Прогнозирование временных рядов с использованием XGboost для чайников

Анисимов М., Белобородова П., Варданян Г., Гнилова О.

31 мая 2017 г.

Содержание

1 Как работает XGBoost?

XGBoost используется для задач машинного обучения с учителем, то есть предполагается наличие обучающей выборки с несколькими признаками x_i, которая нужна, чтобы настроить модель и прогнозировать целевую переменную y_i.

Под моделью понимается некая математическая структура, показывающая, как делается прогноз для y_i при известных x_i. Например, так записывается линейная модель: $\hat{y_i} = \sum_j w_j x_{ij}$

В зависимости от контекста проблемы могут решаться разные задачи, например, регрессии (как в случае с временными рядами) или классификации. Однако в любом случае нужно найти способ подбора параметров $\Theta = \{w_j \mid j = 1, ..., d\}$ по обучающей выборке. Для этого вводится целевая функция, состоящая из двух частей: функции потерь на обучающей выборке и штрафующей компоненты:

$$Obj(\Theta) = L(\theta) + \Omega(\Theta)$$

$L(\theta)$ показывает, насколько хорошо модель делает прогнозы на обучающей выборке. И часто для этой цели используют среднеквадратичную ошибку:

$$L(\theta) = \sum_i (y_i - \hat{y_i})$$

Или логистическую функцию потерь:

$$L(\theta) = \sum_i [y_i ln(1 + e^{-\hat{y}_i}) + (1 - y_i) ln(1 + e^{\hat{y_i}})]$$

$\Omega(\Theta)$ отвечает за "сложность"модели, не давая параметрам принимать слишком большие значения, и тем самым предотвращает переобучение. Причём можно использовать как L1-норму ($\Omega(w) = \lambda\|w\|_1$), так и L2-норму ($\Omega(w) = \lambda\|w\|^2$).

Теперь рассмотрим модель, которую использует XGBoost, – композицию деревьев.

В дереве каждый объект путём классификации попадает в какой-нибудь лист, которому соответсвует число, то есть прогнозное значение для данного объекта. Как правило, единственное дерево даёт прогноз низкого качества, отчего на практике строят несколько деревьев и суммируют их прогнозы. Математически это можно записать следующим образом:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F},$$

где K – число деревьев, f – функция, ставящая в соответсвие признакам число (прогноз), $\mathcal{F}$ – пространство функций, содержащее все возможные деревья. Параметры в этой модели: $\Theta = \{f_1, f_2, ..., f_K\}$. Тогда задача сводится к оптимизации следующей целевой фцункции:

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Трудность задачи заключается в том, что параметры – это функции, определяющие структуру дерева и значения в листах, а не числа, поэтому не получится просто продифференцировать и найти экстремум. Также сложность представляет обучение большого количества деревьев одновременно. Но выход есть! И он называется бустинг (или additive training): начнём с константного прогноза и на каждом шаге будем добавлять новую функцию:

$$\begin{aligned}
&\hat{y}_i^{(0)} = 0\\
&\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)\\
&\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)\\
&...\\
&\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}$$

Но как же определить, какую функцию добавлять на шаге t? Для этого надо оптимизировать целевую функцию:

$$\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) =\\
&= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + const
\end{aligned}$$

Будем использовать сренеквадратичную ошибку в качестве функции потерь, тогда задача примет вид:

$$\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^{t} \Omega(f_i) =\\
&= \sum_{i=1}^{n} [2 \cdot \underbrace{(\hat{y}_i^{(t-1)} - y_i)}_{\text{остаток предыдущего шага}} \cdot f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + const
\end{aligned}$$

В случае со среднеквадратичной ошибкой найти минимум целевой функции несложно, так как это парабола относительно f, в общем же случае нужно разложить целевую функцию в ряд Тейлора до второго члена:

$$Obj^{(t)} \simeq \sum_{i=1}^{n}[l(y_i, \hat{y_i}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2}h_i f_t(x_i)^2] + \Omega(f_t) + const,$$

где в общем случае g_i и h_i – это:

$$g_i = \partial_{\hat{y_i}^{(t-1)}} l(y_i, \hat{y_i}^{(t-1)})$$

$$h_i = \partial^2_{\hat{y_i}^{(t-1)}} l(y_i, \hat{y_i}^{(t-1)})$$

Или для среднеквадратичной ошибки:

$$g_i = \partial_{\hat{y_i}^{(t-1)}} (\hat{y_i}^{(t-1)} - y_i)^2 = 2(\hat{y_i}^{(t-1)} - y_i)$$

$$h_i = \partial^2_{\hat{y_i}^{(t-1)}} (\hat{y_i}^{(t-1)} - y_i)^2 = 2$$

Уберём все константы, которые и так бы исчезли при дифференцировании, и получим окончательную целевую функцию для нового дерева:

$$Obj^{(t)} = \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t(x_i)^2] + \Omega(f_t)$$

