Прогнозирование RTO

Описание задачи

Предсказать суммарный PTO для каждого магазина на каждую неделю апреля 2021 года, основываясь на данных с 01.01.2019 по 28.03.2021

Анализ данных. Описание данных

Сбор данных не требуется, так как датасет был предоставлен на соревновании. Сами данные представляют 2 таблицы:

- Train - информация по PTO и траффику каждого магазина за указанный период. Размер - 818000 × 5

- Submission sample - пример требуемого предсказания. Напротив каждого id магазина PTO за каждую из 4 недель апреля.

Описание данных. Train

- Id: номер магазина (1000 уникальных)
- Timestamp: время в формате yyyy-mm-dd
- Region: регион, в котором находится магазин (54 уникальных)
- Traffic: метаданные traffic
- RTO: целевая переменная RTO

Предобработка данных

Таким образом, мы имеем дело с временным рядом:

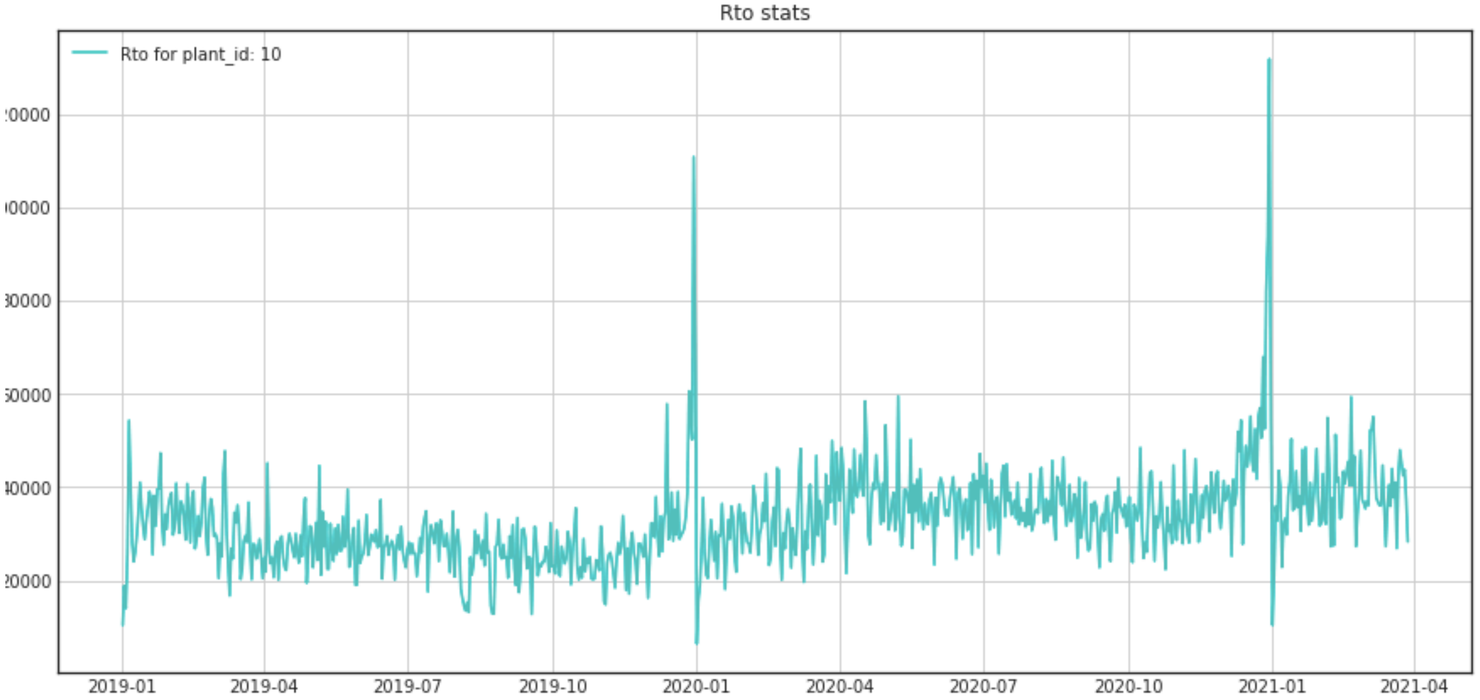
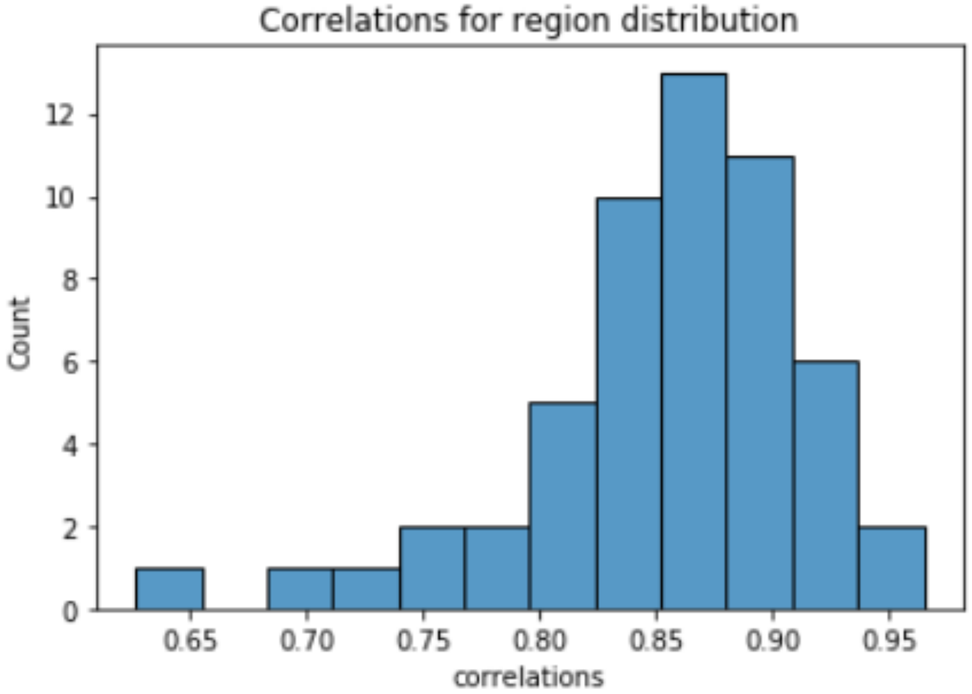


