Проект по курсу от Megafon

Выполнил: Кузнецов Виктор Владимирович

Данный проект выполнен на основании датасета представленного Мегафон.

Для приведении датасета к нормальному виду был применена библиотека Dask с сохранением в формате parquet с приведением типов данных.

Apache Parquet - это бесплатный и ориентированный на столбцы формат хранения данных с открытым исходным кодом экосистемы Apache Hadoop. Он похож на другие форматы файлов с столбчатым хранилищем, доступные в Hadoop, а именно RCFile и ORC. Он совместим с большинством фреймворков обработки данных в среде Hadoop. Он обеспечивает эффективные схемы сжатия и кодирования данных с повышенной производительностью для обработки сложных объемных данных.

Apache Parquet реализован с использованием алгоритма измельчения и сборки записей, который вмещает сложные структуры данных, которые могут быть использованы для хранения данных. Значения в каждом столбце физически хранятся в смежных ячейках памяти, и это столбчатое хранилище обеспечивает следующие преимущества:

Сжатие по столбцам эффективно и экономит место для хранения

Методы сжатия, специфичные для конкретного типа, могут быть применены, поскольку значения столбцов, как правило, имеют один и тот же тип

Запросы, извлекающие определенные значения столбцов, не должны считывать все данные строк, что повышает производительность

Различные методы кодирования могут быть применены к различным столбцам

Apache Parquet реализован с использованием фреймворка Apache Thrift, что повышает его гибкость; он может работать с рядом языков программирования, таких как C++, Java, Python, PHP и т.д.

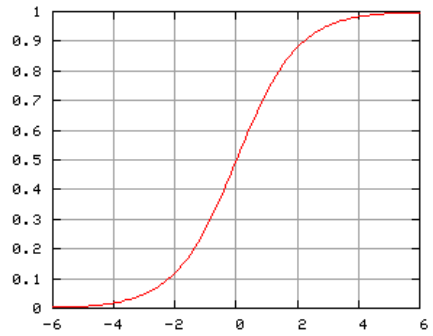
По состоянию на август 2015 года Parquet поддерживает фреймворки обработки больших данных, включая Apache Hive, Apache Drill, Apache Impala, Apache Crunch, Apache Pig, Cascading, Presto и Apache Spark.

Wiki

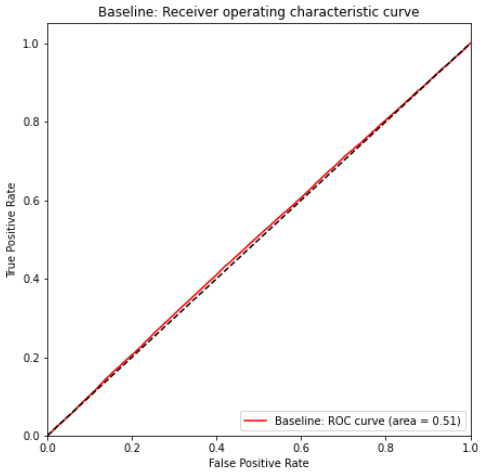
В данных состоят из:

1. Констант - 7
2. Бинарных значений - 23
3. Категорий - 63
4. Численных значений - 171

Для базовой модели взята Логическая регрессия.



Логистическая регрессия или логит-модель (англ. logit model) — это статистическая модель, используемая для прогнозирования вероятности возникновения некоторого события путём его сравнения с логистической кривой. Эта регрессия выдаёт ответ в виде вероятности бинарного события (1 или 0).



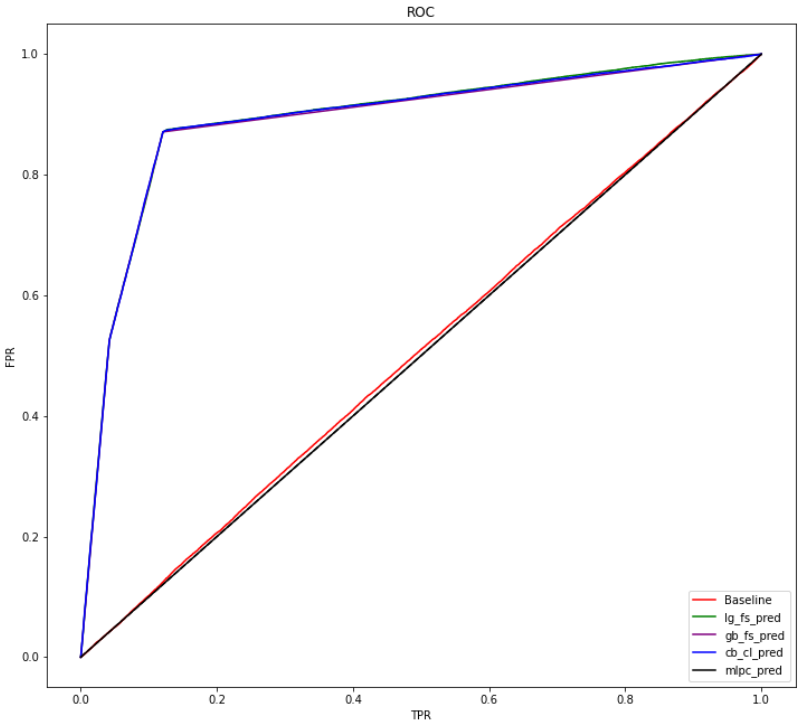
Для нахождения лучшей модели были проведены сравнения со следующими моделями:

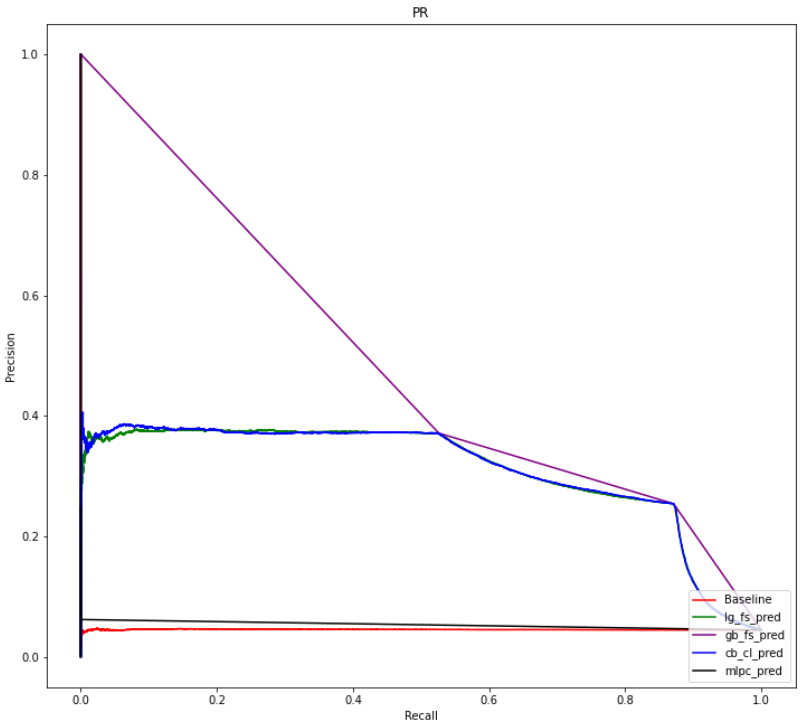
LogisticRegression

GradientBoostingClassifier

MLPClassifier

CatBoostClassifier


ROC
FPR
TPR
Baseline
lg_fs_pred
gb_fs_pred
cb_cl_pred
mlpc_pred


PR
Precision
Recall
Baseline
lg_fs_pred
gb_fs_pred
cb_cl_pred
mlpc_pred

На матрице решений были получены следующие значения для F1:

Базовая модель

macro avg precision = 0.49, AUC_PR = 0.046, AUC_ROC = 0.506

Confusion matrix

[[443978 60673]

 [20911   2960]]

LogisticRegression

macro avg precision = 0.70, AUC_PR = 0.310, AUC_ROC = 0.892

Confusion matrix

[[483865 20786]

 [11590 12281]]

GradientBoostingClassifier

macro avg precision = 0.49, AUC_PR = 0.488, AUC_ROC = 0.889

Confusion matrix

[[504651   0]

 [23871   0]]

CatBoostClassifier

macro avg precision = 0.70, AUC_PR = 0.311, AUC_ROC = 0.890

Confusion matrix

[[484238 20413]

 [11796 12075]]

MLPClassifier

macro avg precision = 0.49, AUC_PR = 0.054, AUC_ROC = 0.500

Confusion matrix

[[504643   8]

 [23870   1]]

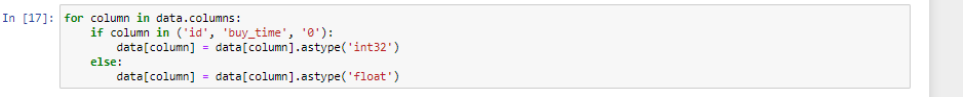
Была выбран модель для предсказания CatBoostClassifier в связи с тем что он давал точность на целевой метрике F1 70% и AUC ROC 89%

Принцип подхода заключается в следующем:

1. Преобразование в формат parquet. Парсил строку из csv файла.

