Дата последней сборки документа:
21 января 2024 г. в 02:47

Заметки по прогнозированию временных рядов

Содержание

1. Вводные замечания

Возможно самой простой формой прогнозирования является *скользящее среднее*. Часто скользящее среднее используется в качестве *метода сглаживания*, чтобы найти более гладкую линию для данных с большим количеством вариаций [2]. Каждую точку данных можно описать средним значением n окружающих точек данных, где n – обозначает размер окна. При размере окна 10 мы опишем точку данных так, чтобы ее значения представляло собой среднее значение 5 значений, предшествующих точке, и 5 значений, следующих после точки. При прогнозировании будущие значения рассчитываются как среднее n предыдущих значений, поэтому при размере окна 10 речь будет идти о среднем значении 10 предыдущих значений.

Суть сглаживания скользящим средним заключается в том, что вам нужен большой размер окна, чтобы сгладить шум и уловить фактический тренд, но при большом размере окна ваши прогнозы будут значительно отставать от тренда, поскольку вы будете использовать все более ранние наблюдения для вычисления среднего.

Идея экспоненциального сглаживания заключается в том, что мы применяем *экспоненциально уменьшающиеся веса* к усредняемым по времени значениям, придавая недавним значениям больший вес, а большее ранним значениям – меньший.

Хольт усовершенствовал технику экспоненциального сглаживания, чтобы она позволяла учитывать тренд, и назвал ее *двойным экспоненциальным сглаживанием* (double exponential smoothing). А в сотрудничестве с Винтерсом Хольт добавил поддержку сезонности в 1960 году, и техника

получала название *тройного экспоненциального сглаживания* (экспоненциальное сглаживание Хольта-Винтерса).

Недостатком этих методов прогнозирования является то, что *они могут медленно приспосабливаться к новым тенденциям*, и поэтому прогнозируемые значения отстают от реальности – *они плохо работают, когда требуется более длительные горизонты прогнозирования*, и существует мнрожество гиперпараметров для настройки, что может быть трудным и очень времязатратным процессом [2, стр. 14].

ARIMA и модель Бокса-Дженкинса часто используется как вазимозаменяемые термины, хотя технически модель Бокса-Дженкинса относится к методу оптимизации параметров для ARIMA-модели.

ARIMA – это аббривиатура трех понятий: *автогрессия*, *интегрированное* и *скользящее среднее*. *Авторегрессия* означает, что модель использует зависимость между точкой данных и некоторым количеством запаздывающих точек данных (лагов). То есть модель делает прогноы на основе предыдущих значений. Это похоже на предсказание того, что завтра будет тепло, потому что до сих пор было тепло всю неделю.

Интегрированное означает, что вместо применения любой исходной точки данных используется разница между этой точкой данных и некоторой предыдущей точкой данных. По сути, это означает, что мы преобразуем ряд значений в ряд изменений значений. Интуитивно это предпалагает, что завтра будет более или менее такая же температура, как сегодня, потому что температура всю неделю сильно не менялась.

Проблема с ARIMA-моделями заключается в том, что они не поддерживают сезонность или данные с повторяющимися циклами, такими как повышение температуры днем и падение ночью или повышение летом и падение зимой. SARIMA (Seasonal ARIMA) бала разработана для преодоления данного недостатка. Подобно нотации ARIMA, нотация для модели SARIMA представляет собой $SARIMA(p,d,q)(P,D,Q)m$, где P – порядок сезонной авторегрессии, D – порядок сезонного дифференциирования, Q – порядок сезонного скользящего среднего, а m – периодичность (количество периодов в полном сезонном цикле).

VARIMA для случаев с несколькими временными рядами в качестве векторов. FARIMA – дробная ARIMA и ARFIMA – дробная проинтегрированная ARIMA. Последние две включают дробную степень дифференциирования, обеспечивающую длительную память в том смысле, что наблюдения, удаленные друг от друга с точки зрения времени, могут иметь несущественные зависимости.

SARIMAX – сезонная ARIMA, где X означает экзогенные или дополнительные переменные, добавленные в модель, например добавление прогнгоза осадков в модель прогнозирования температуры.

ARIMA обычно показывает очень хорошие результаты, но недостатками являются сложность подбора параметров и необходимость тщательного разведочного анализа. Настройка и оптимизация моделей ARIMA часто требуют значительных вычислительных ресурсов, а успешные результаты могут зависеть от навыков и опыта прогнозиста.

Когда дисперсия данных не является постоянной во времени, ARIMA-модели сталкиваются с проблемами [2, стр. 16]. В экономике и финансах непостоянство дисперсии может быть обычным явлением. Для решения этой проблемы были разработаны модели *авторегрессии условной гетероскедостичности* (Autoregressive Conditional Heteroscedasticity – ARCH). Гетероскедостичность

– это способ сказать, что дисперсия или разброс данных не являются постоянными повсюду, а противоположенным термином является гомоскедостичность.

Тим Боллерслев и Стивен Тейлор в 1986 году дополнили модель ARCH компонентой *скользящего среднего*, предложив модель Generalized ARCH, или GARCH. Когда колебания случайны, может быть полезна GARCH.

