Problem Statement

Santander wants to find which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

This kernels consists of :

- Importing Data
- Reducing Memory Usage
- Missing Values
- Basic EDA
- Feature Correlation
- Baseliner
- Decision Tree
- Model Importances
- Bayesian Optimisation
- ELI5
- 5Fold Submission

In [1]:

```
!pip install pydotplus
```

```
Collecting pydotplus
  Downloading https://files.pythonhosted.org/packages/60/bf/62567830b700d9f6930e9ab6831d6ba256f
7b0b730acb37278b0ccdffacf/pydotplus-2.0.2.tar.gz (278kB)
    100% |████████████████████████████████| 286kB 8.7MB/s eta 0:00:01
Requirement already satisfied: pyparsing>=2.0.1 in /opt/conda/lib/python3.6/site-packages (from
 pydotplus) (2.2.0)
Building wheels for collected packages: pydotplus
  Building wheel for pydotplus (setup.py) ... done
  Stored in directory: /tmp/.cache/pip/wheels/35/7b/ab/66fb7b2ac1f6df87475b09dc48e707b6e0de80a6
d8444e3628
Successfully built pydotplus
Installing collected packages: pydotplus
Successfully installed pydotplus-2.0.2
You are using pip version 19.0.3, however version 19.1.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

In [2]:

```
import gc
import os
import time
import math
import subprocess
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.model_selection import cross_val_score, train_test_split, KFold, StratifiedKFold

from sklearn.metrics import roc_auc_score
from bayes_opt import BayesianOptimization
from sklearn.model_selection import StratifiedKFold

# Importing all models

# Classification
from sklearn.linear_model import LogisticRegression, ElasticNet, Lasso, Ridge
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, BaggingClassifier, ExtraTreesClassifier, GradientBoostingCl
assifier, \
    RandomForestClassifier, VotingClassifier


import lightgbm as lgb
import xgboost as xgb
import catboost as cat
from catboost import Pool, CatBoostClassifier

import warnings
print(os.listdir("../input"))
warnings.simplefilter('ignore')
```

```
['test.csv', 'train.csv', 'sample_submission.csv']
```

Importing Data and Reducing Memory

In [3]:

```
def reduce_mem_usage(df):
    """ iterate through all the columns of a dataframe and modify the data type
        to reduce memory usage.
    """
    start_mem = df.memory_usage().sum() / 1024 ** 2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('category')

    end_mem = df.memory_usage().sum() / 1024 ** 2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

    return df

def import_data(file):
    """create a dataframe and optimize its memory usage"""
    df = pd.read_csv(file, parse_dates=True, keep_date_col=True)
    df = reduce_mem_usage(df)
    return df
```

In [4]:

```
train = import_data("../input/train.csv")
test = import_data("../input/test.csv")
sub = import_data("../input/sample_submission.csv")

print("\n\nTrain Size : \t{}\nTest Size : \t{}".format(train.shape, test.shape))
```

```
Memory usage of dataframe is 308.23 MB
Memory usage after optimization is: 83.77 MB
Decreased by 72.8%
Memory usage of dataframe is 306.70 MB
Memory usage after optimization is: 83.58 MB
Decreased by 72.7%
Memory usage of dataframe is 3.05 MB
Memory usage after optimization is: 7.48 MB
Decreased by -145.1%


Train Size :    (200000, 202)
Test Size :     (200000, 201)
```

In [5]:

```
train.head()
```

Out[5]:

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	
0	train_0	0	8.921875	-6.785156	11.906250	5.093750	11.460938	-9.281250	5.117188	18.625000	-4.921875	5.746094	2.
1	train_1	0	11.500000	-4.148438	13.859375	5.390625	12.359375	7.042969	5.621094	16.531250	3.146484	8.085938	-0.
2	train_2	0	8.609375	-2.746094	12.078125	7.894531	10.585938	-9.085938	6.941406	14.617188	-4.917969	5.953125	-0.
3	train_3	0	11.062500	-2.152344	8.953125	7.195312	12.585938	-1.835938	5.843750	14.921875	-5.859375	8.242188	2.
4	train_4	0	9.835938	-1.483398	12.875000	6.636719	12.273438	2.449219	5.941406	19.250000	6.265625	7.679688	-9.

In [6]:

```
test.head()
```

Out[6]:

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10
0	test_0	11.062500	7.781250	12.953125	9.429688	11.429688	-2.380859	5.847656	18.265625	2.132812	8.812500	-2.025391
1	test_1	8.531250	1.253906	11.304688	5.187500	9.195312	-4.011719	6.019531	18.625000	-4.414062	5.972656	-1.380859
2	test_2	5.484375	-10.359375	10.140625	7.046875	10.265625	9.804688	4.894531	20.250000	1.523438	8.343750	-4.707031
3	test_3	8.539062	-1.322266	12.023438	6.574219	8.843750	3.173828	4.941406	20.562500	3.375000	7.457031	0.009499
4	test_4	11.703125	-0.132690	14.132812	7.750000	9.101562	-8.585938	6.859375	10.601562	2.988281	7.144531	5.101562

All the features are masked. This way I don't think we can create features based on domain knowledge if we don't know what the feature represent.

- But we can make aggregate features like count, max, min, mean, groupby features if we somehow find any relation between any two features.

Missing Values

Is the data missing at random? Or is this missing systematically? If the user chooses not to respond to some questions, they will be captured as 'blank' in the system/data.

So there could be 3 reasons for these blanks:

1. systematically skipping questions
2. random skipping questions
3. no opinion about those questions At any rate, 1 and 2 might be solved by imputing the blanks/NAs, but the third option might not, resulting in a bias (?).

In [7]:

```
def missing_values(df):
    # Total missing values
    mis_val = df.isnull().sum()

    # Percentage of missing values
    mis_val_percent = 100 * df.isnull().sum() / len(df)

    # Make a table with the results
    mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

    # Rename the columns
    mis_val_table_ren_columns = mis_val_table.rename(
        columns={0: 'Missing Values', 1: '% of Total Values'})

    # Sort the table by percentage of missing descending
    mis_val_table_ren_columns = mis_val_table_ren_columns[
        mis_val_table_ren_columns.iloc[:, 1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)

    # Print some summary information
    print("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
                                                              "There are " + str(mis_val_table_ren_columns.s
hape[0]) +
          " columns that have missing values.")