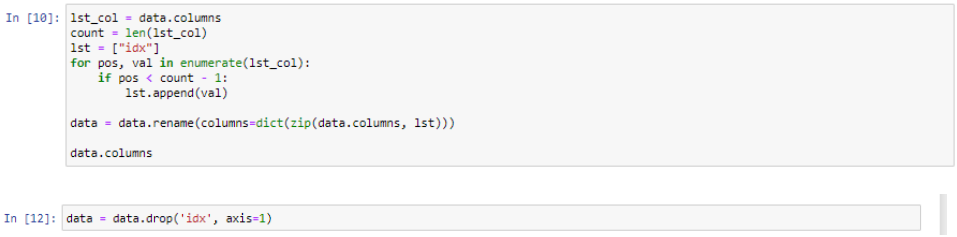
Объем данных 2.95 Гб

2. Преобразовал строки в численный тип данных.



Объем данных 1.58 Гб

3. После обнаружения смещения в данных. Сместил колонки на одну вправо и удалил лишную.



4. Объединил датасет и данные с фичами по полю id
5. Добавил данные по полю buy_time все возможные типы данных

In [12]:
```
def GetDateTime(df):
    df['hours'] = df['buy_time'].dt.hour
    df['days'] = df['buy_time'].dt.day
    df['day_week'] = df['buy_time'].dt.dayofweek
    df['day_year'] = df['buy_time'].dt.dayofyear
    df['day_month'] = df['buy_time'].dt.daysinmonth
    df['weeks_on'] = df['buy_time'].dt.week
    df['week_year'] = df['buy_time'].dt.weekofyear
    df['months'] = df['buy_time'].dt.month
    df['quartal'] = df['buy_time'].dt.quarter
    df['years'] = df['buy_time'].dt.year
    return df
```

In [20]:
```
test = data.merge(test, left_on='id', right_on='id').compute()
test = test.drop('buy_time_y', axis=1)
test = test.rename(columns={"buy_time_x":"buy_time"})
test['buy_time'] = pd.to_datetime(test['buy_time'], unit='s')
test = test.sort_values(by=['buy_time'], ascending=False)
test = GetDateTime(test)
test
```

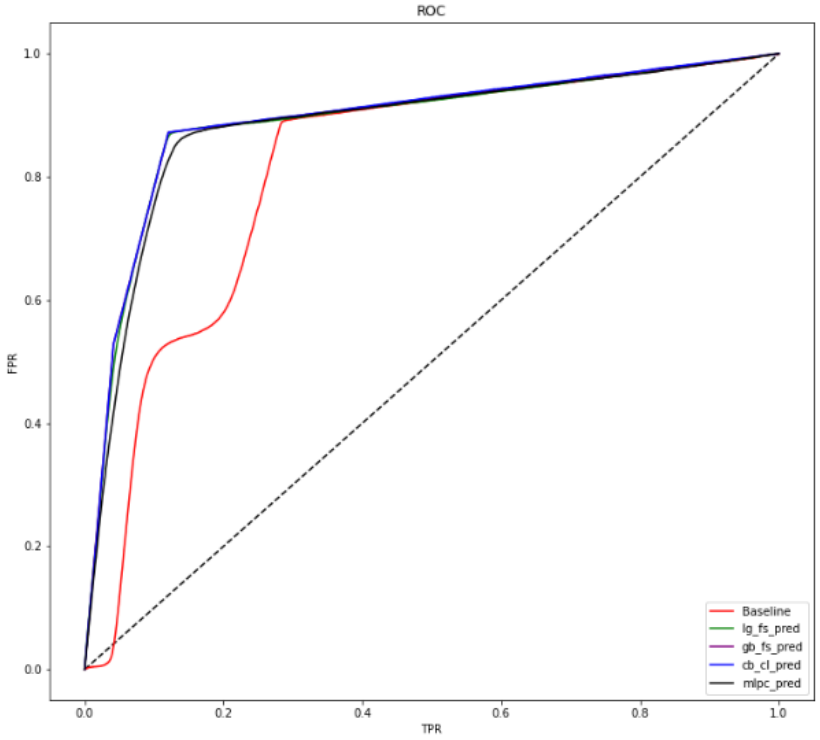
6. Разделил тенировочный датасет на тестовый и тренировочный
7. Разделил колонки на константы, бинарные, категориальные, численные и временные типы данных.
8. Выполнил нормализацию и приведение к стандартному распределению численных данных.