Обе модели ARCH и GARCH не могут обрабатывать ни тренд, ни сезонность, хотя на практике часто для работы с *сезонными колебаниями* и *трендом* временного ряда применяется ARIMA-модель, а затем для моделирования *ожидаемой дисперсии* используется ARCH-модель.

Градиентный бустинг сейчас стал популярен для прогнозирования временных рядов. Следует помнить, что деревья не умеют экстраполировать! Если временной ряд содержит тренд, можно попробовать детрендинг. Мы удаляем из ряда тренд, предварительно спрогнозированный линейной моделью. На полученных остатках обучаем градиентный бустинг и получаем прогнозы, к ним добавляем тренд. Преимуществом градиентного бустинга является способность фиксировать нелинейные зависимости и взаимодействия высокого порядка.

2. Математический аппарат Facebook Prophet

2.1. Библиотека Facebook Prophet

Используется модель разложимых временных рядов (Harvey & Peters, 1990) с тремя основными компонентами:

- тренд,
- сезонность,
- праздники

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t,$$

где $g(t)$ – функция тренда, моделирующая непериодические изменения значений временного ряда, $s(t)$ представляет собой периодические изменения (например, еженедельную и ежегодную сезонность), а $h(t)$ – представляет собой эффекты праздников, которые возникают в течение одного или нескольких дней. Член ошибки ε_t представляет собой любые случайные возмущения, которые не учитываются моделью.

Эта спецификация аналогична *обобщенной аддитивной модели* (GAM) (Hastie& Tibshirani, 1987), классу регрессионных моделей с потенциально нелинейными сглаживаниями, применяемыми к регрессорам.

Выбор в пользу GAM имеет то преимущество, что она легко декомпозируется и допускает включение новых компонент по мере необходимости, например при выявлении нового источника сезонности. GAM также обучается очень быстро, либо с помощью бэкфиттинга, либо с помощью L-BFGS (Byrd et al., 1995) (мы предпочитаем последнее), чтобы пользователь мог интерактивно изменять параметры модели.

По сути, мы формулируем проблему прогнозирования как задачу подгонки кривой, которая внутренне отличается от моделей временных рядов, поскольку те явно учитывают структуру временной зависимости в данных. Хотя мы отказываемся от некоторых важных преимуществ использования генеративной модели типа ARIMA, выбор в пользу *GAM* обеспечивает ряд практических преимуществ [2, стр. 24]:

- гибкость, заключающуюся в том, что мы можем *легко учесть сезонность с несколькими периодами* и позволить аналитикам выдвинуть разные предположения о трендах,
- в отличие от моделей ARIMA, *измерения не должны находиться на одинаковом расстоянии друг от друга* и нам не нужно интерполировать пропущенные значения, возникшие, например, по причине удаления выборосов,
- обучение выполняется очень быстро, что позволяет аналитику интерактивно исследовать большое количество спецификаций модели,
- прогнозная модель имеет легко интерпретируемые параметры, которые аналитик может изменить согласно выдвинутым предположениям относительно прогнозов.

2.1.1. Модель тренда

В Facebook реализованы две модели тренда:

- кусочно-логистическая модель роста с насыщением,
- кусочно-линейную модель.

Нелинейные рост с насыщением Для прогнозирования роста основной компонентой процесса генерации данных является модель, предсказывающая, как выросла численность население и как она будет расти дальше. Моделирование роста в Facebook обычно похоже на рост населения в естественных экосистемах, где наблюдается нелинейный рост, который насыщается при предельной пропускной способности. Предельной пропускной способностью для количества пользователей Facebook в определенном регионе может быть количество людей, имеющих доступ к сети Интернет. Такой рост обычно моделируется с помощью логистической модели роста, которая имеет вид

$$g(t) = \frac{C}{1 + \exp\left(-k(t-m)\right)},$$

где C – верхний порог (пропускная способность), k – скорость роста, m – параметр смещения, позволяющий сдвигать функцию вдоль оси времени.

Предельная пропускная способность непостоянна, поскольку количество людей в мире, которые имеют доступ к Интернету, увеличивается. Таким образом, мы заменяем фиксированную пропускную способность C на изменяющуюся во времени пропускную способность $C(t)$. Во-вторых, скорость роста непостоянна.

Мы включаем изменения тренда в модель роста, явно определяя точки изменения (change points), в которых скорость роста может измениться.

Тогда кусочно-логистическая модель роста принимает вид [2, стр. 26]

$$g(t) = \frac{C(t)}{1 + \exp\left(-(k + a(t)^T)\delta)(t - (m + a(t)^T)\gamma\right)}$$

По сути $C(t)$ – это функция верхней границы тренда.

Линейный тренд с точками изменения При прогнозировании задач, в которых нет роста с насыщением, кусочно-постоянная скорость роста дает экономную и часто полезную модель. В

таком случае модель тренда выглядит следующим образом

$$g(t) = (i + a(t)^T \delta)t + (m + a(t)^T \gamma),$$

где k – скорость роста, δ содержит корректировки скорости, m – параметр смещения, а γ_i устанавливается равной $-s_j \delta_j$, чтобы функция была непрерывной.

Автоматический отбор можно выполнить вполне естественным путем, выбрав априорное распределение Лапласа для δ. Что немаловажно, применение распределения Лапласа для корректировок δ не влияет на первоначальную скорость роста τ, поэтому когда τ стремится к 0, обучение сводится к стандартному (а не кусочному) логистическому или линейному росту.