    # Return the dataframe with missing information
    return mis_val_table_ren_columns

miss_train = missing_values(train)
miss_test = missing_values(test)
```

```
Your selected dataframe has 202 columns.
There are 0 columns that have missing values.
Your selected dataframe has 201 columns.
There are 0 columns that have missing values.
```

Cool we don't have any missing values.

Basic EDA

In [8]:

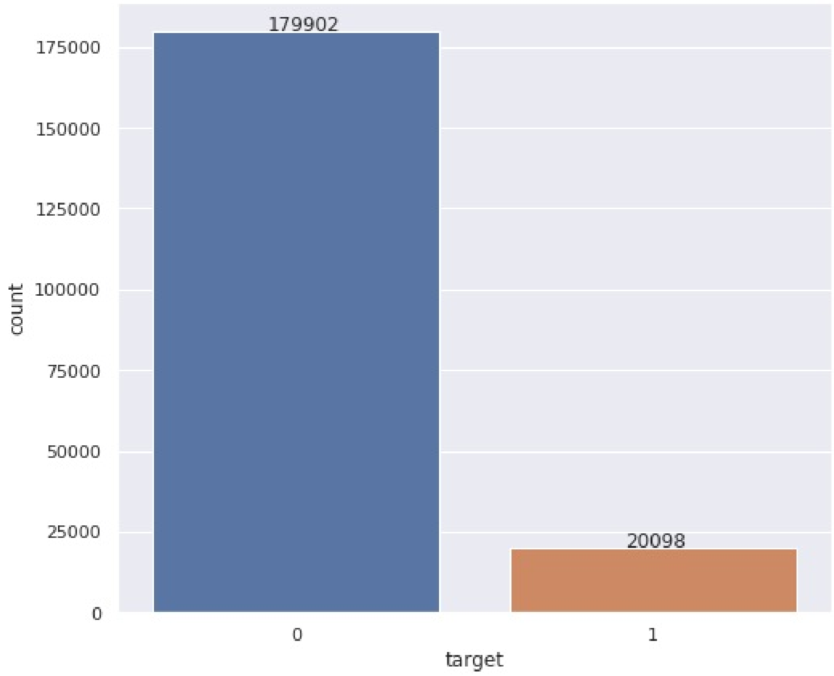
```
def univariate(df, col, vartype, hue=None):
    '''
    Univariate function will plot the graphs based on the parameters.
    df      : dataframe name
    col     : Column name
    vartype : variable type : continuos or categorical
                Continuos(0)   : Distribution, Violin & Boxplot will be plotted.
                Categorical(1) : Countplot will be plotted.
    hue     : It's only applicable for categorical analysis.

    Call: univariate(df=data,col='col',vartype=0)
    '''
    sns.set(style="darkgrid")

    if vartype == 0:
        fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(20, 8))
        ax[0].set_title("Distribution Plot")
        sns.distplot(df[col], ax=ax[0])
        ax[1].set_title("Violin Plot")
        sns.violinplot(data=df, x=col, ax=ax[1], inner="quartile")
        ax[2].set_title("Box Plot")
        sns.boxplot(data=df, x=col, ax=ax[2], orient='v')

    if vartype == 1:
        temp = pd.Series(data=hue)
        print(len(temp.unique()))
        fig, ax = plt.subplots()
        width = len(df[col].unique()) + 6 + 4 * len(temp.unique())
        fig.set_size_inches(width, 7)
        ax = sns.countplot(data=df, x=col, order=df[col].value_counts().index, hue=hue)
        if len(temp.unique()) > 0:
            for p in ax.patches:
                if p.get_height() > 0:
                    ax.annotate('{:1.1f}%'.format((p.get_height() * 100) / float(len(loan_data))),
                                (p.get_x() + 0.05, p.get_height() + 20))
        else:
            for p in ax.patches:
                ax.annotate(p.get_height(), (p.get_x() + 0.32, p.get_height() + 20))
        del temp
    else:
        exit

    plt.show()
```

Target Distribution

In [9]:

```
univariate(train, 'target', 1)
```

0



In [10]:

```
train['target'].value_counts(normalize=True)
```

Out[10]:

```
0    0.89951
1    0.10049
Name: target, dtype: float64
```

Well the data is quite imbalanced. Almost 9:1 ratio of customer not proceding to make a successful transaction.|

Lets check the masked features

Some measures by which we can do some EDA on masked features are checking their :

- mean
- std
- skew
- kurtosis

In [11]:

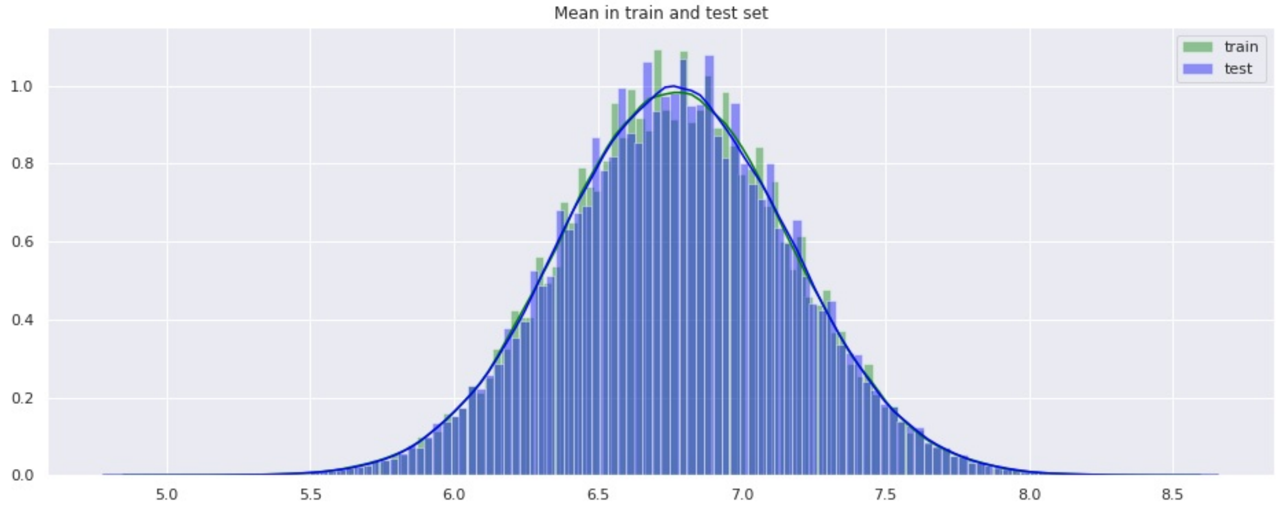
```
target = 'target'
features = train.columns.tolist()
features.remove(target)
features.remove("ID_code")
print("Feature Length : {}".format(len(features)))
```

```
Feature Length : 200
```

Mean

In [12]:

```
plt.figure(figsize=(16,6))
plt.title("Mean in train and test set")
sns.distplot(train[features].mean(axis=1), color="green", kde=True, bins=120, label='train')
sns.distplot(test[features].mean(axis=1), color="blue", kde=True, bins=120, label='test')
plt.legend()
plt.show()
```



Standard Deviation

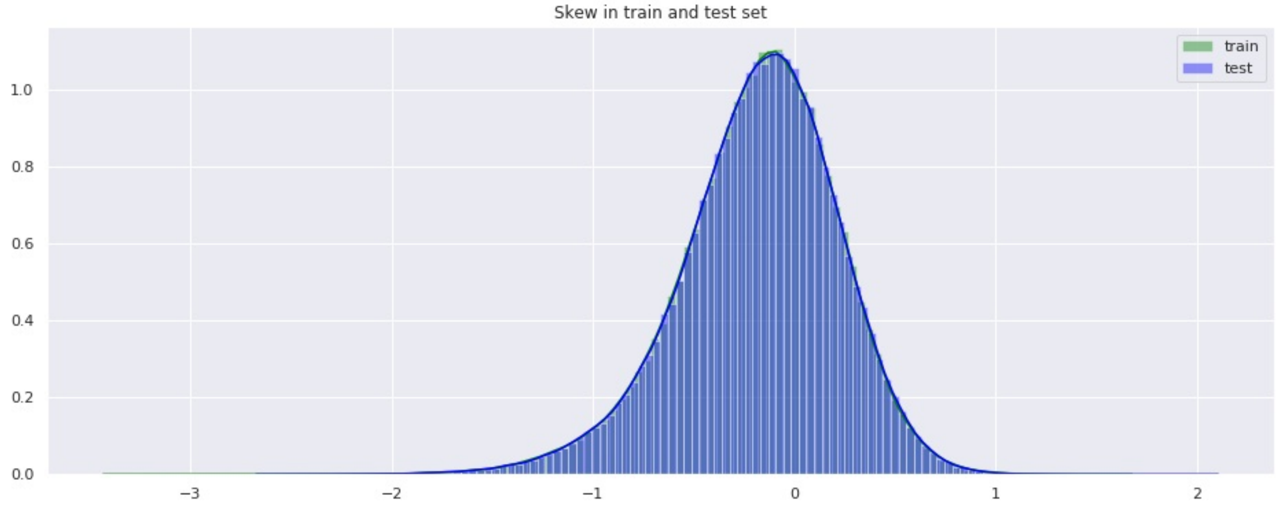
In [13]:

```
plt.figure(figsize=(16,6))
plt.title("STD in train and test set")
sns.distplot(train[features].std(axis=1), color="green", kde=True, bins=120, label='train')
sns.distplot(test[features].std(axis=1), color="blue", kde=True, bins=120, label='test')
plt.legend()
plt.show()
```

Skewness

In [14]:

```
plt.figure(figsize=(16,6))
plt.title("Skew in train and test set")
sns.distplot(train[features].skew(axis=1), color="green", kde=True, bins=120, label='train')
sns.distplot(test[features].skew(axis=1), color="blue", kde=True, bins=120, label='test')
plt.legend()
plt.show()
```



In [15]:

```
# Skewness and Kurtosis
print("Skewness: %f" % train['target'].skew())
print("Kurtosis: %f" % train['target'].kurt())
```

```
Skewness: 2.657642
Kurtosis: 5.063112
```

Kurtosis

As we have most scores of features near mean i.e(located new mean/normally distributed) and kurtosis is 5.063 therefore this is leptokurtic.

Comparing Distributions of Features

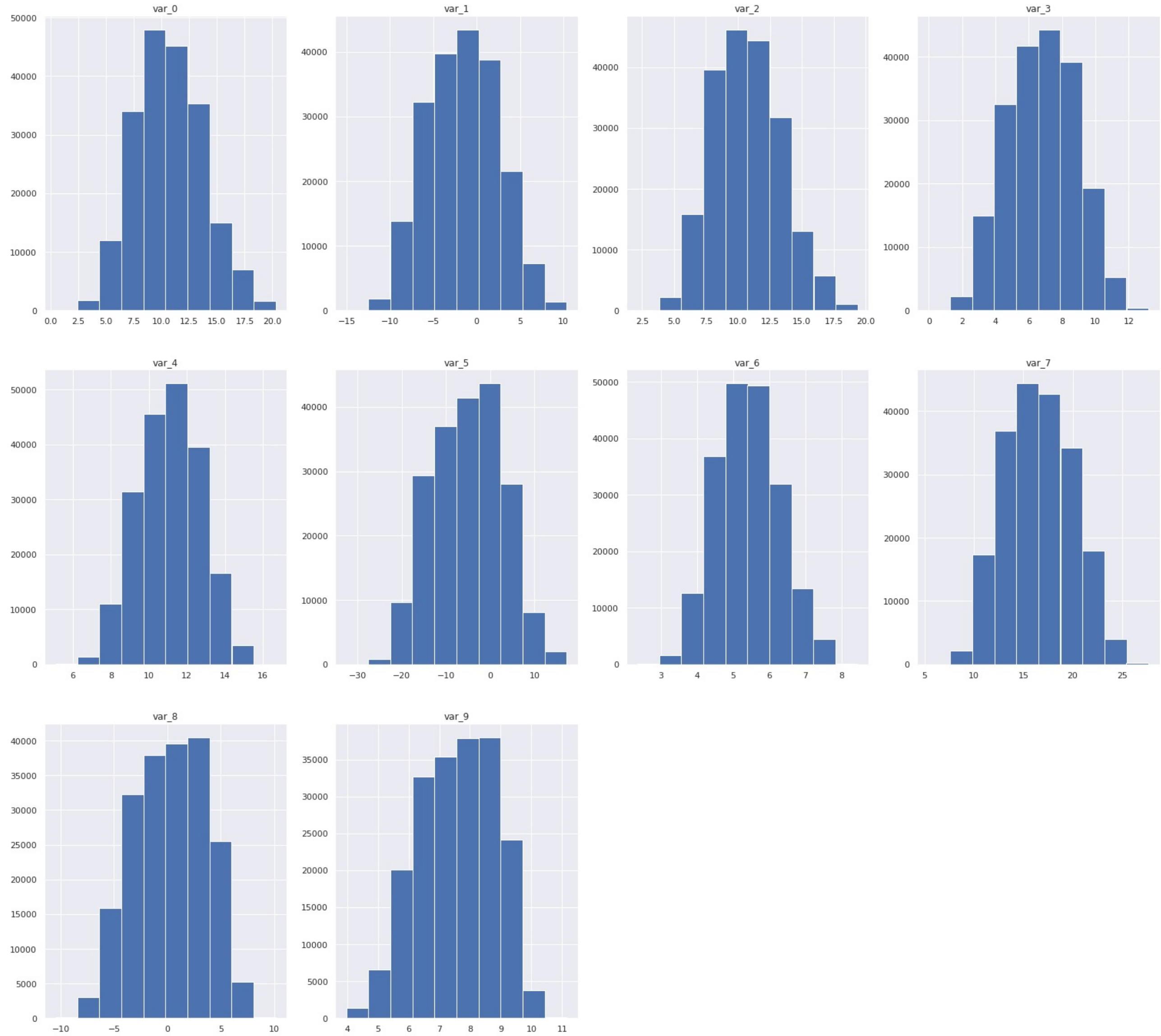
In [16]:

```
def plot_and_compare(data, num):
    print('Distributions of first {} columns'.format(num))
    plt.figure(figsize=(26, 24))
    for i, col in enumerate(list(data.columns)[2:num + 2]):
        plt.subplot(math.ceil(num/4), 4, i + 1)
        plt.hist(data[col])
        plt.title(col)
    plt.show()
```

In [17]:

```
plot_and_compare(train, 10)
```

Distributions of first 10 columns



We can see that first 10 features have almost Normal Distribution. Lets check for more(50).

In [18]:

```
plot_and_compare(train, 50)
```

```
Distributions of first 50 columns
```

Not only they are normally distributed almost all the features have almost same distribution with same scales.

Lets check correlation to confirm the same.

Correlation

In [19]:

```
def plot_corr(data):
    data_correlation = data.corr()
    f, ax = plt.subplots(figsize=(25, 25))
    sns.heatmap(data_correlation,
                xticklabels=data_correlation.columns.values,
                yticklabels=data_correlation.columns.values, annot=True)
    plt.show()
```

In [20]:

```
# AWFUL lot of time taken as the data is of 200 variables consisting of 0.2 Million recordss
# plot_corr(train.iloc[:10])
```

Number of Unique Variables

In [21]:

```
tuniq = train.nunique().sort_values().reset_index()
tuniq.head()
```

Out[21]:

	index	0
0	target	2
1	var_68	15
2	var_108	127
3	var_12	150
4	var_25	237

Train and Test Distributions

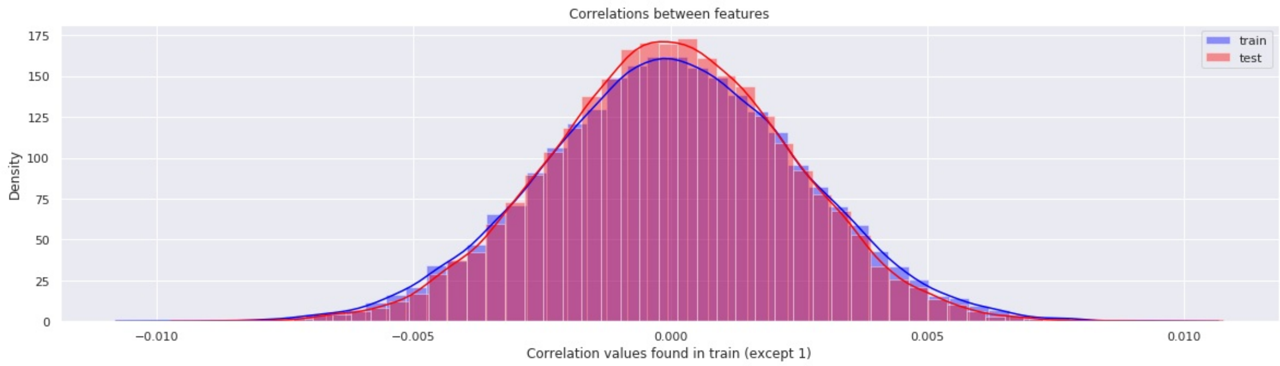
Do both Train and Test set have same distributions?

In [22]:

```
train_correlations = train.drop(["target"], axis=1).corr()
train_correlations = train_correlations.values.flatten()
train_correlations = train_correlations[train_correlations != 1]

test_correlations = test.corr()
test_correlations = test_correlations.values.flatten()
test_correlations = test_correlations[test_correlations != 1]

plt.figure(figsize=(20,5))
sns.distplot(train_correlations, color="Blue", label="train")
sns.distplot(test_correlations, color="Red", label="test")
plt.xlabel("Correlation values found in train (except 1)")
plt.ylabel("Density")
plt.title("Correlations between features");
plt.legend()
```

Out[22]:

<matplotlib.legend.Legend at 0x7f2d9d1c9160>



As we have the peak at 0.0 this states that we have no correlation between features at all. (Might have to check that)

Modelling

Baseliner

In [23]:

```
def baseliner_clas(train, feat, target, cv, metric):

    eval_dict = {}
    models = [lgb.LGBMClassifier()]
    # LogisticRegression(), SVC(), GaussianNB(), KNeighborsClassifier(), DecisionTreeClassifier(), ExtraTree
Classifier(), AdaBoostClassifier(), BaggingClassifier(),
    # RandomForestClassifier(), ExtraTreesClassifier(), GradientBoostingClassifier(), xgb.XGBClassifier(), c
at.CatBoostClassifier(verbose=0)

    for model in models:
        model_name = str(model).split("(")[0]
        results = cross_val_score(model, train[feat], train[target], cv=cv,
                                  scoring=metric)
        print(model_name, results.mean(), results)
        eval_dict[model_name] = results.mean()

    return eval_dict
```

In [24]:

```
base_1 = baseliner_clas(train, features, target, 3, 'roc_auc')
```

```
LGBMClassifier 0.863698056662805 [0.86160664 0.86346855 0.86601898]
```

In [25]:

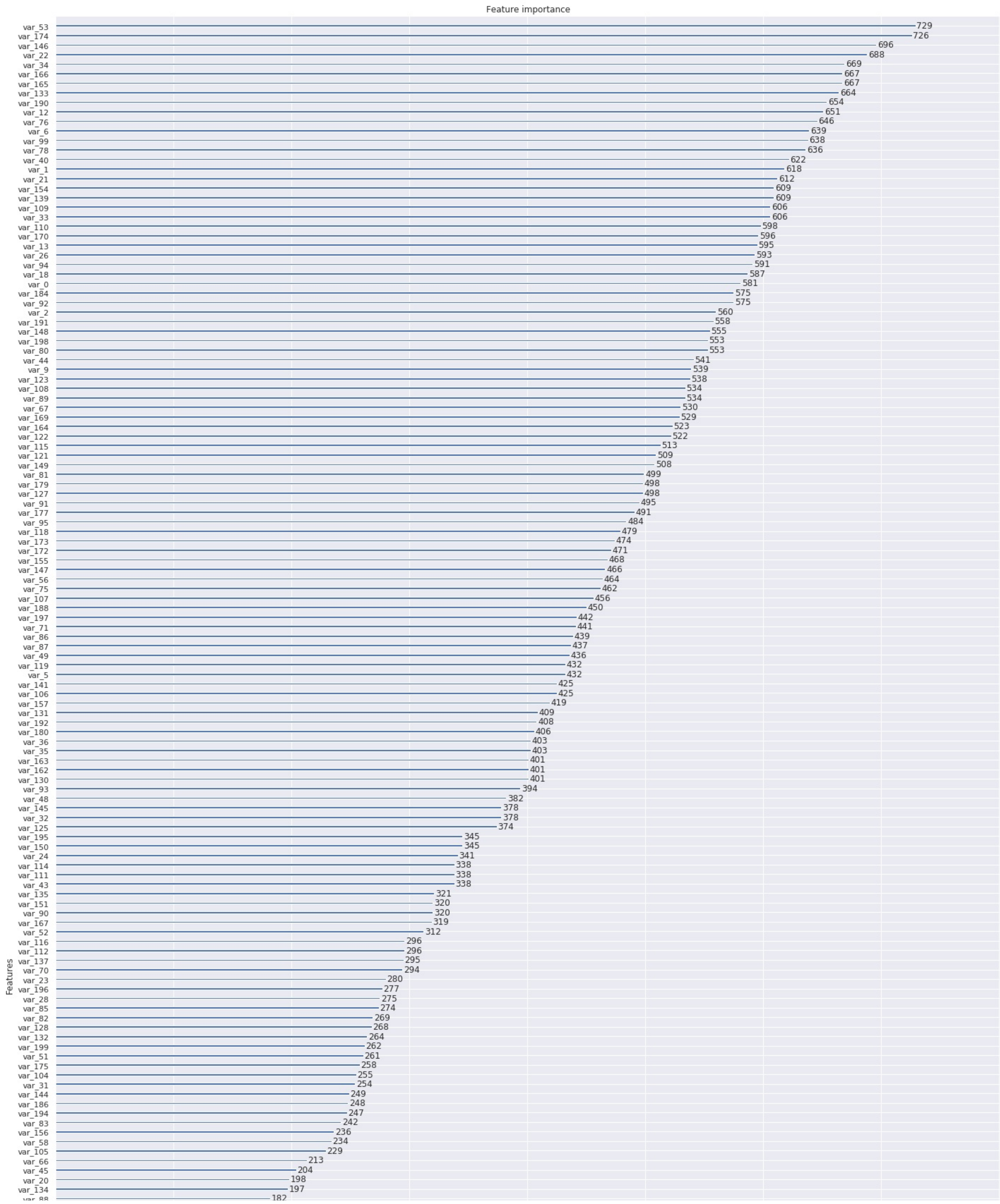
```
def lgb_model(train, feat, target):
    x_train, x_valid, y_train, y_valid = train_test_split(train[feat], train[target], test_size=0.2, random_
state=13)