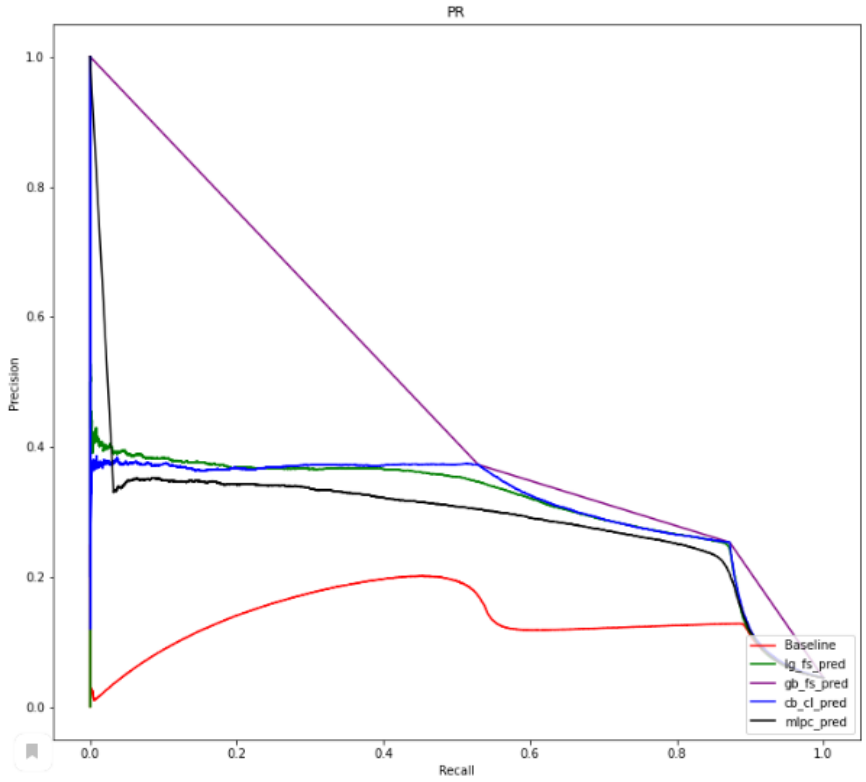
In [59]:
```
from sklearn.preprocessing import QuantileTransformer
def NormalazeToQuantile(train, test, cols):
    for col in cols:
        print(col)
        qt = QuantileTransformer(output_distribution='normal')
        train[col] = qt.fit_transform(train[col].values.reshape(-1,1))
        test[col] = qt.fit_transform(test[col].values.reshape(-1,1))
```

In [60]:
```
NormalazeToQuantile(train, test, f_numeric)
```

9. После анализа классификаторов был выбран CatBoostClassifier

```
Baseline: AUC_PR = 0.130
Baseline: AUC_ROC = 0.811
lg_fs_pred: AUC_PR = 0.308
lg_fs_pred: AUC_ROC = 0.887
gb_fs_pred: AUC_PR = 0.490
gb_fs_pred: AUC_ROC = 0.889
cb_cl_pred: AUC_PR = 0.309
cb_cl_pred: AUC_ROC = 0.890
mlpc_pred: AUC_PR = 0.291
mlpc_pred: AUC_ROC = 0.882
```





10. После снижения парогра чувствительности до 30% выбранная модель показала 70% точность

In [202]:
```
from sklearn.metrics import f1_score

for m in models:
    print(m[0])
    f1_ = f1_score(m[1], m[2] > 0.30, average='macro')
    print("macro avg f1 score = %.2f "  % float(f1_))
    cm = confusion_matrix(y_test, m[2] > 0.29)
    plot_confusion_matrix(cm, classes=["0", "1"], model_name=m[0])
    print()
```

```
Baseline
macro avg f1 score = 0.49
Confusion matrix, without normalization
[[500698   4151]
 [ 23609     91]]

lg_fs_pred
macro avg f1 score = 0.68
Confusion matrix, without normalization
[[487071  17778]
 [ 13538  10162]]

gb_fs_pred
macro avg f1 score = 0.49
Confusion matrix, without normalization
[[504849      0]
 [ 23700      0]]

cb_cl_pred
macro avg f1 score = 0.70
Confusion matrix, without normalization
[[483740  21109]
 [ 11158  12542]]

mlpc_pred
macro avg f1 score = 0.65
Confusion matrix, without normalization
[[488457  16392]
 [ 15596   8104]]
```