Когда происходит экстраполяция модели за пределы исторических данных для получения прогноза, тренд получает постоянную скорость. Каждая из этих точек имеет изменение скорости $\delta_j \approx Lapalce(0, \tau)$. Параметр τ – коэффициент масштаба для автоматического выбора точек смены тренда. Мы симулируем значения будущих изменений скорости, которые подражают прошлым значениям путем замены τ на дисперсию, оцениваемую по имеющимся данным.

Предположение, что тренд продолжит меняться с той же частотой и скоростью изменений, что и в исторических данных, является довольно сильным, поэтому мы не ожидаем высокой точности от доваерительных интервалов. Однако они являются полезным показателем уровня неопределенности и в особенности *показателем переобучения.*

Временные ряды в бизнес-задачах часто имеют *многопериодную сезонность* как результат человеческого поведения, которые они отражают. Например, 5-дневная рабочая неделя может оказывать влияние на временной ряд, повторяющееся каждую неделю, в то время как графики отпусков и школьных каникул могут вызывать эффекты, повторяющиеся каждый год.

Мы предложили использовать ряды Фурье, чтобы получить гибкую модель периодических изменений (Harvey & Shephard, 1993). Пусть P – постоянное значение периода для рассматриваемого временного ряда (например, $P = 365.2$ для годовых данных или $P = 7$ для недельных данных). Мы можем аппроксимировать произвольные сезонные эффекты с помощью стандартного ряда Фурье

$$s(t) = \sum_{n}^{N} \left(a_n \cos \frac{2\pi n t}{P} + b_n \sin \frac{2\pi n t}{P} \right)$$

Подгонка сезонности требует оценки $2N$ параметров $[a_1, b_1, \ldots, a_N, b_N]^T$. Это делается путем построения матрицы векторов сезонности для каждого значения t в наших исторических и прогнозных данных, например ниже приведен пример для годовой сезонности и $N = 10$

$$X(t) = \left[\cos\left(\frac{2\pi \cdot 1 \cdot t}{365.25}\right), \ldots, \sin\left(\frac{2\pi \cdot 10 \cdot t}{365.25}\right) \right]$$

Тогда сезонная компонента будет иметь вид

$$s(t) = X(t)\beta.$$

В нашей генеративной модели мы берем $\beta \sim Normal(0, \sigma^2)$, чтобы сгладить сезонность. Для годовой и недельной сезонности были найдены значения $N = 10$ и $N = 3$ соответственно, ко-

торые работают достаточно хорошо для большинства задач. Выбори этих параметров можно автоматизировать с помощью критерия отбора модели типа AIC [2, стр. 28].

Для оценивания параметров обучаемой модели используются принципы байесовской статистики: либо поиск апостериорного максимума (MAP) с помощью L-BFGS, либо полный байесовский вывод.

Выбрать наблюдения из данного диапазона

```
df.set_index("date").between_time("8:00", "18:00")  # работает только временными индексами
# или так
df[(df["date"].dt.hour >= 8) & (df["date"].dt.hour < 18)].set_index("date")
```

По умолчанию Prophet для годовой сезонности использует ряд Фурье порядка 10, для недельной сезонности – ряд Фурье порядка 3 и для дневной сезонности – ряд Фурье порядка 4. Обычно эти значения по умолчанию работают очень хорошо, и настройка не нужна.

2.2. Добавление условных сезонностей

Условная сезонность должна иметь цикл, который короче периода, в рамках которого она действует. Так, например, не имеет смысла задавть годовую сезонность, которая будет действовать всего несколько месяцев.

Для прогнозирования использования электроэнергии в студенческом городе вам потребуется задать либо дневную, либо недельную сезонность, а возможно, даже и то, и другое, в зависимости от паттерна поведения студентов, одну дневную/недельную сезонность для летних месяцев, когда студенты вернулись в свои родные города, и другую – дневную/недельную сезонность для остального времени года. В идеале *условная сезонность* должна иметь *как минимум два полных цикла* всякий раз, когда она действует [2, стр. 77].

Процедура добавления этой условной сезонности заключается в добавлении новых булевых столбцов в обучающий датафрейм (а затем соответствующих столбцов в датафрейм, удлиненный на горизонт прогнозирования), указывающих, является ли данное наблюдение выходным или будним днем.

Например можно создать два булевых столбца в наборе данных, которые будут маркировать будни и выходные. То есть у модели получается как бы дополнительный категориальный признак с двумя уникальными значениями для описания дневной сезонности.

```
def is_weekend(ds):
    date = pd.to_datetime(ds)
    return (date.dayofweek == 5 or date.dayofweek == 6)

df["weekend"] = df["ds"].apply(is_weekend)
df["weekday"] = ~df["weekend"]

model = Prophet(
    seaonlity_mode="multiplicative",  # мультипликативная сезонность
    yearly_seasonality=6,  # порядок Фурье для годовой сезонности
    weekly_seasonality=6,  # порядок Фурье для недельной сезонности
    daily_seasonality=False  # отключаем дневную сезонность по умолчанию
)
```