Теперь разберёмся с регуляризацией и определимся с $\Omega(f_t)$! Для этого вспомним, как мы определили дерево $f(x)$:

$$f_t(x) = w_{q(x)}, w \in \mathbb{R}^T, q : \mathbb{R}^d \to \{1, 2, ..., T\},$$

где w –вектор прогнозов в листах, q – функция, относящая объект к определённому листу, T – количество листов. Тогда определим сложность модели так:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Теперь введём список объектов, попадающих в лист j как $I_j = \{i \mid q(x_i) = j\}$ и перегруппируем объекты по листу, в который они попадают, учитывая, что у объектов из одного листа будет один и тот же прогноз:

$$\begin{aligned}
Obj^{(t)} &\simeq \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t(x_i)^2] + \Omega(f_t) \\
&= \sum_{i=1}^{n}[g_i w_{q(x)} + \frac{1}{2}h_i w^2_{q(x)}] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \\
&= \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right) w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right) w_j^2\right] + \gamma T
\end{aligned}$$

Это выражение можно записать ещё компактнее, если обозначить $G_j = \sum_{i\in I_j} g_i$ и $H_j = \sum_{i\in I_j} h_i$:

$$Obj^{(t)} = \sum_{j=1}^{T}\left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2\right] + \gamma T$$

Получили квадратичную функцию относительно w_j, поэтому оптимальные значения найти легко:

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$Obj^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

Последнее равенство – это оценка структуры дерева: чем ниже значение Obj^*, тем лучше структура.

Последнее, что осталось рассмотреть – вопрос о том, как строить следующее разбиение в дереве. Деревья строятся жадно. Поэтому начнём с дерева глубины 0 и будем для каждого листа в дереве пытаться построить ещё одно разбиение. Тогда изменение целевой функции будет следующим:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma$$

где:

$\frac{G_L^2}{H_L + \lambda}$ – прогноз левого листа
$\frac{G_R^2}{H_R + \lambda}$ – прогноз правого листа
$\frac{(G_L + G_R)^2}{H_L + H_R + \lambda}$ – текущий прогноз (если не разбивать дальше)
γ – штраф за добавление нового листа, то есть за сложность дерева

Посчитав значение этой функции для всех возможных разбиений, выбираем наибольшее и строим разбиение. Причём, если вышло, что сумма всех прогнозов меньше, чем γ, то ветку добавлять не надо!

2 Как создавать признаки?

Изучив все возможные результаты запросов в гугле и покапавшись в недрах форумов на Kaggle, мы поняли, что признаки придумывать очень трудно, но не для всех данных это одинаково трудно.

Ряды, которые анализировали мы, можно разделить на следующие группы (в порядке убывания трудности придумывания признаков):

1. годовые данные
2. данные по кварталам
3. данные за месяц
4. данные за день

Однако способы создания признаков для них совпадают – разница лишь в том, что для более частых наблюдений можно придумать больше простых признаков. Вообще, все типы признаков, которые мы пытались создать и опробовать в обучении можно также разделить на категории:

1. признаки из информации о времени: можно закодировать месяц, день, время суток и т. д.
2. признаки, полученные манипуляцией данными: лаги различной степени, усреднённые предыдущие значения, среднее по группам

3. замысловатые признаки, которые можно получить, воспользовавшись библиотекой: мы опробовали tsfresh и FATS

4. признаки, которые мы добавляем, потому что откуда-то знаем, что они связаны с данными: информация о праздниках, о регулярных значимых событиях

А теперь рассмотрим примеры!

2.1 Про шампанское

Начнём со случая, когда нам достались данные с выраженной сезонностью, а мы ещё не читали форум на Kaggle. Выглядят они так:

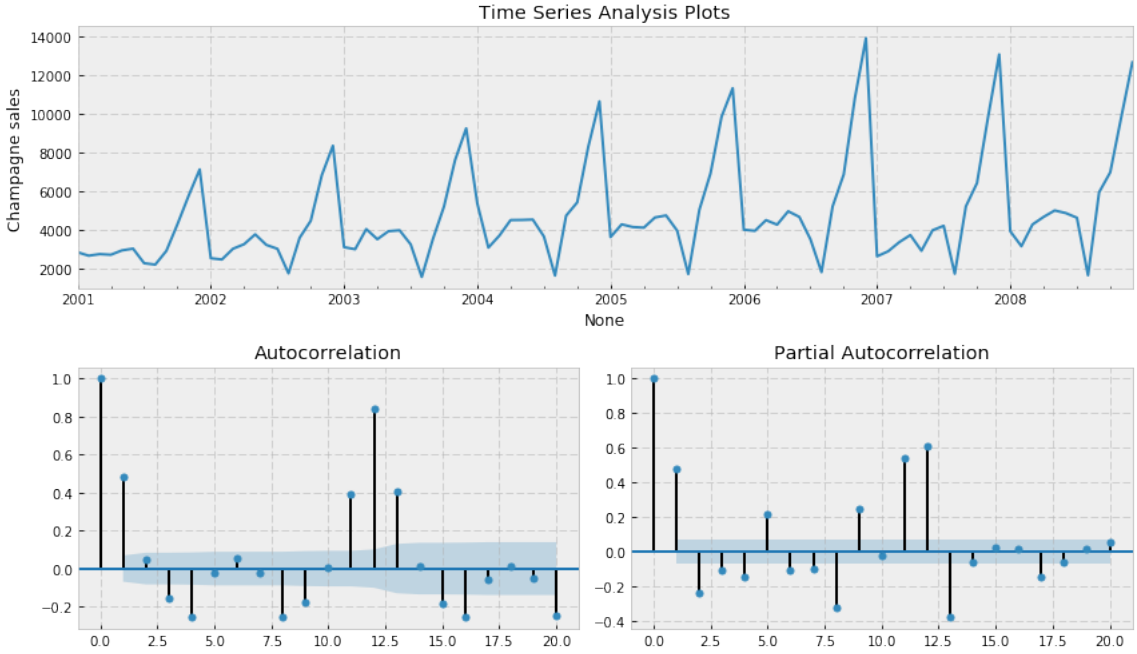


Рис. 1: Продажи шампанского

Видно, что продажи резко растут каждый декабрь, а в августе шампанское пьют лишь самые преданные фанаты напитка. Также заметим, что в данных есть восходящий тренд по декабрьским значениям. Но пока с этим делать ничего не будем, а добавим только признаки, кодирующие все месяцы в году, следующим образом:

```
import pandas as pd

df = pd.read_csv('monthly-champagne-sales-in-1000s.csv') # загружаем данные
df['Year'] = df['Month'].map(lambda x: x[0]) # из столбца Month со значениями
                                             # вида Y-MM извлекаем год
df['Month'] = df['Month'].map(lambda x: x[-2:]) # и месяц
df.index = pd.date_range(start='01-2001', end='01-2009', freq='M') # создаём
                                                  # индексы в формате даты
df_features = df['Month'] # записываем признаки в новый датафрейм
df_features = pd.get_dummies(df_features, drop_first=False) # и кодируем их
df.drop(labels=['Month', 'Year'], axis=1, inplace=True) #  удаляем признаки
                                                    # из исходного датасета
```

Обучим на них XGboost с параметрами, стоящими по умолчанию:

```
gbm = XGBRegressor()
res = gbm.fit(X=df_features.loc[:'2007'], y=df['Champagne sales'][:'2007'])
y_pred = pd.DataFrame(data=gbm.predict(df_features))
y_pred.index = df.index
```

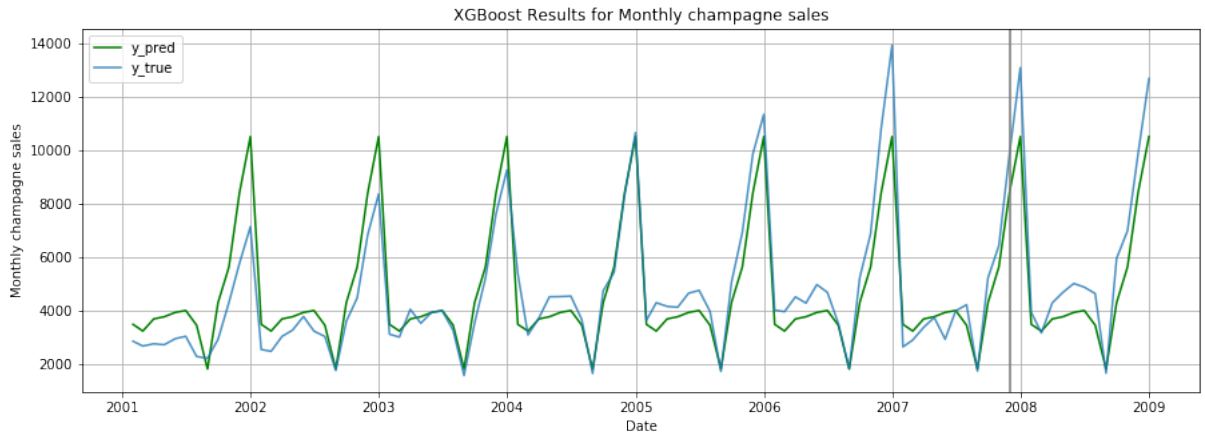


Рис. 2: Результаты с примитивными признаками

Для ориентира и сравнения возьмём прогноз "завтра-будет-как-вчера"и посчитаем для обоих результатов среднеквадратичные ошибки:

Рис. 3: Результаты с примитивными признаками

Соответсвующие среднеквадратичные ошибки:

$$MSE_{primitive} = 5912249, MSE_{XGBoost-train} = 1096935, MSE_{XGBoost-test} = 1201570$$

Несмотря на то, что XGBoost уже показал значительно более точные результаты, весь прогноз - это среднее значение по месяцам. При этом оказалось (что было вполне ожидаемо), что среди самых значимых признаков - закодированные декабрь и август:

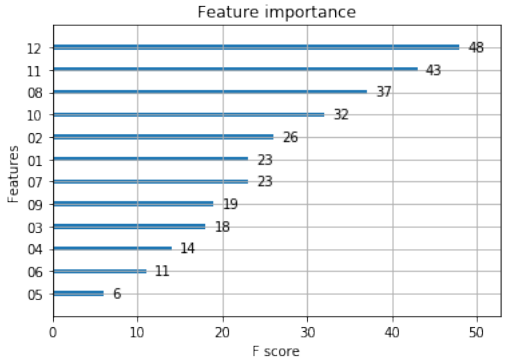


Рис. 4: Важность признаков для примитивных признаков

Но не будем довольствоваться этим результатом и отправимся на Kaggle, где узнаем, что участники часто в качестве признаков используют лаги данных, то есть те же значения, но сдвинутые на некоторое количество периодов назад. Разумеется, излишне было бы брать очень большие лаги, так как они не будут нести новой информации, поэтому выберем признак $t-1$, предполагая, что значения из период в период не будут изменяться слишком сильно, и $t-12$, предполагая, что этот признак учтёт закономерность с декабрями и августами. Пробуем!

```
temps = pd.DataFrame(df.values)
mnth = pd.DataFrame(df_features.values)
df_features_new = pd.concat([temps.shift(12), temps.shift(1), mnth], axis=1)
df_features_new.columns = ['t-12', 't-1', '01', '02', '03', '04', '05', '06', '07', '08',
df_features_new.index = df.index

gbm = XGBRegressor()
res = gbm.fit(df_features_new.loc[:'2007'], df['Champagne sales'][:'2007'])
y_pred = pd.DataFrame(data=gbm.predict(df_features_new))
y_pred.index = df.index
```

Снова обучаем модель и видим, что стало значительно лучше! Среднеквадратичная ошибка для такого прогноза составляет 61371 на обучающей выборке и 373913 на тестовой (против семизначных результатов предыдущих прогнозов).

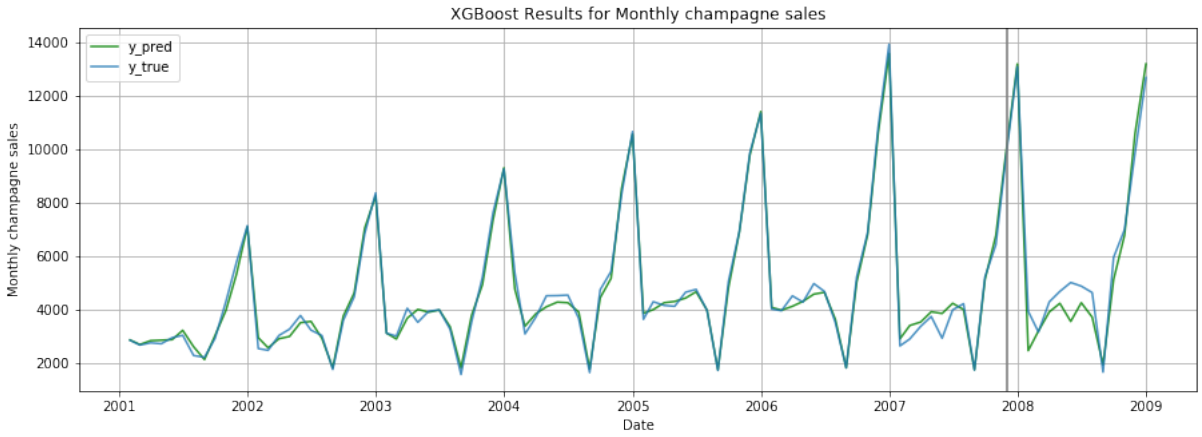


Рис. 5: Итоговый результат для шампанского

2.2 Про скорость обращения денег

Следующий ряд поучителен тем, что данные в нём годовые и никаких очевидных закономерностей в них не наблюдается. Это данные по скорости обращения денег за период 1860-1970 годов, и вот как они выглядят:

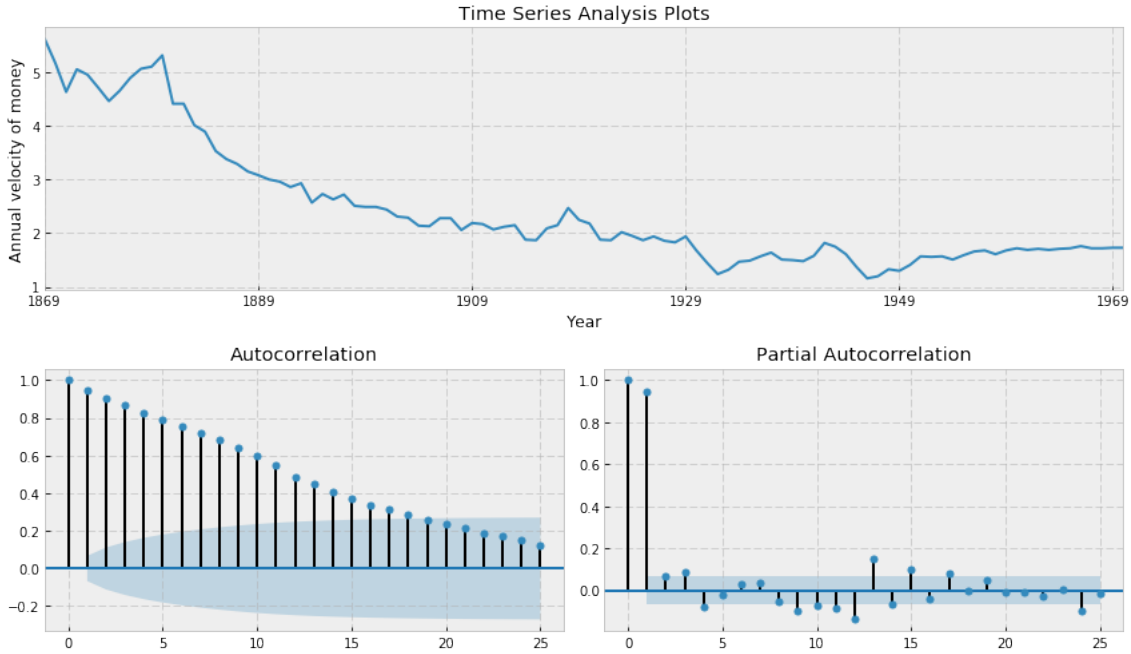


Рис. 6: Скорость обращения денег

Начнём, как и в предыдущем случае с наивного прогноза, предсказывающего завтрашний значения вчерашними:

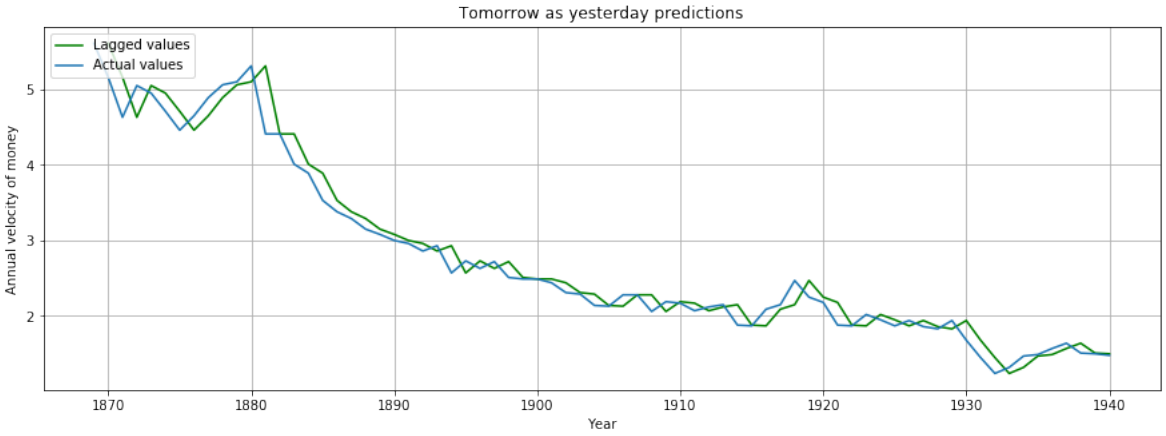


Рис. 7: Скорость обращения денег - наивный прогноз

Среднеквадратичная ошибка для этого прогонза составляет 0.04456.

