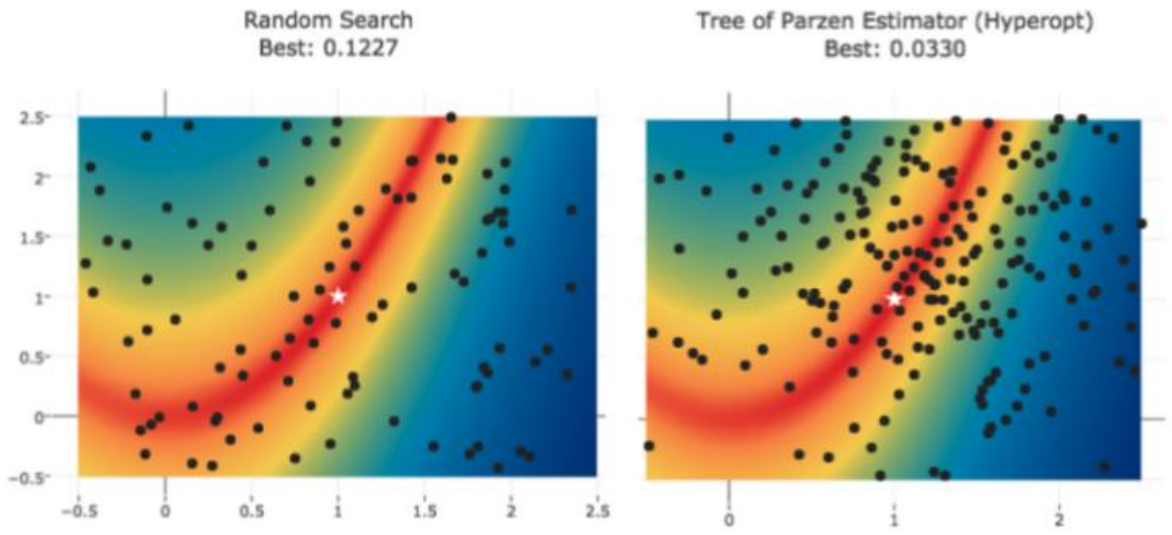
Attempts with Hyperparameter tuning using grid search:

The grid search CV function after scikit learn library was used to do an exhaustive hyperparameter tuning over the provided sample space of the parameters of the above-mentioned models. The best performing models one ranked and the top ranked models of each type word used in an attempt to make a stack model robust enough two predict correctly on the test data. In the later instances the author took 1st three models and made sub stack models out of those three models as in grid search there is always some randomness

and author wanted to be sure that the highest ranked model would actually perform the best out of all in the grid search. This technique increase the score by a very small amount and hence it was required to find a better technique to search through all the sample space of models with extensive parameters such as XG boost and LGBM.

Enter the HyperOpt:

After reading a considerable number of blog posts and kaggle forums it was found out that hyper opt library has a very extensive hyperparameter tuning support and it uses an algorithm called Tree parzen estimator, which is a form of Bayesian estimation. The problem with grid and random search is that these are uninformed methods because they do not use the past results from different values of hyperparameters in the objective function. Evaluating hyperparameters in the objective function is very time-consuming, and the concept of Bayesian optimization is to limit calls to the evaluation function by choosing the next hyperparameter values based on the previous results. This allows the algorithm to spend more time evaluating promising hyperparameter values and less time in low-scoring regions of the hyperparameter space. To illustrate this point, I am attaching a figure that I found on one of the Kaggle forums notebooks explaining this point beautifully. As you can seem TPE guesses are more closely based near the High score and low loss regions, and the best values.



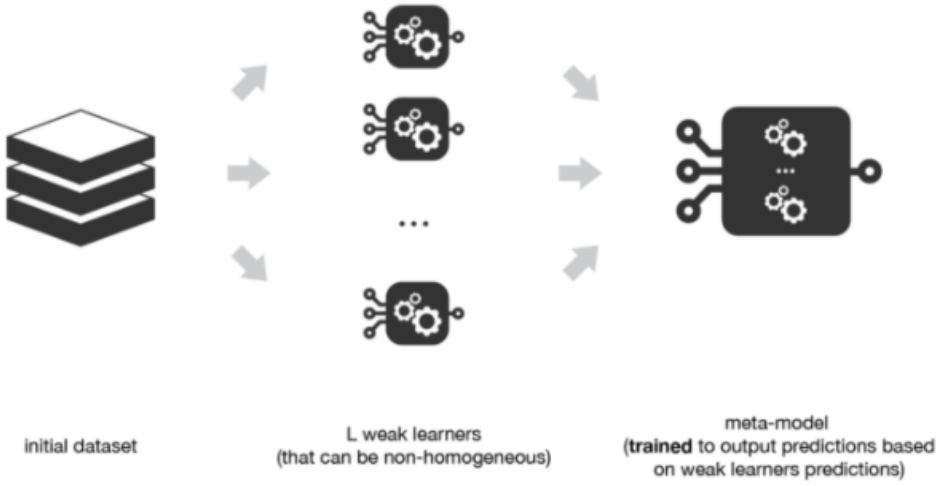
STACKING of classifiers:

Stacking mainly differs from bagging and boosting on two points. First stacking often considers heterogeneous weak learners (different learning algorithms are combined) whereas bagging and boosting consider mainly homogeneous weak learners. Second, stacking learns to combine the base models using a meta-model whereas bagging and boosting combine weak learners following deterministic algorithms. The idea of stacking is to learn several different weak learners and combining them by a meta-model. So basically meta-model works on the output predictions of the weak learners and the available data. For classification problem, it is imperative to choose classifiers as weak learners. To fit a stacking ensemble composed of L learners, we need to do the following:

1. Split data into 2 folds.

2. Choose L weak learners and fit them to the first fold and make predictions for observations in the second fold.
3. Fit the meta-model on the second fold using predictions made by the weak learners as inputs.

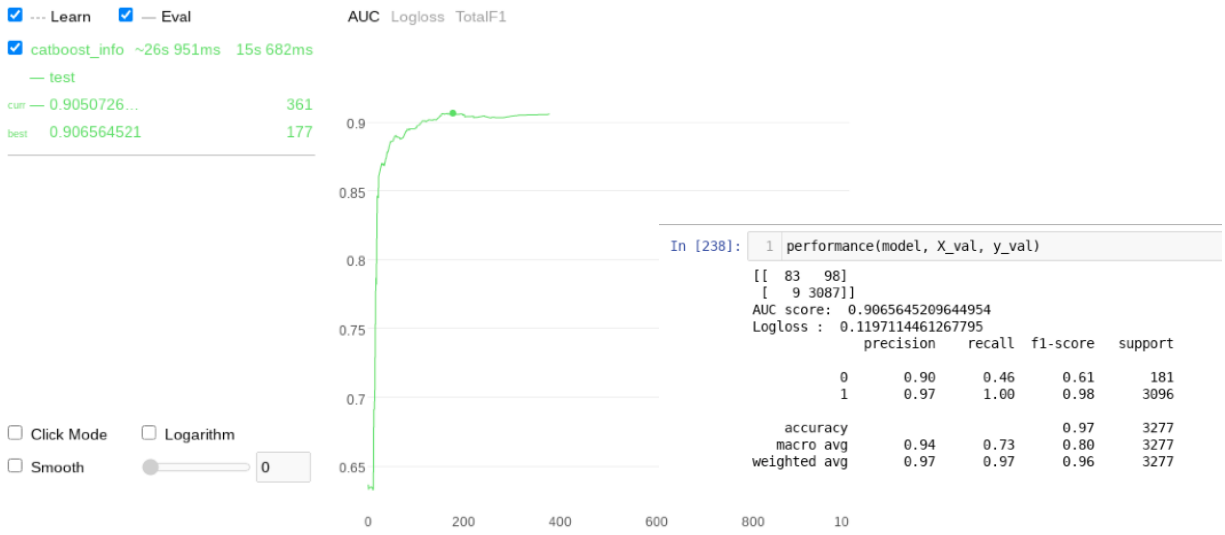
Luckily for us the StackingCVClassifier functions performs the above operations quite well and hence the author heavily used it for stacking.



Stacking consists in training a meta-model to produce outputs based on the outputs returned by some lower layer weak learners.

