Отчет по задаче “Sustainable Industry: Rinse Over Run” на платформе drivendata.

ФИО: Кузьмин Никита.
Группа: 208.
Ник: paniquex.
Финальный результат: 0.5041.

Содержание:

Постановка задачи:

Цель соревнования - предсказать количество загрязненности, которое будет возвращено на последней стадии промывки пищевого и промышленного оборудования. Основываясь на данных, можно описать цель соревнования в других терминах: необходимо предсказать суммарное количество загрязнений для каждого процесса, которое рассчитывается, как **sum**(**max**(0, **return_flow**) * **return_turbidity**), когда **target_time_period**=True.

Эту задачу можно рассмотреть с разных сторон:

1. Как задачу прогнозирования временного ряда, для которой можно построить различные модели (**ARIMA**, **SARIMA**, **LSTM**).
2. Как задачу регрессии, "сжимая" временной ряд с помощью агрегирующих функций для каждого процесса.
3. Как задачу двойной регрессии, целью которой является предсказание **return_flow** и **return_turbidity** по отдельности для каждого процесса, когда **target_time_period**=True. Ответ на задачу получается по формуле, представленной выше.

   Для того, чтобы достичь цели вводится функция ошибки

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\frac{|\hat{y}_i - y_i|}{max(|y_i|, 290000)}$$

   , которую нужно **минимизировать**.

Из-за нехватки времени выберу самое простое - попробую сжимать ряд и решать задачу регрессии для **final_rinse_total_turbidity_liter**(тем более, что есть бэнчмарк от которого можно отталкиваться).

Ход решения:

1. Попытался улучшить бэнчмарк, варьируя параметры **RandomForestRegressor** и смотрел качество на кросс-валидации - результат вышел хуже - 1.4687. Скорее всего, переобучил модель, поэтому она настроилась на тренировочную выборку и давала результат лучше кросс-валидации, чем бэнчмарк.

2. В бэнчмарке организаторы отбрасывают 20% тренировочных данных. Попробую оставить эту часть, так как их обоснование этого шага меня не убедило. При этом еще использовал **GridSearchCV** для того, чтобы подобрать параметры модели **RandomForestRegressor**. Для оценки качества модели использовал кросс-валидацию для временных рядов - **TimeSeriesSplit(UPD:** осознал, что это бесполезное решение, так как после агрегации достаточно использовать только **KFold**) и написал модифицированную функцию **mean_abs_percentage_error**, которая приведена в описании задания. С помощью этой функции создал **scorer** и использовал его для параметра **accuracy** в **GridSearchCV**. Кроме кросс-валидации из тренировочной выборки выделил отложенную (20%), на которой снова проверял качество модели после кросс-валидации.
   Хоть и на кросс-валидации и отложенной выборке эта модель показала результат намного лучше, чем бэнчмарк (1.1 против 1.4), но она оказалась не самой хорошей для тестовой выборки - результат 1.8132. Тут два возможных вывода:
   - 1) Либо те 20% данных, которые убрали организаторы играют ключевую роль.
   - 2) Либо я снова переобучил модель или где-нибудь накосячил в коде.

   Нужно будет перепроверить.

3. Попробую выбросить 15% тренировочных данных и проведу подбор параметров для **RandomForestRegressor** аналогично пункту 2) для того, чтобы определить действительно ли так важен выброс этих 15%(20%). Но есть особенность - те модели, которые на кросс-валидации дают лучшие результаты; при проверке на отложенной выборке результаты этих моделей являются достаточно плохими. Параметр max_leaf_nodes выбрал по результатам предыдущих тестов - при возрастании max_leaf_nodes, ошибка убывала.
   Результат все-таки получилось улучшить - **1.3958**, а у бэнчмарка - 1.4637.