График зависимости RTO от времени для id=10

Предобработка данных

Посмотрим на корреляцию численных признаков traffic и rto.
Для всех данных значение составляет 0.85.
Если посмотреть на каждый регион по отдельности, то получим следующее распределение:

Ввиду высокой корреляции избавимся от признака traffic.



Эксперимент №1

Предсказание по регионам

Идея эксперимента

Идея: разбить датасет по регионам, для каждой группы провести fit-predict, агрегировать результат

```
for region in regions:
    dfs.append(train[train.region_nm == region])
```

Предсказание будем проводить для всего train датасета. Тест датасет будем формировать следующим образом:

```
def create_test_df(train_df):
    ids = train_df.id.unique()
    test_df = pd.DataFrame(columns=['id', 'year', 'month', 'day'])
    for _id in ids:
        data = {
                'id': [_id for i in range(30)],
                'year': [2021 for i in range(30)],
                'month': [4 for i in range(30)],
                'day': [i+1 for i in range(30)],
        }
        test_df = pd.concat([test_df, pd.DataFrame(data)])
    return test_df
```

Подготовка данных

Пример полученного тестового датасета:

	id	year	month	day
0	734	2021	4	1
1	734	2021	4	2
2	734	2021	4	3
3	734	2021	4	4
4	734	2021	4	5

	id	year	month	day
25	734	2021	4	26
26	734	2021	4	27
27	734	2021	4	28
28	734	2021	4	29
29	734	2021	4	30

Таким образом, посчитав считаем PTO для каждого датасета и агрегировав понедельно, получим итоговое предсказание. В качестве модели был выбран RandomForest с дефолтными параметрами.

Оценка эксперимента

Результат (private и public):

114347.54437	97408.89157

Вывод: сложно оценить скор, так как не известен benchmark. В дальнейшем предполагается добавить фич и подтюнить модели.

Эксперимент №2

Предсказание по регионам с дополнительными временными признаками

Идея эксперимента

Идея: выделить больше временных признаков: dayofweek, dayofmonth, weekofyear:

```
In [38]: train.timestamp = pd.to_datetime(train.timestamp)
         train['year'] = train.timestamp.dt.year
         train['month'] = train.timestamp.dt.month
         train['day'] = train.timestamp.dt.day
         train['dayofweek'] = train.timestamp.dt.dayofweek
         train['dayofmonth'] = train.timestamp.dt.day
         train['weekofyear'] = train.timestamp.dt.weekofyear
```

Все остальное сделать по аналогии.