CATBoost:

Based on the exhaustive reading of kaggle forums it was found out that there is a new kind of boosting algorithm which works really well on categorical based data and classification tasks. The author decided to give it a try so that the crossvalidation AUC score can be increased two high 90s (0.96-0.99). CatBoost was very successful in increasing the internal CV AUC score achieved with certain critical considerations. The cat boost works very well on the raw data and attempts of training it on 1 hot encoded data proved out to be counterproductive. Some of the main advantages of cat boost worth that it provides good quality of predictions with default parameters, has a GPU training mode and hence author could train multiple models and also do very exhaustive hyperopt based parameter tuning oncat boost. There is a sophisticated categorical text an embedding feature support which also helps once authored performs various kinds of feature encoding and embedding using the above-mentioned methods. The cat boost library comes with some very powerful and sophisticated model analysis tools which provide better intuition both on training of the models and the estimation of overfitting. As per the available documentation of CAD boost it uses bag of words models, naïve bayes estimators and object embeddings which are very useful for textual data. The intuition of author was that the provided dataset used to be text based to a large extent and Label encoding was applied by the teaching team on it so that the text/confidential data is masked and abstracted. This gives a “feel” of numerical data but instead it is actually text-based data to a large extent. Hence when CatBoost was included into the inventory of models, the AUC scores jumped. CatBoost, standalone, is quite capable of achieving 0.93 score on public leaderboard and has been tried by the author. The following figures show the extensive visual tools available with the cat boost library so that a better intuition can be made about the fitting of model. Cat boost also comes with it soon cross validation function which again allows repeated stratified K folding to make sure that the final CV does not have much noise.

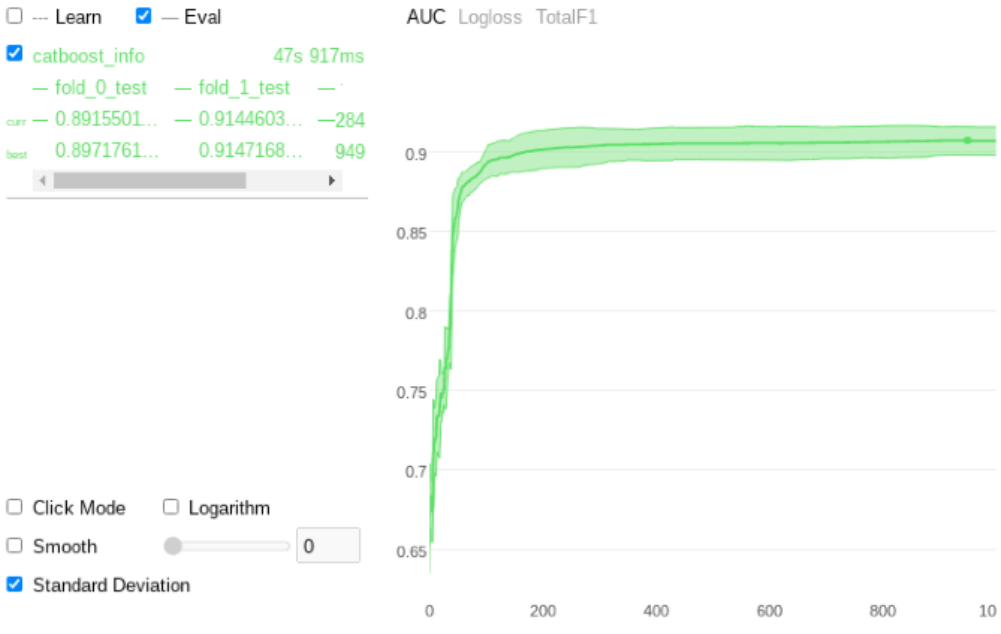



```
In [238]: 1 performance(model, X_val, y_val)

[[  83   98]
 [   9 3087]]
AUC score:  0.9065645209644954
Logloss :  0.1197114461267795
              precision    recall  f1-score   support

           0       0.90      0.46      0.61       181
           1       0.97      1.00      0.98      3096

    accuracy                           0.97      3277
   macro avg       0.94      0.73      0.80      3277
weighted avg       0.97      0.97      0.96      3277
```