Для создания условных сезонностей мы используем метод `.add_seasonality()` с параметром `condition_name`

```
model.add_seasonality(
```

```
    name="daily_weekend",
    period=1,
    fourier_order=3,
    condition_name="weekend",
)

model.add_seasonality(
    name="daily_weekday",
    period=1,
    fourier_order=3,
    condition_name="weekday",
)

model.fit(df)
future = model.make_future_dataframe(periods=365 * 24)
# индикаторные переменные нужно задать и для прогнгоза
future["weekend"] = future["ds"].apply(is_weekend)
future["weekday"] = future["weekend"]

forecast = model.predict(future)
```

Первый способ применить регуляризацию сезонности – применить регуляризацию глобально, одинаково влияя на все сезонности в модели. Параметр `seasonality_prior_scale` по умолчанию принимает значение 10. Уменьшение этого числа соответствует большей регуляризации, т.е. меньшей гибкости кривой сезонности модели.

```
model = Prophet(
    seasonality_mode="multiplicative",
    yearly_seasonality=4,
    seasonality_prior_scale=10,  # глобальная регуляризация
)
model.fit(df)
future = model.make_future_dataframe(periods=365)
forecast = model.predict(future)
```

Допустим, вас устраивает кривая годовой сезонности с настройками регуляризации по умолчанию, но кривая недельной сезонности имеет много экстремальных значений и характеризуется переобучением. В этом случае можно использовать метод `.add_seasonality()` для создания новой недельной сезонности с другими значениями параметра `seasonality_prior_scale`

```
model = Prophet(
    seasonality_mode="multiplicative",
    yearly_seasonality=4,
    weekly_seasonality=False,  # отключаем недельную сезонность по умолчанию
)

model.add_seasonality(
    name="weekly",
    period=7,
    fourier_order=4,
    prior_scale=0.01,
)

model.fit(df)
future = model.make_future_dataframe(periods=365)
forecast = model.predict(future)
```

Все сезонности могут иметь разную силу регуляризации, для этого используйте вызовы метода `.add_seasonality()`. Разумные значения для параметра `prior_scale` варьируют от 10 до 0.01.

Регуляризация просто уменьшает масштаб влияния праздника. В Prophet есть параметр `holiday_prior_s` Праздники, как сезонность, можно регуляризовать глобально и локально.

Глобальная регуляризация влияния праздников настраивается с помощью парааметра `holiday_prior_sca` Разумные значения `holiday_prior_scale` варьируют от 10 до 0.01. Как и в случае с параметром `seasonality_prior_scale` в большинстве случаев значени параметров `holiday_prior_scale` от 10 до 0.01 будут работать хорошо.

Использование параметра `holiday_prior_scale` позволяет скорректировать влияние всех праздников глобально: каждый праздник регуляризуетсяя одинаково. Однако Prophet позволяет поразному регуляризовать отдельно взятые праздники

```
holidays = make_holidays_df(year_list=year_list, country="US")
black_friday = pd.DataFrame({
    "holiday": "Black Friday",
    "ds": pd.to_datetime([
        "2014-11-28",
        "2015-11-27",
        ...
    ]),
    "prior_scale": 1,
})

taste_of_chicago = pd.DataFrame({
    "holiday": "Taste of Chicago",
    "ds": pd.to_datetime([
        "2014-07-09",
        ...
    ]),
    ""lower_window": 0,
    "upper_window": 4,
    "prior_scale" 0.1,
})

holidays = pd.concat([
    holidays,
    taste_of_chicago
]).sort_values("ds").reset_index(drop=True)
```

Модель Prophet – это *обобщенная аддитивная модель*. Тренд – это самый фундаментальный строительный блок нашего прогноза. Мы добавляем к нему сезонности, праздники и дополнительные регрессоры.

При логистическом росте Prophet всегда требует, чтобы был установлен «потолок» – значение, которое ваш прогноз никогда не превзойдет. В Prophet этот параметр называется `cap`. Чтобы добавить его в Prophet, нам нужно создать новый столбец под названием cap в датафрейме исторических данных, а также продублировать его в удлиненном датафрейме.

По умолчанию Prophet разместит 25 потенциальных точек изменения в первых 80% временного ряда. Чтобы управлять процедурой автоматического обнаружения точек изменения в Prophet, вы можете изменить оба этих параметра с помощью параметров `n_changepoints` (количество точек изменения) и `changepoint_range` (диапазон охвата данных точками изменения,

то есть первые changepoint_range процентов данных, в которых необходимо разместить точки изменения) при создании экземпляра модели.

```
model = Prophet(
    seasonality_mode="multiplicative",
    yearly_seasonality=4,
    n_changepoints=5,
)
```

Это приводит к выводу пяти равномерно распределенных потенциальных точек изменения в первых 80% данных.

А можно расположить все 25 потенциальных точек не в первых 80% обучающих данных, а в первых 50%

```
model = Prophet(
    seasonality_mode="multiplicative",
    yearly_seasonality=4,
    changepoint_range=0.5,  # в первых 50% данных
)
```

Важно помнить, что вы размещаете лишь *потенциальные точки изменения*. Prophet по-прежнему будет пытаться занулить как можно больше потенциальных точек изменений. Prophet никогда не разместить точку изменения в будущих данных. Вот почему по умолчанию Prophet будет использовать только первые 80% данных, чтобы предотвратить выбор плохой точки изменения с помощью нескольких последующих точек.