    train_set = lgb.Dataset(x_train, label=y_train)
    valid_set = lgb.Dataset(x_valid, label=y_valid)

    MAX_ROUNDS = 2000
    params = {
        "boosting": 'gbdt', # "dart",
        "learning_rate": 0.01,
        "nthread": -1,
        "seed": 13,
        "num_boost_round": MAX_ROUNDS,
        "objective": "binary",
        "metric": "auc",
    }

    model = lgb.train(
        params,
        train_set=train_set,
        valid_sets=[train_set, valid_set],
        early_stopping_rounds=50,
        verbose_eval=100
    )

    lgb.plot_importance(model, figsize=(24, 50))

    return model
```

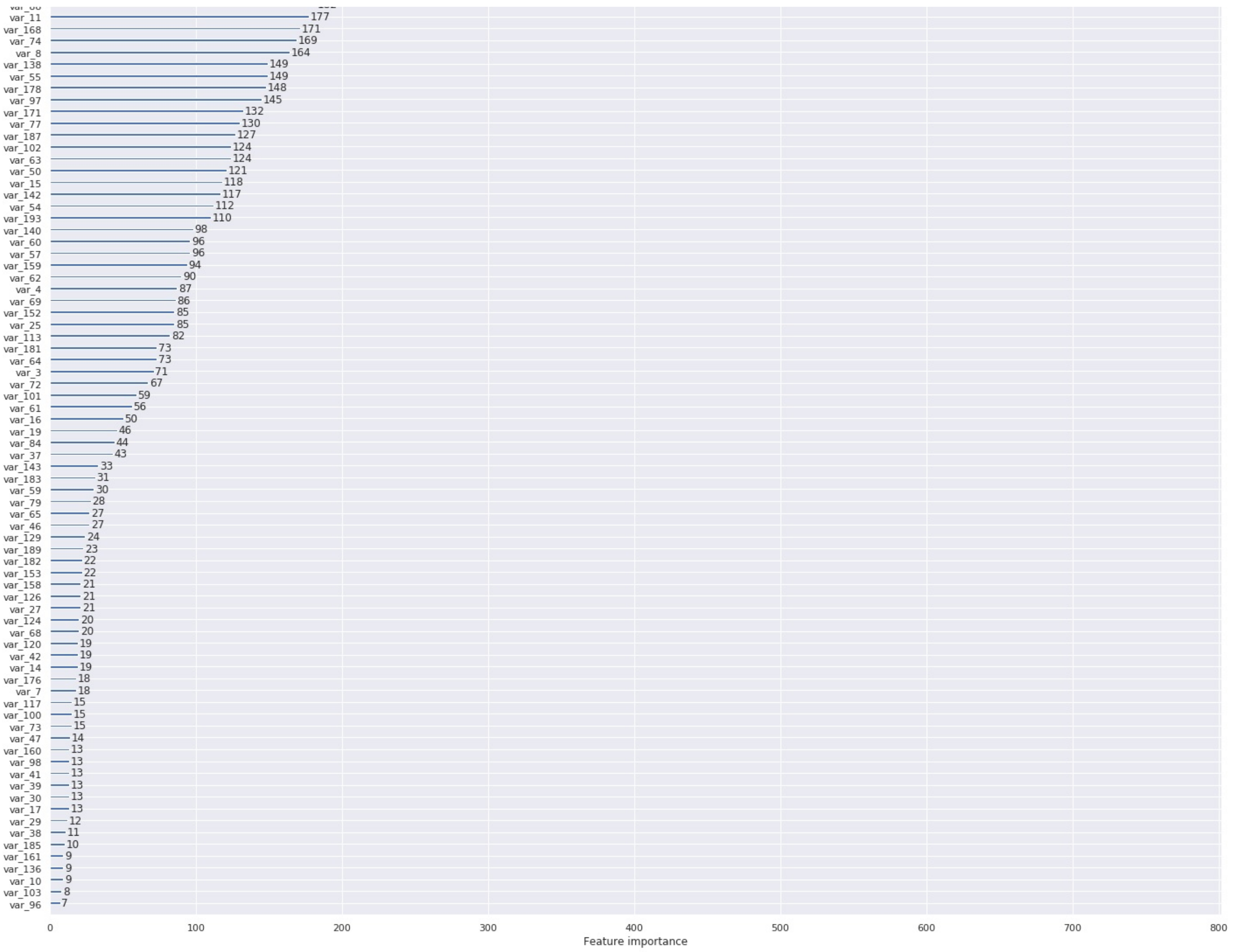
LGB Feature Importance

In [26]:

```
model = lgb_model(train, features, target)
```

```
Training until validation scores don't improve for 50 rounds.
[100]	training's auc: 0.808229	valid_1's auc: 0.783972
[200]	training's auc: 0.849473	valid_1's auc: 0.816763
[300]	training's auc: 0.872012	valid_1's auc: 0.833576
[400]	training's auc: 0.887753	valid_1's auc: 0.845329
[500]	training's auc: 0.899421	valid_1's auc: 0.85369
[600]	training's auc: 0.908498	valid_1's auc: 0.860118
[700]	training's auc: 0.916037	valid_1's auc: 0.865098
[800]	training's auc: 0.922215	valid_1's auc: 0.869045
[900]	training's auc: 0.927419	valid_1's auc: 0.872287
[1000]	training's auc: 0.932136	valid_1's auc: 0.87518
[1100]	training's auc: 0.936347	valid_1's auc: 0.877597
[1200]	training's auc: 0.940073	valid_1's auc: 0.879591
[1300]	training's auc: 0.943386	valid_1's auc: 0.881509
[1400]	training's auc: 0.946381	valid_1's auc: 0.883191
[1500]	training's auc: 0.949136	valid_1's auc: 0.884555
[1600]	training's auc: 0.951636	valid_1's auc: 0.885831
[1700]	training's auc: 0.954023	valid_1's auc: 0.887035
[1800]	training's auc: 0.956277	valid_1's auc: 0.888005
[1900]	training's auc: 0.958292	valid_1's auc: 0.888845
[2000]	training's auc: 0.960191	valid_1's auc: 0.889645
Did not meet early stopping. Best iteration is:
[2000]	training's auc: 0.960191	valid_1's auc: 0.889645
```



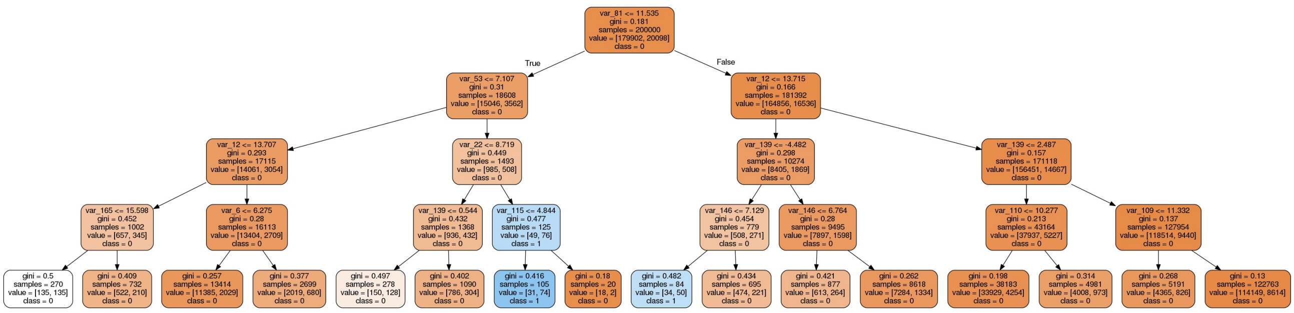


Decision Tree

In [27]:

```
from sklearn.tree import export_graphviz
import pydotplus
from IPython.display import Image as PImage
from sklearn.externals.six import StringIO

dtree = DecisionTreeClassifier(max_depth=4)
dtree.fit(train[features], train[target])

dot_data = StringIO()
export_graphviz(dtree, out_file=dot_data, max_depth = 4,
                     impurity = True, feature_names = features,
                     class_names = ['0', '1'], rounded = True, filled= True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
PImage(graph.create_png())
```

Out[27]:



Bayesian Optimisation

In [28]:

```
bayesian_tr_index, bayesian_val_index  = list(StratifiedKFold(n_splits=2, shuffle=True, random_state=1).spli
t(train, train.target.values))[0]
print(len(bayesian_tr_index), len(bayesian_val_index))
```

```
100000 100000
```

In [29]:

```
# Function for LGB model creation for bayesian optimisation

def LGB_bayesian(
    num_leaves,  # int
    min_data_in_leaf,  # int
    learning_rate,
    min_sum_hessian_in_leaf,    # int
    feature_fraction,
    lambda_l1,
    lambda_l2,
    min_gain_to_split,
    max_depth):

    # LightGBM expects next three parameters need to be integer. So we make them integer
    num_leaves = int(num_leaves)
    min_data_in_leaf = int(min_data_in_leaf)
    max_depth = int(max_depth)

    assert type(num_leaves) == int
    assert type(min_data_in_leaf) == int
    assert type(max_depth) == int

    param = {
        'num_leaves': num_leaves,
        'max_bin': 63,
        'min_data_in_leaf': min_data_in_leaf,
        'learning_rate': learning_rate,
        'min_sum_hessian_in_leaf': min_sum_hessian_in_leaf,
        'bagging_fraction': 1.0,
        'bagging_freq': 5,
        'feature_fraction': feature_fraction,
        'lambda_l1': lambda_l1,
        'lambda_l2': lambda_l2,
        'min_gain_to_split': min_gain_to_split,
        'max_depth': max_depth,
        'save_binary': True,
        'seed': 13,
        'feature_fraction_seed': 13,
        'bagging_seed': 13,
        'drop_seed': 13,
        'data_random_seed': 13,
        'objective': 'binary',
        'boosting_type': 'gbdt',
        'verbose': 1,
        'metric': 'auc',
        'is_unbalance': True,
        'boost_from_average': False,

    }


    xg_train = lgb.Dataset(train.iloc[bayesian_tr_index][features].values,
                           label=train.iloc[bayesian_tr_index][target].values,
                           feature_name=features,
                           free_raw_data = False
                           )
    xg_valid = lgb.Dataset(train.iloc[bayesian_val_index][features].values,
                           label=train.iloc[bayesian_val_index][target].values,
                           feature_name=features,
                           free_raw_data = False
                           )

    num_round = 5000
    clf = lgb.train(param, xg_train, num_round, valid_sets = [xg_valid], verbose_eval=250, early_stopping_ro
unds = 50)

    predictions = clf.predict(train.iloc[bayesian_val_index][features].values, num_iteration=clf.best_iterat
ion)

    score = roc_auc_score(train.iloc[bayesian_val_index][target].values, predictions)
```

```python
    return score

# Region Space for Bayesian Optimisation
region_space_LGB = {
    'num_leaves': (5, 20),
    'min_data_in_leaf': (5, 20),
    'learning_rate': (0.01, 0.3),
    'min_sum_hessian_in_leaf': (0.00001, 0.01),
    'feature_fraction': (0.05, 0.5),
    'lambda_l1': (0, 5.0),
    'lambda_l2': (0, 5.0),
    'min_gain_to_split': (0, 1.0),
    'max_depth':(3,15),
}

LGB_BO = BayesianOptimization(LGB_bayesian, region_space_LGB, random_state=13)
```

In [30]:

```python
init_points = 5
n_iter = 5

LGB_BO.maximize(init_points=init_points, n_iter=n_iter, acq='ucb', xi=0.0, alpha=1e-6)
```