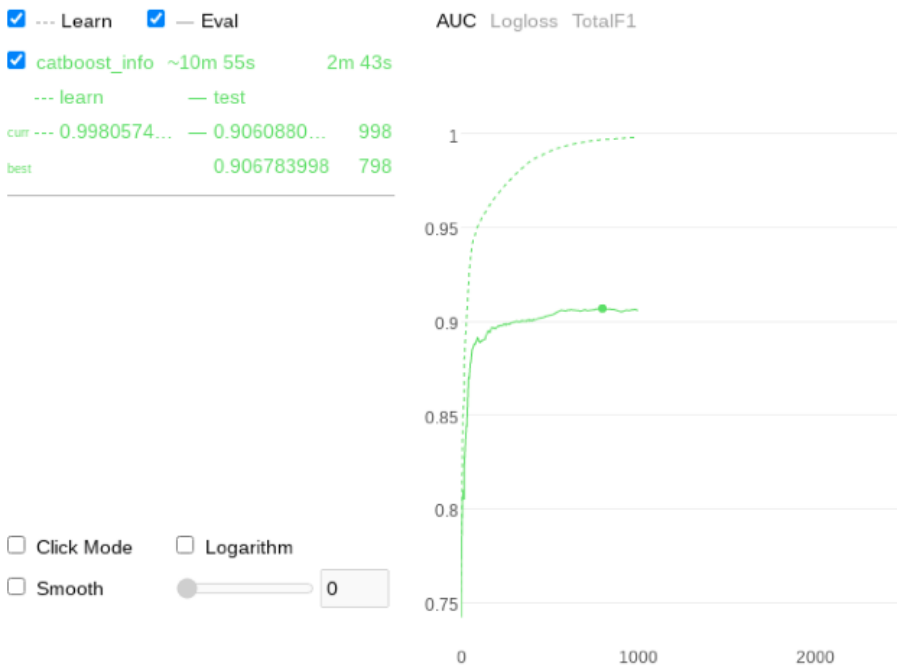
Cross validation to check for overfitting

- Explanation of what cv() function in Catboost does: "The dataset is split into N folds. N-1 folds are used for training, and one fold is used for model performance estimation. N models are updated on each iteration K. Each model is evaluated on its' own validation dataset on each iteration. This produces N metric values on each iteration K. "

```
l]: 1 cv_params = model.get_params()
    2 cv_params.update({
    3     'loss_function': 'Logloss'
    4 })
    5
    6 # CV is always done with the entire train set X_Train, y
    7
    8 cv_data = cv(Pool(X_Train, y, cat_features=cat_features), params=cv_params,
    9              plot=True,
   10              verbose=False)
```



The graph below shows the difference between the learning and testing AUC scores when the model was fitted on a subset of data. The problem was found out and the code was updated to make sure the model trains with the full set of data. The author found that the cv calculation function of CatBoost was quite reliable as the delta between internal AUC and public leader board AUC was less that 0.005. The private leaderboard did cause some drop in the final scores and author lost one ranking position because of it.



Closing Remarks: The above discussion motivates the author in understanding the intricate training and data-based insights. These insights may differ from one dataset to another, but general principles remain the same and hence robust EDA, feature engineering, understanding of models and their interactions with the dataset is required to achieve decent score in the final leaderboard. The features with more catergorical encoding work best with CatBoost and hence the author used the supervised encoded dataset to fit CatBoost. This was different in case of XGBoost and LGBM and RFs. It was observed that they worked very well when the raw data was processed only using the techniques like one-hot encoding and the frequency encoding.

The author feels that all in all, right submissions were selected for the private LB calculations but still there was significant drop of 0.08 AUC score points, which meant loss of 1 position in final rankings. Hence, author is revisiting his CV calculations and is also working on the following post deadline tasks.

5. Post Deadline work:

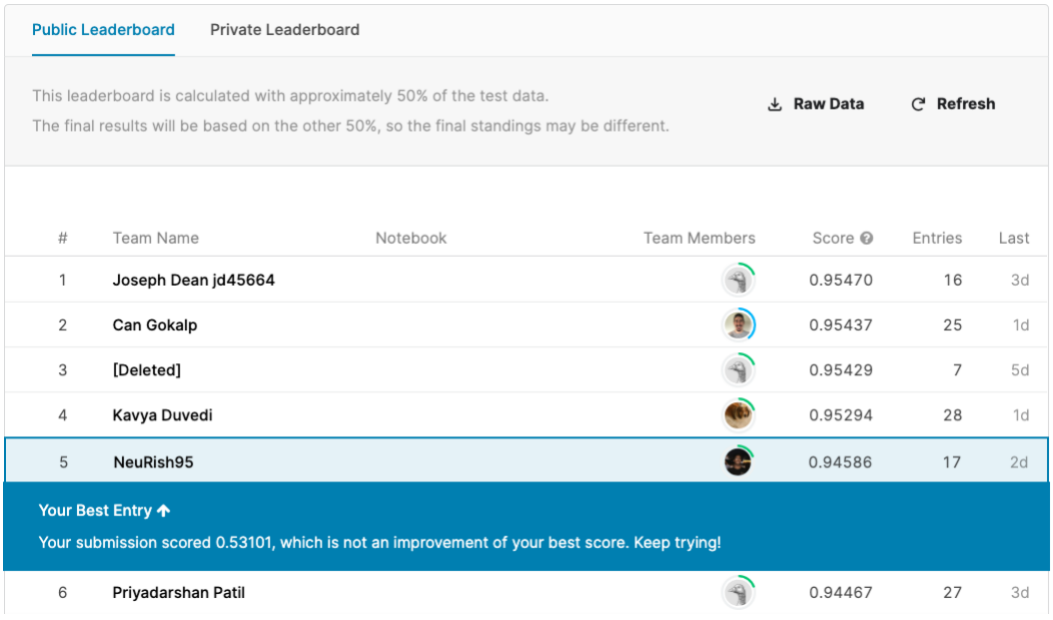
With the above insights from the drop of score in the private leaderboard, author tried to create multiple combinations and blends of above-mentioned models to gain more insights. This is still a work in progress. Application of NN as metaclassfier in a stacked model made of CatBoost, XGBoost and LGBM is on the cards.