Модель, у которой будет много точек изменения с большими значениями, будет иметь более широкий доверительный интервал.

Как правило, если вы задаете точки изменения в очень поздние моменты временного ряда, ваша модель будет иметь более высокую вероятность переобучения

```
model = Prophet(
    seasonality_mode="multiplicative",
    yearly_seasonality=4,
    changepoints=["2017-11-01"],
    changepoint_prior_scale=50,  # регуляризация для точек изменения тренда
)
```

Не нужно достаточно часто настраивать количество точек изменения или диапазон охвата данных точками изменения. Значения по умолчанию почти всегда работают очень хорошо. Если вы обнаружите, что Prophet увеличивает или уменьшает количество точек изменения, лучше контролировать это с помощью регуляриации. Для регуляризации мы используем параметр changepoint_prior_scale. Слишком гибкая (сложная) модель имеет высокую вероятность переобучения данных, то есть моделирует шум помимо истинного сигнала. Недостаточно гибкая модель плохо соответсвует данным или не улавливает весь доступный сигнал.

Разумные значения для changepoint_prior_scale обычно находятся в диапозоне от 0.001 до 0.5.

Собственные точки изменения

```
wc_2024 = pd.DataFrame({
    "holiday": "World Cup 2014",
    "ds": pd.to_datetime(["2014-06-12"]),
    "lower_window": 0,
    "upper_window": 31,
})
```

```
wc_2018 = pd.DataFrame({
    "holiday": "World Cup 2018",
    "ds": pd.to_datetime(["2018-06-14"]),
    "lower_window": 0,
    "upper_window": 31,
})

signing = pd.DataFrame({
    "holiday": "Bayern Munich",
    "ds": pd.to_datetime(["2017-07-11"]),
    "lower_window": 0,
    "upper_window": 14,
})

special_events = pd.concat([wc_2014, wc_2018, signing])

changepoints = [
    "2014-06-12",
    "2014-07-13",
    "2017-07-11",
    "2017-07-31",
    "2018-06-14",
    "2018-07-15",
]
```

Для каждого особого события мы добавляем одну потенциальную точку изменения в начале события и одну – в конце.

Иногда даже в тех случаях, когда в данных нет никакой сезонности, можно использовать ту или иную модель сезонности. Потому как она может влиять, к примеру, на праздники. Еще важно уделять внимание регуляризации точек изменения тренда, так как значение по умолчанию часто излишне регуляризует данные [2, стр. 138]

```
model = Prophet(
    seasonality_mode="multiplicative",
    holidays=special_events,
    yearly_seasonality=False,
    weekly_seasonality=False,
    changepoint_prior_scale=1,
    changepoints=changepoints,
)
```

2.3. Добавление регрессоров

Prophet предлагает обобщенный метод добавления любого дополнительного *регрессора*, как бинарного, так и непрерывного.

Следует учитывать, что при использовании дополнительных регрессоров, бинарных и непрерывных, нужно знать *будущие* значения *для всего периода прогноза*. Это не является проблемой для праздников, потому что мы точно знаем, когда наступит каждый праздник в будущем. Все будущие значения должны быть либо известны, как в случае с праздниками, либо спрогнозированы отдельно. Однако вы должны быть осторожно при построении прогноза с использованием данных, которые сами были предсказаны: ошибка в первом прогнозе увеличит ошибку во втором прогнозе, и ошибки будут постоянно накапливаться.

Чтобы добавить в модель новый регрессор, следует воспользоваться методом `.add_regressor()`. В метод `.add_regressor()` мы передаем имя регрессора, которое является именем соответсвующего столбца в кадре данных. Кроме того, можно воспользоваться параметром `prior_scale`, чтобы применить регуляризацию к регрессору, как мы это делали с праздниками, сезонностью и точками изменения тренда.

Еще с помощью параметра `mode` можно указать, должен ли регрессор быть аддитивным или мультипликативным. Чтобы избежать мультиколлинеарности в категориальном регрессоре, закодированном с помощью техники одного активного состояния, лучше удалить один из столбцов. Мультиколлинеарность может затруднить интерпретацию отдельного эффекта. Впрочем Prophet довольно устойчив к мультиколлинеарности, поэтому она не должна существенно повлиять на результаты.

```
model = Prophet(
    seasonality_mode="multiplicative",
    yearly_seasonality=4,
)
# добавляем регрессор clear
model.add_regressor(
    name="clear",
    prior_scale=10,
    standardize="auto",
    mode="multiplicative",
)
# добавляем регрессор not clear
model.add_regressor("not clear")

# добавляем регрессор rain or snow
model.add_regressor("rain or snow")
```

Помним, что нам нужны будущие данные для наших дополнительных регрессоров и мы собираемся прогнозировать только две недели.

```
from datetime import timedelta
# удаляем последние две недели обучающих данных
train = df[df["ds"] < df["ds"].max() - timedelta(weeks=2)]

mdoel.fit(train)
future = model.make_future_dataframe(periods=14)
future["clear"] = df["clear"]
future["not clear"] = df["not clear"]
future[rain or snow"] = df["rain or snow"]

forecast = model.predict(future)
```

С помощью фукнции `regressor_coefficients()` можно вывести регрессионные коэффициенты

```
from prophet.utilities import regressor_coefficients

regressor_coefficients(model)
```

В столбце coef сохраняются ожидаемые значения коэффициента. То есть ожидаемое влияние регрессора на `y` при увеличении регрессора на единицу. Например, коэффициент для регрессора temp равен 0.012282. Этот коэффициент говорит нам о том, что с каждым градусом выше среднего происходит увеличение количества поездок на 0.012282, то есть на 1,2% [2, стр. 149].