4. Попробовал поработать с линейными моделям - **SGDRegressor, Ridge, LinearRegression. SGDRegressor** давал слишком хороший результат на отложенной выборке и кросс-валидации ~ 0.4-0.6, но я не стал отправлять это решение, так как проверил среднюю разность с предыдущими значениями целевой переменной, она оказалась значительной. Возможно, в следующий раз я все-таки

решусь отправить эту модель. **Ridge** получилось довести только до 2.1 на отложенной выборке с помощью **StdScaler** для шкалирования числовых признаков + **GridSearchCV** для подбора параметров. **LinearRegression** не дала хороших результатов.
Продолжил улучшать **RandomForestRegressor**. Добавил **новых признаков**: сумму для каждого процесса его признаков + среднее абсолютное отклонение. Для каждого процесса беру среднее значение каждого из признаков за предыдущие 25 моментов времени. Еще раз с помощью **GridSearchCV** нашел оптимальные параметры и частично их подкорректировал. На отложенной выборке ошибка составила ~ 0.87. А на тестовых данных результат получился - **1.0566**.

5. Теперь взял для каждого процесса среднее значение каждого из признаков за предыдущие 110 моментов времени. Для анализа моделей написал функцию **models_validation(X, y, models, folds)**, которая оценивает качество на кросс-валидации и на отложенной выборке. С помощью нее протестировал некоторые модели(**ElasticNet, Lasso, Ridge..**) с базовыми параметрами для того, чтобы понять то, для какой модели буду подбирать параметры(эта модель может оказаться не самой лучшей при оптимальных параметрах из представленных, но попробовать стоит). Результаты оказались неоднозначные, если не рассматривать модели, похожие на **RandomForestRegressor**. Но, к сожалению, ни одна из них не подошла достаточно близко к результатам **RandomForestRegressor**. Проверил с произвольными параметрами нейронную сеть **MLPRegressor**. Можно будет ближе к концу соревнования попробовать развить её. Отправил **RandomForestRegressor**. Результат - 1.4559.

6. Попробуем рискнуть и оставить информацию о процессах, которые находились в заключительной фазе(**'final_rinse'**). Хоть и в тестовой выборке информации об этой фазе совсем нет, но, возможно, она нам как-то поможет при предсказании, ведь в предобработке мы усредняем все признаки по фазам. Такой информации оказалось даже больше, чем о процессах в других фазах. Проверим уже настроенный **RandomForestRegressor** - превосходный результат на кросс-валидации и отложенной выборке. На других моделях улучшился, но незначительно. Отправляем **RandomForestRegressor**. Результат был ожидаемым - **2.2702**.

7. Снова удаляем информацию о заключительной фазе. Попробуем посмотреть на признаки **'return_turbidity'** во всех фазах и **'return_temperature'** на последней(возможно, чем меньше мутность, тем больше температура на выходе, либо наоборот.). Полезной информации не дали эти графики. Нужно будет отдельно смотреть на них за короткие промежутки времени. Используем **RFECV** для того,

чтобы отобрать важные признаки для моделей. Для линейных - сначала прошкалируем данные с помощью **MinMaxScaler**. После этого модель **Ridge** показала результат значительно лучше предыдущего, но все-таки несравнимый с **RandomForestRegressor**. Попробовал новую модель - **KNeighboursRegressor**. Без подбора параметров дает качество, сравнимое с **RandomForestRegressor**. Нужно подобрать параметры с помощью **GridSearchCV**. Качество стало еще лучше, отправим модель. Результат - 1.4096. Хуже, чем лучший, но все-таки обошел бэнчмарк.

8. В этом шаге я просто попробовал слегка изменить параметры и, сравнивая с результатами **RandomForestRegressor** улучшить качество модели. Результат - 1.3542. Опять же превосходит бэнчмарк, но уступает лучшему решению.