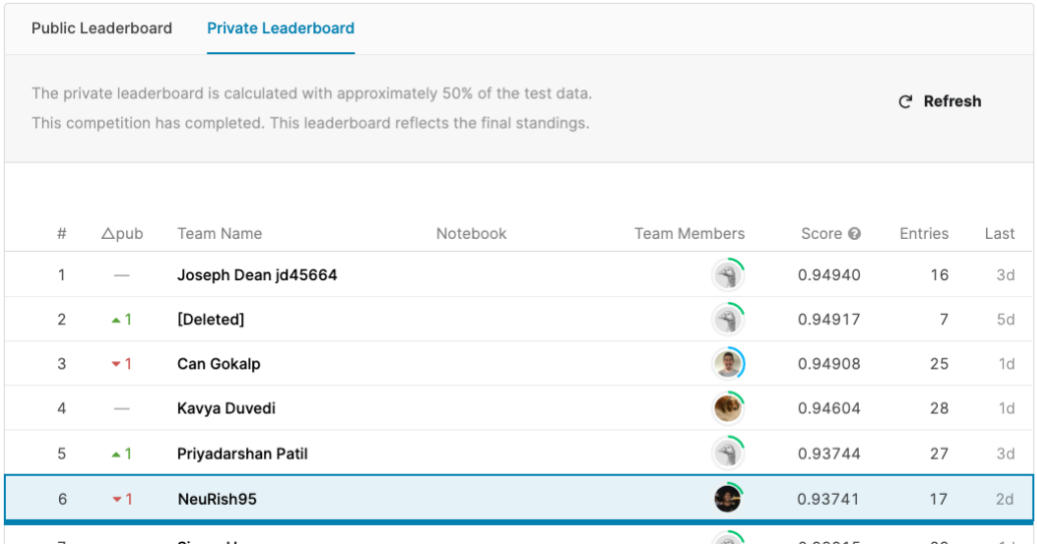
6. References:

I. https://www.kaggle.com/kabure/extensive-eda-and-modeling-xgb-hyperopt
II. https://www.kaggle.com/tilii7/hyperparameter-grid-search-with-xgboost

III. https://www.kaggle.com/mitribunskiy/tutorial-catboost-overview
IV. https://www.kaggle.com/dmitrylarko/kaggledays-sf-3-amazon-supervised-encoding
V. https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/
VI. https://catboost.ai/docs/concepts/python-reference_catboostclassifier.html
VII. https://www.kaggle.com/orpitadas/amazon-catboost-shap-hyperopt-90-auc-silvermedal
VIII. https://www.kaggle.com/ilialar/hyperparameters-tunning-with-hyperopt
IX. https://www.kaggle.com/erikbruin/house-prices-lasso-xgboost-and-a-detailed-eda
X. https://www.kaggle.com/bigironsphere/parameter-tuning-in-one-function-with-hyperopt
XI. https://arxiv.org/pdf/1603.02754.pdf
XII. https://rapids.ai/start.html

SCREENSHOTS OF SCORES:





7. Appendix and Backup

Adding noise using the CV method: The key is to use cross validation again to add the noise to the target encodings. Say if all the target encodings are divided into the n folds, then n-1 folds are used to create the target mean embedding and it is applied on the nth fold as noise.

The following graph shows target distributions per feature.