Для бинарного регрессора rain or snow коэффициент показывает, что в дожливые или снежные дни пассажиропоток снизится на 20,7% по сравнению с пасмурными днями, поскольку категория `claudy`, став исключенным столбцом, является у нас опорной категорией, с которой мы сравниваем остальные категории, ставшие теперь столбцами.

Если бы мы включили все 4 состояния погоды, мы бы сказали, что пассажиропоток снизился на 20,7% по сравнению со значением, прогнозируемым на тот же день модель без включения этих 4 состояний погоды.

Дополнительные регрессоры в Prophet всегда моделируются как *линейные зависимости*. Это означает, что, например, наш дополнительный регрессор temp, который увеличивает количество поездок на 1,2% с каждым градусом, моделирует тренд, который будет продолжаться до бесконечности. Если бы температура поднялась до 120 градусов по Фаренгейту, у нас не было бы возможности изменить линейную зависимость и сообщить Prophet, что количество поездок, вероятно, теперь уменьшится, поскольку на улице стало очень жарко. Хотя это является ограничением Prophet, на практике это не всегда является проблемой. Линейная зависимость очень часто предстает хорошим показателем реальной зависимости, особенно для небольшого диапазона данных [2, стр. 151].

Если во временном ряду есть групповая аномалия (несколько точек, идущих подряд), которая сильно искажает прогнозы, то ее можно просто вырезать [2, стр. 156]. Prophet очень хорошо обрабатывает пропущенные данные, поэтому небольшой гэп не вызовет никаких проблем.

```
# удаляем данные с выбросами
df2 = df[(df["ds"] < "2016-07-29") | (df["ds"] > "2016-09-01")]

model = Prophet(
    seasonality_mode="multiplicative",
    yearly_seasonality=6,
)
model.fit(df2)
future = model.make_future_dataframe(periods=365 * 2)
forecast = model.predict(future)
```

Аномалии можно не удалять из временного ряда, а передать им `None`. Тогда Prophet будет делать прогнозы для этих дат [2, стр. 158].

```
df3.loc[df3["ds"].dt.year == 2016, "y"] = None
```

2.4. Автоматическое обнаружение выбросов

Винзоризация – грубый инструмент, который, как правило, не работает с неплоскими трендами. Винзоризация требует, чтобы аналитик указал конкретный нижний и/или верхний процентиль, всем значениям выше или ниже этого процентиля приндительно присваиваеются значения, соответствующие нижнему и/или верхнему процентилю.

```
stats.mstats.winsorize(df["y"], limits=(0, 0.05), axis=0)
```

Тримминг – похожая техника, однако здесь экстремальные значения удаляются.

Вместо использования процентилей иногда имеет смысл использовать стандартное отклонение. Выбросы просто удаляются. Точки, которые лежат на 1,65 стандартного отклонения выше среднего исключаются.

```
df[stats.zscore(df["y"]) < 1.65]
```

Этот метод тоже не подходит, когда данные имеют тренд. Очевидно, что точки, расположенные позже во временном ряду с восходящим трендом, будут обрезаны с большей вероятностью, чем те, которые лежат ранее [2, стр. 163].

Для вычисления среднего и стандартного отклонения можно воспользоваться скользящем средним.

```
df["moving_average"] = df.rolling(
    window=300,
    min_periods=1,
    center=True,
    on="ds"
)["y"].mean()

df["std_dev"] = df.rolling(
    window=300,
    min_periods=1,
    center=True,
    on="ds"
)["y"].std()

df["lower"] = df["moving_average"] - 1.65 * df["std_dev"]
df["upper"] = df["moving_average"] + 1.65 * df["std_dev"]

df = df[(df["y"] < df["upper"]) & (df["y"] > df["lower"])]
```

Важное преммущество этого метода состоит в том, что он учитывает тренд.

Еще можно объявить выбросом все, что выходит за границы доверительных интервалов. Однако использование этого метода основано на неявном предположении, что данные являются стационарными и имеют постоянную дисперсию [2, стр. 165].

2.5. Доверительные интервалы

Есть три источника неопределенности, которые составляют общую неопределенность модели Prophet:

- неопределенность в тренде,
- неопределенность в сезонности, праздниках и дополнительных регрессорах,
- неопределенность из-за шума в данных.

2.5.1. Моделирование неопределенности тренда

По умолчанию Prophet оценивает только неопределенность тренда плюс неопределенность, возникающую из-за случайного шума в данных. Шум моделируется как нормальное распределение вокруг тренда, а неопределенность тренда моделируется с помощью апостериорного максимума (MAP).