Оценка эксперимента

Результат: 122100.39340 105955.19531

Вывод: скор существенно ухудшился. Учитывая это изменение, а также разницу между моими результатами и результатами моих коллег стали закрадываться сомнения в общей правильности выбранного подхода. Как раз в это время появилась лекция от Святослава, где был предложен существенно другой способ

Эксперимент №3

Benchmark (weekly granulation, new features, validation)

Идея эксперимента

Так как результат должен быть представлен в виде суммарного PTO по неделям, то проведем понедельную агрегацию и в тестовом датасете:

```
: def make_week_frame(df, cols_to_keep, target_metrics, agg_func='sum'):
    """
    Создание фрейма с недельной гранулярностью. Среднее значение метрики
    с понедельника по воскресенье включительно сохраняется в дату воскресенья.
    """
    assert agg_func in ['sum', 'mean']
    df['timestamp'] = pd.to_datetime(df['timestamp'])
    week_df = (df
               .assign(period=pd.PeriodIndex(df['timestamp'], freq='W-Sun'))
               .groupby(['id', 'period'] + cols_to_keep)
               .agg({metric: agg_func for metric in target_metrics}).reset_index())
#     week_df['timestamp'] = \pd.to_datetime((week_df['year'] * 100 + week_df['week']).ast
#                                            format = '%Y%W%w')
    week_df['timestamp'] = week_df['period'].dt.to_timestamp(how='end')
    week_df['timestamp'] = week_df['timestamp'].dt.normalize()
#     week_df['week'] = week_df['timestamp'].dt.weekofyear

    return week_df
```

```
: week_df = make_week_frame(df, ['region_nm'], ['rto', 'traffic'], 'sum')
```

Идея эксперимента

Помимо этого, добавим 2 признака: значение показателя n_weeks_before_rto и разница значения показателя на данной неделе и на предыдущей неделе n_Weeks_before_rto_diff (n = 1, …4).

```
week_df = week_df.sort_values(['timestamp', 'id'])
for i in range(4):
    week_df[f'{i+1}_Weeks_before_rto'] = week_df.groupby(['id'])['rto'].shift(i+1)
    week_df[f'{i+1}_Weeks_before_rto_diff'] = week_df.groupby(['id'])['rto'].diff(i+1)

week_df = week_df.dropna().reset_index().drop(columns=['index'])
for i in range(4):
    week_df[f'{i+1}_Weeks_after_rto'] = week_df.groupby(['id'])['rto'].shift(-(i+1))
```

Поскольку в датафрейме последняя неделя заканчивается 28 марта 2021 года, то предсказания по последним 1000 записям фрейма на 1, 2, 3, 4 недели будет являться искомым результатом.

Оценка эксперимента

Результат (модель - Linear Regression, Normalize:True)

55212.85891	50542.60284

Вывод: Скор значительно улучшился. В дальнейшем будем оптимизировать этот подход.

Эксперимент №4

Random Forest

Идея и оценка эксперимента

Будем использовать более сложную модель, на этот раз - RandomForestRegressor. Для подбора оптимальных параметров будем пользоваться функцией кросс-валидации для временных рядов.

На каждом сплите мы обучаемся данных прошлого, а тестируемся на данных будущего относительно некоторого момента времени, что позволяет избежать подглядывание в будущее при обучении

```
def TimeSeriesCV(df, model_func, metric_func, params, n_splits=5):
    tscv = TimeSeriesSplit(n_splits=n_splits, test_size=df['id'].nunique() * 4)
    error_dict = {i: [] for i in range(1, 5)}
    for train, test in tqdm_notebook(tscv.split(df.dropna().values)):
        models_dict = {i: model_func(**params) for i in range(1, 5)}
        for i in range(1, 5):
            train_data, train_target, test_data, test_target = make_train_target_split(
                week_df, i, train, test)
            models_dict[i].fit(train_data, train_target)
            predictions = models_dict[i].predict(test_data)
            error_dict[i].append(metric_func(test_target, predictions))
    return error_dict
```

Оценка эксперимента

Результат (модель - Random Forest, max_depth: 20, n_estimators': 100)

45729.19992	42461.31190

Вывод: Скор значительно улучшился. Попробуем еще одну модель Random Forest

Эксперимент №5

Random Forest с другими параметрами

Оценка эксперимента

Результат (модель - Random Forest, max_depth: 20, n_estimators': 500)