9. Попробовал реорганизовать данные в **prep_time_series_features** - сделал так, чтобы значения признаков одного процесса, но для разных фаз считались, как отдельные признаки, чтобы не было агрегации по фазам. Матрица получилась с 400+ признаками и очень разреженная, заполнил **NaN** нулями для начала. После этого посмотрел на результаты моделей, но они стали только хуже. Попробовал заполнить средними по признаку(было очевидно, что не сработает, но попытаться стоило). Затем "-1" - плохо. Возможно, модели не справились с задачей из-за слишком большой разреженности матрицы(**UPD1**: понял, что нужно было в **RandomForestRegressor** и ему подобных параметр **max_features** сделать равных максимальному количеству признаков в данных и постепенно уменьшать. Есть вероятность, что это поможет. Попробую позже.).
   Не стану отправлять это решение, надеюсь, что получится его доработать или убедить себя в его непригодности (**UPD2**: дальше описал, что нашел ошибку и исправил её. Качество улучшилось). Раз уж так не пошло, то попробую **XBoostRegressor**(не очень мне нравится эту модель использовать, но в любом случае посмотреть на её результат стоит). Частично подобрал параметры модели с помощью **GridSearch** и проверил качество. Я практически не сомневался, что результаты будут очень хорошими. Интересно, что **XGBoost** не любит, когда в названиях признаков содержатся "[", "]", "<", ">". Пришлось их чуть-чуть подправить. Отправил решение - **0.5041** **и 61 место**. Все-таки попробую дальше доработать **XGBoost**, чтобы попробовать попасть в топ-50. Возможно, протестирую эту модель на разреженной матрице.

ИДЕЯ: взять часть данных из тестовых, которая расположена при **target_time** == False (спойлер - плохая идея).

10. Попробовал изменить настроить параметры **XGBoost**, но это не привело к хорошему результату на тестовой выборке. Результат - 0.5262

11. Добавил **новых признаков** - стандартное отклонение, медиану по последним 60 измерениям по каждому признаку + логарифм от абсолютного значения для каждого признака + 0.1 и инвертированное преобразование **CoxBox**. Попробовал использовать много новых моделей - **catboost**, **lgbm**, перенастроил **XGBoost**. **Catboost** и **lgbm** особо не настраивал, поэтому результат они показали плохой. На кросс-валидации и на отложенной выборке качество у **XGBoost** стало на 0.3 лучше, но при отправке этого решения получился результат - 0.5553.

12. Решил снова попробовать взять полную тренировочную выборку и протестировать на **XGBoost**. Эта модель показала отличное качество на кросс-валидации(0.6) и на отложенной выборке(0.39). Но на тестовых данных результат - 0.7.

ЗАМЕЧАНИЕ: Если брать отдельные значения ошибки на каждом фолде кросс-валидации, а не их среднее, то можно заметить достаточно большое отклонение на 1-2 фолдах(из 5). Пока не могу понять то, с чем это могло быть связано. (**UPD:** хоть и организаторы модифицировали метрику MAPE, чтобы ошибки на маленьких значениях сильно не штрафовали, но этого все-таки недостаточно и модель штрафуют больше на маленьких значениях).

13. Тут я попробовал усреднить результаты 9, 10, 11 и нового настроенного **XGBoost**. Но, результат все-таки не улучшился - 0.5217. Возможно, не стоило использовать непроверенный на тестовых данных **XGBoost**.

ИДЕИ:

- разбивать на контрольную и тренировочную выборки по-другому. Чтобы распределение процессов по фазам в контрольной выборке совпадало с соответственным распределением тестовой выборки. (Это так и не реализовал)
- Убрать наблюдения из тренировочной выборки, у которых **object_id** такой, которого нет в тестовой выборке. (А это попробовал далее)

14. Добавил **медиану** и **дисперсию** по заключительным 60 измерениям для каждого признака. Попробовал настроить **lgbm** с помощью **GridSearch**, но результат стал только хуже. Отправил **XBoost** и качество - 0.5184.

15. Попробовал вернуться к своей организации данных, которая считала заданные характеристики для каждой фазы в отдельности(описано в **п. 9**) и нашел в ней баг, который приводил к плохим результатам на кросс-валидации. Исправил его и протестировал на **XGBoost** - качество на кросс-валидации и на отложенной выборке стало лучше, но буквально на +-0.05. Настроил **XGBoost** - результат стал еще лучше. Отправил эту модель - 0.6027.

ТЕСТ: Промежуточно попробовал воспользоваться библиотекой **keras**, построить и обучить нейронную сеть. Просто экспериментировал и получил результат на отложенной выборке - **0.67**. Интересно, что было бы при грамотном подходе.