Чтобы получить оценки неопределенности для сезонности используется семплирование на основе MCMC. Самый большой источник неопределенности в прогнозах Prophet – это возможность изменений тренда в будущем.

```
model = Prophet(
    uncerainty_samples=1000
)
```

Разработчики Prophet отмечают, что их предположение о том, что изменения тренда в будущем никогда не будут более значительными, чем изменения тренда в будущем никогда не будут более значительными, чем изменения тренда в исторических данных, является очень ограничительным.

2.5.2. Моделирование неопределенности сезонности

MAP-оценка выполняется очень быстро, поскольку это режим Prophet по умолчанию, но он не работает с сезонными колебаниями, поэтому необходим другой метод. Чтобы смоделировать неопределенность сезонности, Prophet использует метод MCMC.

Семплирование на основе MCMC выполняется очень медленно. К сожалению, на компьютере с Windows API PyStan, который взаимодействует с моделью Prophet на языке Stan, есть проблемы, что означает, что семплирование на основе MCMC выполняется очень меделенно. В зависимости от количества точек данных обучение модели на компьютере с Windows иногда может занять несколько часов.

Команда Prophet рекомендует пользователям с Windows-компьюетарами работать с Prophet в R или использовать Python на виртуальной машине Linux.

Моделируем сезонную неопределенность

```
# При mcmc_samples=0 Prophet вернется к MAP-оценке и вычислит неопределенность только для компон
    енты тренда
model = Prophet(mcmc_samples=300)
```

Если вы используете семплинг на основе MCMC, обязательно обратите внимание на увеличивашееся количество точек изменения. Если ваша линия тренда кажется слишком извилистой, вы можете просто уменьшить `changepoint_prior_scale`, чтобы применить регуляризацию [2, стр. 191].

2.6. Перекрестная проверка

Дла каждой пороговой точки (cutoff) модель будет обучаться на всех данных до нее, а потом будет сделан прогноз в соответствии с горизонтом. Затем полученный прогноз будет сравнен с фактическими значениями зависимой переменной и будет посчитана метрика качества. Далее модель повторно обучится на всех данных до второй пороговой точки, и процесс будет повторен. Окончательная оценка качества модели будет представлять собой значение качества модели, усредненное по всем пороговым точкам.

Для Prophet рекомендуется использовать как минимум два полных цикла сезонности, поскольку требуется моделировать годовую сезонность.

Пусть есть данные за 3 года, и таким образом, есть как миинимум 2 полных цикла сезонности. Тогда установим размер исходной обучающей выборки равным 2 годам, Руководство магазина хочет прогнозировать на месяц вперед и поэтому устанавливает горизонт, равным 30 дням. Оно планирует запускать модель каждый квартал, поэтому установим период равным 90 дням.

```
df_cv = cross_validation(
    model,
    horizon="90 days",
    period="30 days",
    initial="730 days",
)
```

Мы начинаем обучение с исходного периода (initial) в 2 года, что составляет 730 дней. Устанавливаем горизонт (horizon) равным 90 дней. И наконец, устанавливаем период (period) равным 30 дней, посольку повторно обучаем и оцениваем нашу модель каждые 30 дней.

С вычислительной точки зрения перекрестная проверка – очень затратная операция, и ее можно распараллелить, чтобы ускорить процесс. Все что нужно сделать, – это использовать ключевое слово для `parallel`. Можно выбрать `None`, `"processes"`, `"threads"`, `"dask"`.

```
df_cv = cross_validation(
    model,
    horizon="90 days",
    period="30 days",
    initial="730 days",
    parallel="processes",
)
```

Настройка `parallel="processes"` использует класс `concurrent.futures.ProcessPoolExecutor`, тогда как `parallel="threads"` использует класс `concurrent.futures.ThreadPoolExecutor`. Если есть сомнения, то лучше выбрать `"processes"`.

Для вычисления пяти метрик качества для прогнозов используется функция `performance_metrics()`

```
df_p = performance_metrics(df_cv)
```

Среднеквадратическую ошибку (MSE) нелегко интерпретировать – единицей измерения MSE является квадрат единицы измерения зависимой переменной. Она также чувствительна к выбросам. Однако MSE остается популярной, потому что можно доказать, что MSE равна квадрату смещения плюс дисперсия, поэтому минимизация этого показателя может уменьшить как смещение, так и дисперсию [2, стр. 200].

Средняя абсолютная ошибка (MAE), в отличие от MSE и RMSE, одинаково взвешивает каждую ошибку, она не придает большого значения выборосам или точкам с необычно высокими ошибками.

Охват (coverage) – это посто процентная доля фактичесских значений, которые лежат между прогнозируемыми верхней и нижней границами доверительного интервала. По умолчани границы накрывают 80% данных, поэтому значение охвата должно быть равно 0.8. Если вы обнаружите значение охвата, которое не равно значению `interval_width`, это означает, что ваша модель плохо откалибрована с точки зрения неопределенности. На практие это означает, что вы вероятно, не можете серьезно относится к полученным доверительным интервалам прогнозов.

Выбор метрики качества для оптимизации вашей модели не является тривиальным выбором. Он может оказать существенное влияние на вашу итоговую модель в зависимости от характеристик данных. С математической точки зрения оптимизация модели по MSE создать модель, предсказывающую значения, близкие к среднему значению ваших данных, а оптимизация по MAE даст прогнозы, близкие к медианному значению данных. Оптимизация по MAPE часто дает заниженные прогнозы.