45702.76963 42137.15898

Вывод: Скор улучшился, но незначительно. Перейдем к другим моделям, так как увеличивая эти параметры, можно переобучиться

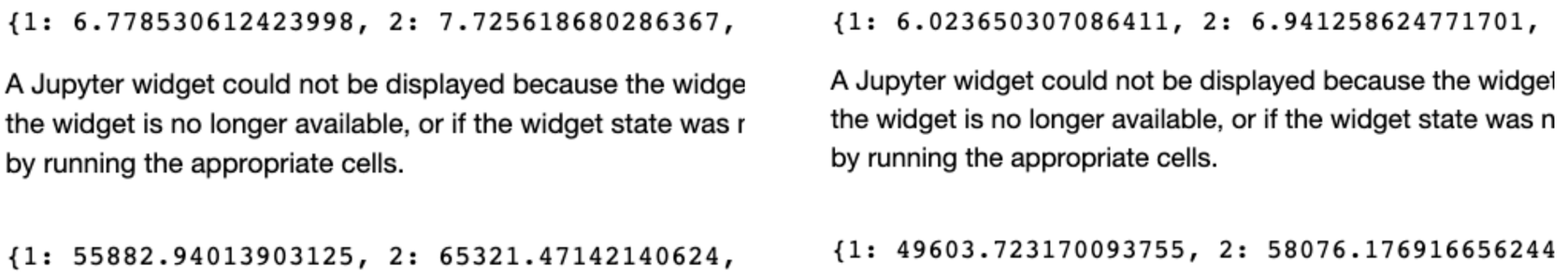
Эксперимент №6

XGBoost

Идея эксперимента

Будем использовать более сложную модель - XGBoost. Во-первых, на кросс-валидации определимся с типом booster: gbtree или gblinear. Лучшие результаты стабильно показывал gblinear, так что свободные параметры будем тюнить для него

MAPE и MAE для gbtree и gblinear соответственно на первых 2 фордах:



Идея эксперимента

Подберем параметры alpha и lambda, отвечающие за регуляризацию. Для этого создадим сетку из параметров, рассчитаем скоры и, усреднив по фолдам, найдем оптимальные параметры. Это alpha=0.5, lambda=0.7

```
cv_results = {}
for a in tqdm([0, 0.1, 0.3, 0.5, 0.7, 1]):
    for l in [0, 0.1, 0.3, 0.5, 0.7, 1]:
        params = {'booster': 'gblinear', 'n_estimators': 100, 'alpha': a, 'lambda': l}
        model_func = XGBRegressor
        cv_scores = TimeSeriesCV(week_df, model_func, mean_absolute_percentage_error, params)
        mean_cv_scores = {k: np.mean(v) for k, v in cv_scores.items()}
        print(mean_cv_scores)
```

Оценка эксперимента

Результат:

41197.02216	37522.99658

Вывод: Скор значительно улучшился. Можно дополнительно поработать над n_estimators

Эксперимент №7

Использование XGBoost с увеличенным n_estimators

Идея эксперимента

Для RandomForest удалось уменьшить скор путем увеличения n_estimators с 100 до 500. Посмотрим на поведение XGBoost модели при этом

Оценка эксперимента

Результат:

41203.52244	37526.30338

Вывод: скор незначительно уменьшился. Видимо, для улучшения показателей нужно провести дополнительный feature engineering.

Эксперимент №8

Добавление годовой сезонности

Идея и оценка эксперимента

Попытаемся учесть годовую сезонность: добавим признак rto_last_year, который будет показывать, какой PTO был год назад.

```
week_df['Year_before_rto'] = week_df.groupby(['id'])['rto'].shift(53)
```

Результат:

41194.67829	37521.57841

Вывод: Скор уменьшился относительно лучшей попытки, имеет смысл оставить фичу.

Итоги

- **Впервые удалось осознанно поработать с задачей предсказания временных рядов. Были рассмотрены различные идеи и методы (агрегация, заглядывание в будущее, TimeSeriesCV)**

- **С помощью методологии CRISP была проведена последовательная работа по улучшению скора, а также зафиксированы все полученные численные результаты**

- **Не было сгенерировано большое количество дополнительных признаков, ввиду несильного понимания предметной области**

Дальнейшие планы

- **Улучшить понимание всяческой теории, связанной с временными рядами**
- **Применить больше методов по обработке и анализу ВР**
- **Научиться генерировать больше релевантных признаков**