16. Вернемся к лучшему решению (9) и попробуем протестировать на разреженной матрице + добавим пару признаков(**tail50mean**, **tail30std**, **tail30median**). Модифицировал функцию **creature_feature_matrix(df, flag)**. Теперь, в зависимости от параметра flag выбирается либо агрегация временных рядов по процессам, либо по процессам и фазам. По кросс-валидации выяснил, что лучше оставить только **tail50mean**. Сравнивал результат при различном заполнении **NaNs** и пришел к выводу, что при замене на **среднее по признаку** результат получается лучше. Мне не понравилось это, так как распределение процессов по фазам в тренировочной выборке и тестовой различаются - может быть переобучение. Сравнил средние по признакам у тестовой и тренировочной выборок. Они слабо различаются. Решил рискнуть два раза - оставить заполнение средним значением и взять полную тренировочную выборку, а не 85%. Риск не оправдался - 0.6622. (**UPD:** нашел в баг в подготовке данных к **XGBoost,** из-за которого удалял признаки, содержащие пайплайны для каждого процесса).
17. Не нравится мне то, что берется только среднее по последним измерениям. Изменю это на несколько признаков, которые берут среднее по N измерениям в различные промежутки времени. На кросс-валидации качество стало лучше, как и на отложенной выборке. Далее, я решил добавить признак, показывающий **продолжительность каждого процесса** и всё-таки мне кажется, что **object_id** тоже дает полезную информацию о процессе, которая влияет на наше предсказание, поэтому его тоже включаю. Экспериментировал с введением различных признаков, например инвертированное преобразование **Бокса-Кокса**, просто **среднее**, **дисперсию от логарифма** каждого признака. Настраивал параметры **XGBRegressor**. Результат получился - 0.5663.

Финальный метод и результат:

Лучшее качество показал метод, описанный в **п. 9**. Но больше я верил в то, что результат лучше будет у решения из **п. 15, 16** и похожие на них, так как оно мне кажется более обоснованным - нем приводится детальная информация по каждой фазе у процесса. Производится “чистка” тренировочных данных от процессов с **object_id,** которых нет в тестовой и другие преобразования. Возможно, не стоило брать так много признаков, а с помощью **feature_importance** у лесов отобрать их, а затем уже разбить по фазам или же результат хуже из-за бага, найденного слишком поздно.

По итогам соревнования получилось достичь таких результатов (ранг указан на 11:00, 01.03.19, поэтому финальный ранг может измениться, но по ошибке можно восстановить позицию).

Что можно было сделать еще:

1. Разобраться с различными моделями прогнозирования временных рядов: **ARIMA**, **SARIMA**, **LSTM**. Для выбора признака, по которому нужно прогнозировать моделью **ARIMA** можно воспользоваться **feature_importance** у **GradientBoostingRegressor**.
2. Попробовать свести одну задачу регрессии к двум другим - предсказание **return_flow** и **return_turbidity** на заключительной фазе промывки. Затем из этих данных получит финальный ответ.
3. Я совсем не работал с выбросами в этой задаче (хотя в задании за прошлый семестр уделил им достаточное внимание и это привело к хорошему результату).
4. Как я уже писал в ходе решения: можно попробовать разбить на контрольную и тренировочную выборки по-другому, чтобы распределение процессов по фазам в контрольной выборке совпадало с соответственным распределением тестовой

выборки. Это бы дало более точные результаты при проверке моделей. (Это так и не реализовал)

5. Взять среднее по 3 лучшим проверенным решениям, а не так, как я сделал в **п. 13** (Этот подход помог улучшить результат в задаче кредитного скоринга из прошлого семестра).
6. Уменьшить количество признаков без помощи **RFECV.** Так как **RFECV** не изменяло качество модели (скорее всего, считало, что оптимальное количество признаков = максимальному количеству). Это могло вызвать переобучение.
7. Увеличить количество признаков, например, с помощью **PolynomialFeatures** и отобрать из них только самые важные.
8. Разобраться с блендингом (так как сейчас только базовое понимание) и протестировать эту модель.