```
|   iter    |  target   | featur... | lambda_l1 | lambda_l2 | learni... | max_depth | min_da...
 | min_ga... | min_su... | num_le... |
-----------------------------------------------------------------------------------------------
-------------------------------------
Training until validation scores don't improve for 50 rounds.
Early stopping, best iteration is:
[172]   valid_0's auc: 0.878482
|  1        |  0.8785   |  0.4      |  1.188    |  4.121    |  0.2901   |  14.67    |  11.8
 |  0.609    |  0.007758 |  14.62    |
Training until validation scores don't improve for 50 rounds.
[250]   valid_0's auc: 0.841747
[500]   valid_0's auc: 0.865883
[750]   valid_0's auc: 0.876802
[1000]  valid_0's auc: 0.88262
[1250]  valid_0's auc: 0.886424
[1500]  valid_0's auc: 0.888831
[1750]  valid_0's auc: 0.890229
[2000]  valid_0's auc: 0.890931
[2250]  valid_0's auc: 0.891364
Early stopping, best iteration is:
[2297]  valid_0's auc: 0.891447
|  2        |  0.8914   |  0.3749   |  0.1752   |  1.492    |  0.02697  |  13.28    |  10.59
 |  0.6798   |  0.00257  |  10.21    |
Training until validation scores don't improve for 50 rounds.
[250]   valid_0's auc: 0.881064
[500]   valid_0's auc: 0.889032
[750]   valid_0's auc: 0.891408
[1000]  valid_0's auc: 0.892038
Early stopping, best iteration is:
[1081]  valid_0's auc: 0.892177
|  3        |  0.8922   |  0.05424  |  1.792    |  4.745    |  0.07319  |  6.833    |  18.77
 |  0.0319   |  0.000660 |  14.45    |
Training until validation scores don't improve for 50 rounds.
[250]   valid_0's auc: 0.878957
[500]   valid_0's auc: 0.883745
Early stopping, best iteration is:
[558]   valid_0's auc: 0.884385
|  4        |  0.8844   |  0.4432   |  0.04358  |  3.733    |  0.2457   |  3.909    |  14.85
 |  0.5093   |  0.004804 |  19.33    |
Training until validation scores don't improve for 50 rounds.
[250]   valid_0's auc: 0.882276
[500]   valid_0's auc: 0.88963
[750]   valid_0's auc: 0.891443
Early stopping, best iteration is:
[750]   valid_0's auc: 0.891443
|  5        |  0.8914   |  0.05001  |  1.235    |  3.561    |  0.1041   |  6.324    |  15.43
 |  0.9186   |  0.002452 |  11.87    |
Training until validation scores don't improve for 50 rounds.
[250]   valid_0's auc: 0.883366
[500]   valid_0's auc: 0.888091
Early stopping, best iteration is:
[481]   valid_0's auc: 0.888299
|  6        |  0.8883   |  0.2888   |  0.06764  |  0.06368  |  0.2366   |  8.822    |  19.91
 |  0.7963   |  0.009826 |  5.699    |
```

```
Training until validation scores don't improve for 50 rounds.
[250]	valid_0's auc: 0.878846
[500]	valid_0's auc: 0.886936
Early stopping, best iteration is:
[665]	valid_0's auc: 0.887614
|  7         |  0.8876    |  0.21      |  4.978     |  0.6158    |  0.2031    |  3.304     |  5.315
 |  0.03679  |  0.009781 |  7.858     |
Training until validation scores don't improve for 50 rounds.
[250]	valid_0's auc: 0.869858
[500]	valid_0's auc: 0.884462
[750]	valid_0's auc: 0.889389
[1000]	valid_0's auc: 0.890797
Early stopping, best iteration is:
[988]	valid_0's auc: 0.890881
|  8         |  0.8909    |  0.2457    |  0.4681    |  4.634     |  0.1089    |  5.749     |  5.199
 |  0.03918  |  0.007056 |  5.492     |
Training until validation scores don't improve for 50 rounds.
[250]	valid_0's auc: 0.863459
[500]	valid_0's auc: 0.879242
[750]	valid_0's auc: 0.886708
[1000]	valid_0's auc: 0.89035
[1250]	valid_0's auc: 0.892283
[1500]	valid_0's auc: 0.893154
Early stopping, best iteration is:
[1685]	valid_0's auc: 0.893483
|  9         |  0.8935    |  0.07253   |  4.779     |  3.22      |  0.07664   |  15.0      |  6.072
 |  0.4635   |  0.007511 |  5.002     |
Training until validation scores don't improve for 50 rounds.
[250]	valid_0's auc: 0.877534
[500]	valid_0's auc: 0.883738
Early stopping, best iteration is:
[600]	valid_0's auc: 0.884501
|  10        |  0.8845    |  0.3822    |  4.973     |  0.07274   |  0.2226    |  3.35      |  19.87
 |  0.9903   |  0.002911 |  19.94     |
===============================================================================================
======================================
```

In [31]:

```python
LGB_BO.max['target']
```

Out[31]:

```
0.893483494395378
```

In [32]:

```python
LGB_BO.max['params']
```

Out[32]:

```
{'feature_fraction': 0.0725311213806508,
 'lambda_l1': 4.778909939998174,
 'lambda_l2': 3.220122059503974,
 'learning_rate': 0.07663544326133201,
 'max_depth': 14.99504485809263,
 'min_data_in_leaf': 6.071757005556453,
 'min_gain_to_split': 0.4634735801572327,
 'min_sum_hessian_in_leaf': 0.007510919138894667,
 'num_leaves': 5.001626877554514}
```

In [33]:

```
param_lgb = {
        'num_leaves': int(LGB_BO.max['params']['num_leaves']),
        'max_bin': 63,
        'min_data_in_leaf': int(LGB_BO.max['params']['min_data_in_leaf']),
        'learning_rate': LGB_BO.max['params']['learning_rate'],
        'min_sum_hessian_in_leaf': LGB_BO.max['params']['min_sum_hessian_in_leaf'],
        'bagging_fraction': 1.0,
        'bagging_freq': 5,
        'feature_fraction': LGB_BO.max['params']['feature_fraction'],
        'lambda_l1': LGB_BO.max['params']['lambda_l1'],
        'lambda_l2': LGB_BO.max['params']['lambda_l2'],
        'min_gain_to_split': LGB_BO.max['params']['min_gain_to_split'],
        'max_depth': int(LGB_BO.max['params']['max_depth']),
        'save_binary': True,
        'seed': 13,
        'feature_fraction_seed': 13,
        'bagging_seed': 13,
        'drop_seed': 13,
        'data_random_seed': 13,
        'objective': 'binary',
        'boosting_type': 'gbdt',
        'verbose': 1,
        'metric': 'auc',
        'is_unbalance': True,
        'boost_from_average': False,
    }
```

Model Interpreting

ELI5

In [34]:

```
import eli5

model = lgb.LGBMClassifier(**param_lgb, n_estimators = 20000, n_jobs = -1)
X_train, X_valid, y_train, y_valid = train_test_split(train[features], train[target], test_size=0.2, stratif
y=train[target])
model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_valid, y_valid)], verbose=1000, early_stopping_
rounds=200)

eli5.show_weights(model, targets=[0, 1], feature_names=list(X_train.columns), top=40, feature_filter=lambda
x: x != '<BIAS>')
```