Однако, если временной ряд является прерывистым, то есть если большинство дат имеют значение 0, то нужно ориентироваться не на медиану, а на среднее. Медиана будет равно 0! В этом случае нужна метрика MSE именно потому, что она чувствительна к выбросам.

Вернемся к кадру данных с результатами перекрестной проверки. Первая строка кадра данных – `"9 days"`. Каждое значение метрики представляет собой скользящее среднее, вычисленное до указанного дня. Функция `performance_metrics()` принимает аргумент `rolling_window`, в котором вы можете изменить ширину окна, но по умолчанию оно равно 0.1. Это представляет собой

долю горизонта, которую нужно включить в окно. Поскольку 10% нашего 90-дневного горизонта составляют 9 дней, это значение и будет первой строкой кадра данных.

Параметру `cutoffs` функции `cross_validation()` можно передать пользовательский список дат для пороговых точек.

```
cutoffs = [
    pd.Timestamp(f"{year}-{month}-{day}")
    for year in range(2005, 2019)
    for month in range(1, 13)
    for day in [1, 11, 21]
]

df_cv = cross_validation(
    model,
    horizon="90 days",
    parallel="processes",
    cutoffs=cutoffs,
)

df_p = performance_metrics(df_cv)
```

2.7. Обычная оптимизация гиперпараметров по сетке на основе перекрестной проверки

```
param_grid = {
    "changepoint_prior_scale": [0.01, 0.001],
    "seasonality_prior_scale": [1.0, 0.1],
}

# комбинации гиперпараметров
all_params = [
    dict(zip(param_grid.keys(), value))
    for value in itertools.product(*param_grid.values())
]

rmse_values = []

# создаем пользовательский список пороговых точек
cutoffs = [
    pd.Timestamp(f"{year}-{month}-{day}")
    for year in range(2010, 2019)
    for month in range(1, 13)
    for day in [1, 11, 21]
]

# выполняем перекрестную проверку
for params in all_params:
    model = Prophet(yearly_seasonality=4, **all_params).fit(df)
    df_cv = cross_validation(
        model,
        cutoffs=cutoffs,
        horizon="30 days",
        parllel="processes"
    )
    df_p = performance_metrics(df_cv, rolling_window=1)
    rmse_values.append(df_p["rmse"].values[0])

best_params = all_params[np.argmin(rmse_values)]
```

2.7.1. Рекомендации по подбору гиперпараметров

Важные гиперпараметры:

- `changepoint_prior_scale`,
- `seasonality_prior_scale`.

Самым важным гиперпараметром является `changepoint_prior_scale`. Диапазон значений от 0.5 до 0.001 будет приемлемым для большинства задач. Вторым по важности гиперпараметром является `seasonality_prior_scale`. Диапазон – от 10 (по сути, регуляризация отсутствует) до 0.01. Для выбора значения гиперпараметра `seasonality_mode` лучше всего просто изучить график временного ряда и посмотреть, растет ли величина сезонных колебаний вместе с трендом или остается постоянной. Для `changepoint_range` обычно подходит значение по умолчанию 80% (точки изменения тренда определяем в первых 80% обучающих данных). Оно обеспечивает хороший баланс, позволяя тренду измениться там, где это необходимо, но не позволяя ему переориентироваться на последних 20% данных, где ошибки не могут быть исправлены. Параметр `growth` лучше не включать в поиск, значения `"linear"`, `"logistic"`, `"flat"` подбирают, исходя из визуального анализа данных [2, стр. 211].

2.8. Байесовская оптимизация гиперпараметров на основе перекрестной проверки

Теперь попробуем баейсовскую оптимизацию гиперпараметров в Prophet

```
param_types = {
    "changepoint_prior_scale": "float",
    "seasonality_prior_scale": "float",
}

bounds = {
    "changepoint_prior_scale": [0.001, 0.5],
    "seasonality_prior_scale": [0.01, 10]
}
```

Затем пишем функию байесовской оптимизации и осуществляем поиск оптимальных значений гиперпараметров

```
def objective(trial):
    params = {}
    for param in [
        "changepoint_prior_scale",
        "seasonality_prior_scale",
    ]:
      params[param] = trial.suggest_uniform(
          param,
          bounds[param][0],
          bounds[param][1],
      )
  m = Prophet(
      yearly_seasonality=4,
      seasonality_mode="additive",
      **params,
  )
  m.fit(df)
```

```
  df_cv = cross_validation(
      m,
      initial="730 days",
      period="30 days",
      horizon="90 days",
      parallel="processes",
  )
  df_p = performance_metrics(df_cv, rolling_window=1)

  return df_p["rmse"].values[0]

sampler = RandomSampler(seed=10)
study = optuna.create_study(
    sampler=sampler,
    direction="minimize"
)
study.optimize(lambda trail: objective(trial), n_trails=10)

study.best_params
study.best_value
```

Список литературы

1. *Маккинли У.* Python и анализ данных, 2015. – 482 с.
2. *Груздев А.* Прогнозирование временных рядов с помощью Facebook, Prophet, ETNA, sktime и LinkedIn Greykite: Строим, настраиваем, улучшаем модели прогнозирования временных рядов с помощью специальных библиотек. – М.: ДМК Пресс, 2023. – 780 с.