```
Training until validation scores don't improve for 200 rounds.
[1000]  training's auc: 0.915152        valid_1's auc: 0.891456
[2000]  training's auc: 0.929421        valid_1's auc: 0.895934
Early stopping, best iteration is:
[2356]  training's auc: 0.93366 valid_1's auc: 0.896235
```

Out[34]:

Weight	Feature
0.0285	var_81
0.0245	var_139
0.0208	var_12
0.0208	var_53
0.0192	var_110
0.0165	var_26
0.0164	var_174
0.0161	var_76
0.0161	var_6
0.0158	var_166
0.0156	var_146
0.0148	var_80
0.0144	var_22
0.0141	var_99
0.0141	var_109
0.0137	var_21
0.0134	var_2
0.0131	var_148
0.0130	var_0
0.0129	var_133
0.0129	var_165
0.0129	var_13
0.0126	var_44
0.0124	var_190
0.0121	var_198
0.0120	var_78
0.0113	var_34
0.0113	var_179
0.0109	var_108
0.0107	var_33
0.0104	var_40
0.0104	var_1
0.0102	var_92
0.0101	var_94
0.0101	var_191
0.0100	var_169
0.0098	var_170
0.0097	var_115
0.0097	var_164
0.0097	var_154
… 160 more …	

Taking top 100 features and checking if the scores improves.

In [35]:

```
top_features = [i for i in eli5.formatters.as_dataframe.explain_weights_df(model).feature if 'BIAS' not in i
][:100]
X1 = train[top_features]
X_train, X_valid, y_train, y_valid = train_test_split(X1, train[target], test_size=0.2, stratify=train[targe
t])
model.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_valid, y_valid)], verbose=1000, early_stopping_
rounds=200)
```

```
Training until validation scores don't improve for 200 rounds.
[1000]  training's auc: 0.904977        valid_1's auc: 0.88435
Early stopping, best iteration is:
[1659]  training's auc: 0.912177        valid_1's auc: 0.88535
```

Out[35]:

```
LGBMClassifier(bagging_fraction=1.0, bagging_freq=5, bagging_seed=13,
        boost_from_average=False, boosting_type='gbdt', class_weight=None,
        colsample_bytree=1.0, data_random_seed=13, drop_seed=13,
        feature_fraction=0.0725311213806508, feature_fraction_seed=13,
        importance_type='split', is_unbalance=True,
        lambda_l1=4.778909939998174, lambda_l2=3.220122059503974,
        learning_rate=0.07663544326133201, max_bin=63, max_depth=14,
        metric='auc', min_child_samples=20, min_child_weight=0.001,
        min_data_in_leaf=6, min_gain_to_split=0.4634735801572327,
        min_split_gain=0.0, min_sum_hessian_in_leaf=0.007510919138894667,
        n_estimators=20000, n_jobs=-1, num_leaves=5, objective='binary',
        random_state=None, reg_alpha=0.0, reg_lambda=0.0, save_binary=True,
        seed=13, silent=True, subsample=1.0, subsample_for_bin=200000,
        subsample_freq=0, verbose=1)
```

- Without removal score : 0.89538
- With removal score : 0.883403

So ELI5 isn't helping in reducing the features.

5Fold Prediction & Submission

In [36]:

```
%%time
folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=13)

y_pred_lgb = np.zeros(len(test))
num_round = 20000
for fold_n, (train_index, valid_index) in enumerate(folds.split(train[features], train[target])):
    print('Fold', fold_n, 'started at', time.ctime())
    X_train, X_valid = train[features].iloc[train_index], train[features].iloc[valid_index]
    y_train, y_valid = train[target].iloc[train_index], train[target].iloc[valid_index]

    train_data = lgb.Dataset(X_train, label=y_train)
    valid_data = lgb.Dataset(X_valid, label=y_valid)

    lgb_model = lgb.train(
        param_lgb,
        train_data,num_round,
        valid_sets = [train_data, valid_data],
        verbose_eval=1000,
        early_stopping_rounds = 1000)

    y_pred_lgb += lgb_model.predict(test[features], num_iteration=lgb_model.best_iteration)/5
```

```
Fold 0 started at Thu May 23 21:53:56 2019
Training until validation scores don't improve for 1000 rounds.
[1000]  training's auc: 0.915072        valid_1's auc: 0.8925
[2000]  training's auc: 0.929153        valid_1's auc: 0.89617
[3000]  training's auc: 0.940718        valid_1's auc: 0.896497
[4000]  training's auc: 0.950536        valid_1's auc: 0.896168
Early stopping, best iteration is:
[3245]  training's auc: 0.943273        valid_1's auc: 0.896583
Fold 1 started at Thu May 23 21:55:43 2019
Training until validation scores don't improve for 1000 rounds.
[1000]  training's auc: 0.915683        valid_1's auc: 0.887972
[2000]  training's auc: 0.929641        valid_1's auc: 0.892794
[3000]  training's auc: 0.941147        valid_1's auc: 0.892949
Early stopping, best iteration is:
[2920]  training's auc: 0.940291        valid_1's auc: 0.893084
Fold 2 started at Thu May 23 21:57:21 2019
Training until validation scores don't improve for 1000 rounds.
[1000]  training's auc: 0.914678        valid_1's auc: 0.892208
[2000]  training's auc: 0.929145        valid_1's auc: 0.896409
[3000]  training's auc: 0.940704        valid_1's auc: 0.896585
Early stopping, best iteration is:
[2952]  training's auc: 0.940199        valid_1's auc: 0.896619
Fold 3 started at Thu May 23 21:59:01 2019
Training until validation scores don't improve for 1000 rounds.
[1000]  training's auc: 0.914758        valid_1's auc: 0.894037
[2000]  training's auc: 0.929022        valid_1's auc: 0.898242
[3000]  training's auc: 0.940578        valid_1's auc: 0.898285
Early stopping, best iteration is:
[2195]  training's auc: 0.931409        valid_1's auc: 0.898555
Fold 4 started at Thu May 23 22:00:21 2019
Training until validation scores don't improve for 1000 rounds.
[1000]  training's auc: 0.914961        valid_1's auc: 0.893687
[2000]  training's auc: 0.929198        valid_1's auc: 0.897151
[3000]  training's auc: 0.940755        valid_1's auc: 0.897263
Early stopping, best iteration is:
[2663]  training's auc: 0.937059        valid_1's auc: 0.897471
CPU times: user 30min 35s, sys: 11.4 s, total: 30min 46s
Wall time: 7min 58s
```

In [37]:

```
# Submitting the 5Fold LGB Predictions
submission_lgb = pd.DataFrame({
        "ID_code": test["ID_code"],
        "target": y_pred_lgb
    })
submission_lgb.to_csv('sub_lgb.csv', index=False)
```

This submission score 0.90038 on public leaderboard. (Almost top 9% in Public LB)

Conclusion

TODO -

1. H20 AutoML
2. Using XGBoost, Catboost
3. Ensembling, Stacking, Blending
4. Feature Removal

In [ ]:

```

```