Что касается признаков, то разумно начать с добавления лагов $t-1$ и $t-2$, потому что у наблюдений не сильно большой разброс, особенно с 1980-х годов. Но поскольку мы уже знаем, что такие признаки окажутся очень значимыми, то имеет смысл перед этим шагом поэкспериментировать, а именно: обучить алгоритм только на среднем двух предыдущих значений, оценить качество, а потом добавить лаги и сравнить значимость признаков.

Добавим средние прошлого и позапрошлого года:

```
import pandas as pd
temps = pd.DataFrame(df.values)     # создаём датафрейм из наблюдений
shifted = temps.shift(1)            # сдвигаем наблюдения на один шаг назад
window = shifted.rolling(window=2) # настраиваем окно так, чтобы в него
                                    # попадали прошлое и позопрошлое значения
means = window.mean()               # усредняем
```

Обучаем алгоритм с дефолтными параметрами и получаем такой результат:

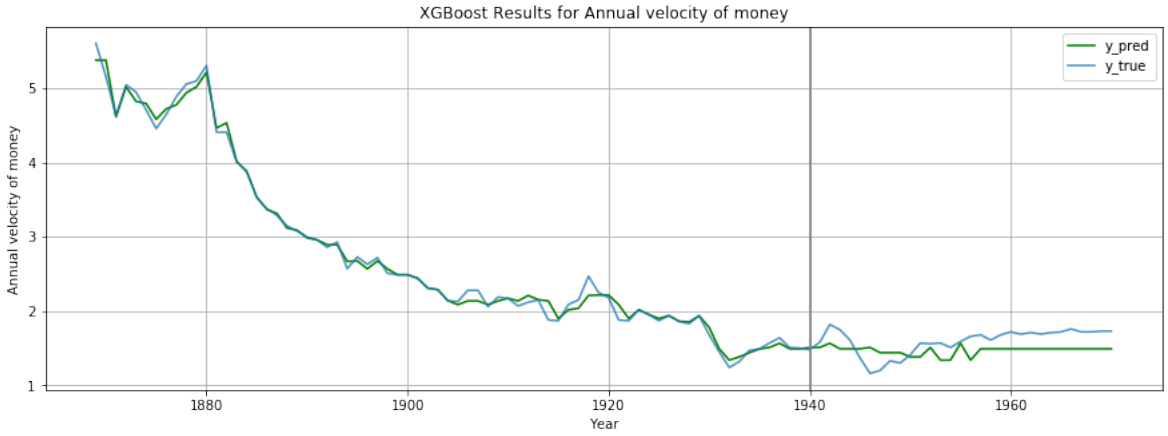


Рис. 8: Скорость обращения денег - прогноз XGBoost по усреднённым двум предыдущим значениям

$$MSE_{train} = 0.00734, MSE_{test} = 0.04035$$

Результат уже лучше, чем у наивного прогноза. Посмотрим, что будет если добавить временные лаги.

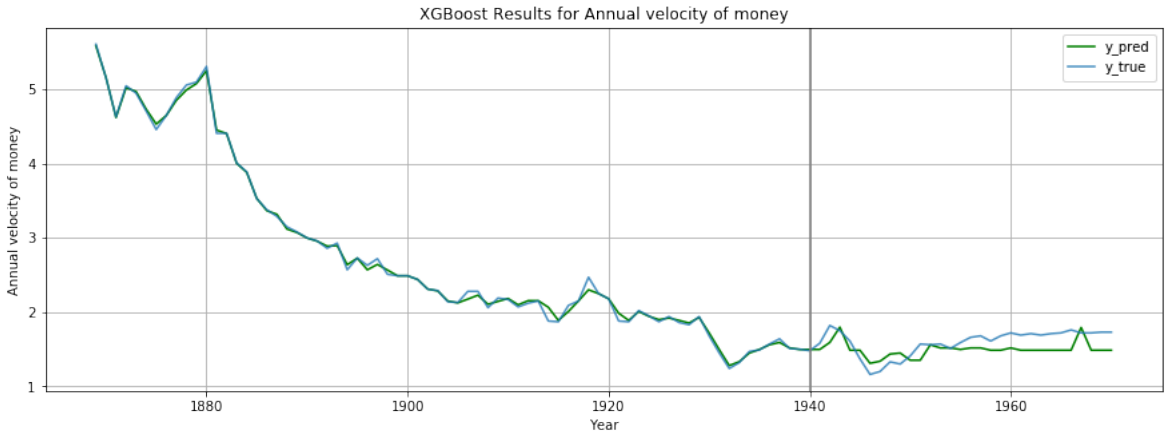


Рис. 9: Скорость обращения денег - прогноз XGBoost с двумя лагами

$$MSE_{train} = 0.00213, MSE_{test} = 0.02772$$

Результат улучшился, а признак со средним оказался не очень значимым:

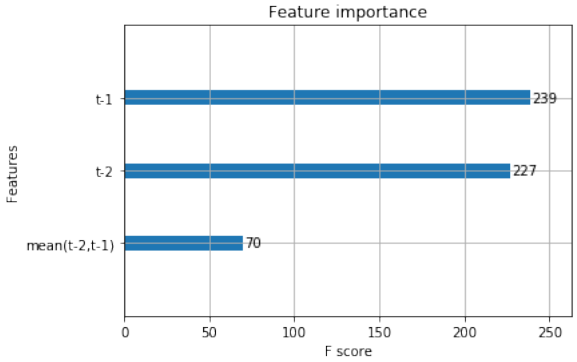


Рис. 10: Скорость обращения денег - значимость признаков

Поэтому отбросим это среднее, а вместо этого добавим большее число лагов и научимся выбирать лучшие признаки. Создаём 12 штук:

```
import pandas as  pd
df_features = pd.DataFrame()
for i in range(12,0,-1):
    df_features['t-'+str(i)] = df['Annual velocity of money'].shift(i)
```

А дальше обучаем, как обычно, и получаем результат, в котором $MSE_{train} = 0.00047$, $MSE_{test} = 0.02806$:

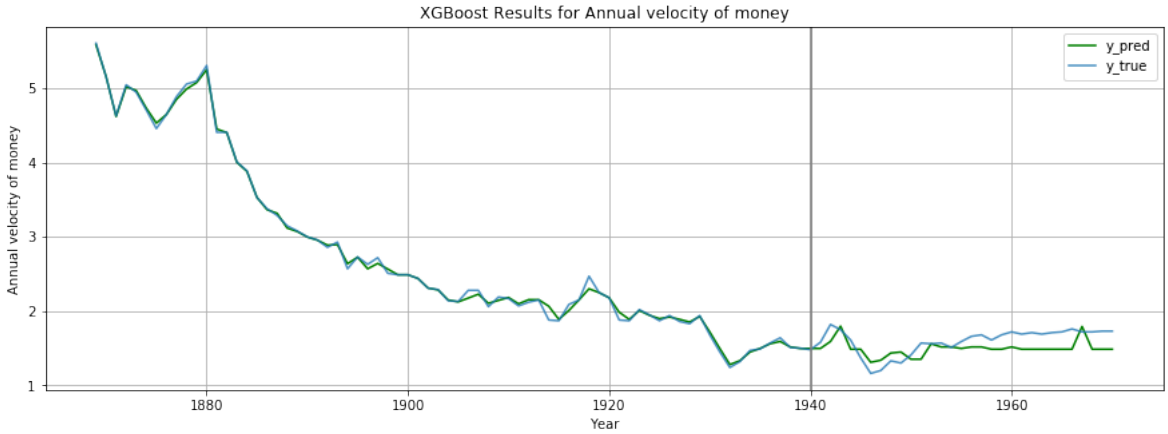


Рис. 11: Скорость обращения денег - много лагов

Смотрим на важность признаков:

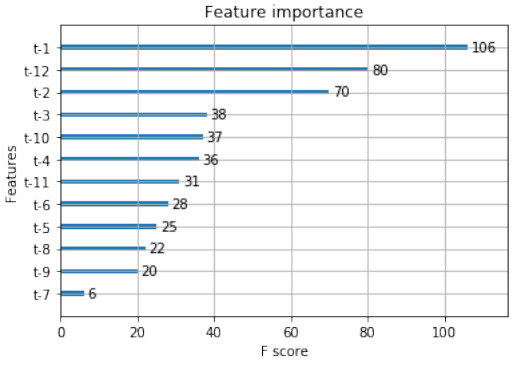


Рис. 12: Скорость обращения денег - много лагов, важность признаков

Теперь оставим только самые важыне - $t-1$ и $t-12$. Обучив модель с такими признакми, получим более низкую среднеквадратичную ошибку на контроле, но более высокую на обучении: $MSE_{train} = 0.0038, MSE_{test} = 0.02$:

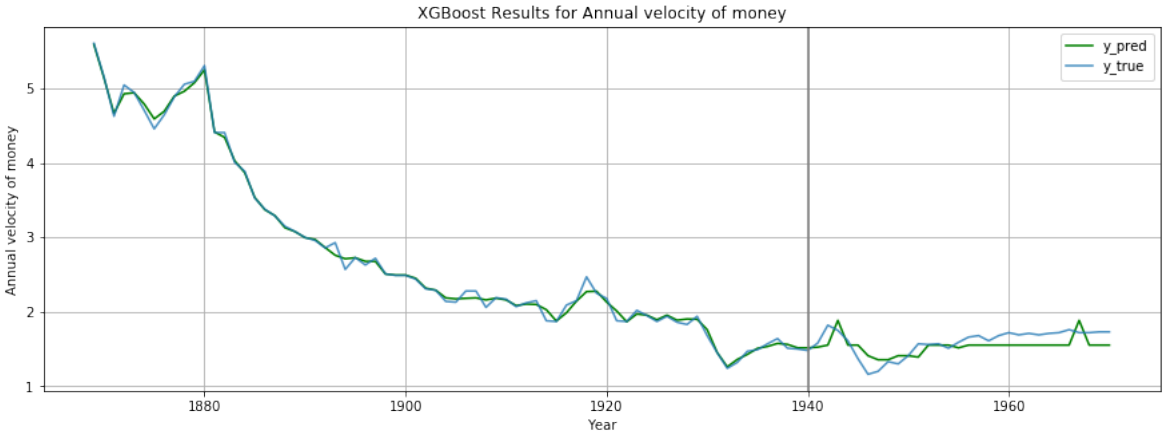


Рис. 13: Скорость обращения денег - результат с лучшими лагами

2.3 Про библиотеки

В начале мы надеялись (как выяснилось потом, надежды были наивны), что существуют библиотеки, которые сделают кучу признаков по любым рядам. Библиотеки для анализа временных рядов, действительно существуют, но, во-первых, кучу признаков они не выдают, а во-вторых, эти признаки несилько улучшали результаты наших моделей, так что нам они не очень пригодились. Тем не менее некоторые функции показались нам полезными, поэтому о них подробнее.

2.3.1 tsfresh - Python

Эта библиотека во многом повторяет функции других библиотек, не предназначенных исключительно для анализа временных рядов, и так же, как, например,`numpy` позволяет считать среднее, медиану, искать максимумы и минимумы и прочее. Но в то же время с помощью `tsfresh` можно получить и менее очевидную информацию:

- `tsfresh.feature_extraction.feature_calculators.index_mass_quantile(x, *arg, **args)` возвращает "центр масс"временного ряда, то есть такие индексы i, которые соответсвутют q% массы временного ряда, лежащего левее i;
- `tsfresh.feature_extraction.feature_calculators.large_standard_deviation(x, *arg, **args)` возвращеает единицу, если стандартное отклонение превышает разброс ряда более, чем в r раз: $std(x) > r \cdot (max(X) - min(X))$;
- `tsfresh.feature_extraction.feature_calculators.symmetry_looking(x, *arg, **args)` позволяет определить, похож ли ряд на симметричный:
  $|mean(X) - median(X)| < r \cdot (max(X) - min(X))$.

Ещё у `tsfresh` есть целый модуль `tsfresh.feature_selection.feature_selector` , который может отбирать наиболее значимые признаки из имеющихся и удалять бесполезные.

2.3.2 FATS - Python

Ещё одна билиотека, позволяющая легко получить большой объём информации о временном ряде. В частности `FATS` умеет считать индексы и коэффициенты (Variability index η^e, Welch/Stetson variability index I), строить линейный тренд по данным, тестировать их на соответсвие какому-либо распределению.

Библиотека чудесная, ещё бы импортировалась!

2.3.3 tseries - R

Библиотека, используемая в языке R для анализа временных рядов. В ней есть масса полезных функций, одна из которых - adf.test() - позволяет проверить ряд на стационарность с помощью теста Дикки-Фулера. Как ей воспользоваться? Приведём пример из интернета:

```
ad = tseries.adf_test(y, alternative="stationary", k=52)
```

В качестве параметров функции передаётся временный ряд (здесь это y) и количество лагов, для которых будет расчитываться тест (k=52). В экономических моделях, например, часто данное значение берут равным году, и если, скажем, данные в ряде еженедельные, то оптимально взять число лагов равным 52 (число недель в году).

Теперь в переменной ad содержится R-объект.

2.3.4 TTR - R

Данная библиотека аналогично содержит в себе множество полезных для анализа временных рядов функций. Приведём несколько примеров:

- `TTR.SMA(x, n = 10, ...)` сглаживает временной ряд при помощи скользящего среднего при заданной ширине окна;
- `TTR.EMA(x, n = 10, wilder = FALSE, ratio = NULL, ...)` тоже сглаживает временной ряд, но уже с помощью подсчёта экспоненциально взвешенного среднего, давая больший вес самым последним наблюдениям;
- `TTR.TDI(price, n = 20, multiple = 2)` используется для нахождения начала и конца линии тренда. Показывает хорошие результаты в нахождении начала тренда и в основном применяется для прогнозирования цен;
- `TTR.volatility(OHLC, n = 10, calc = "close N = 260, mean0 = FALSE, ...)` применяется для расчёта выбранного индикатора волатильности, как правило, на финансовых рынках. Выбрать метод подсчёта волатильности можно, введя соответствущее слово в calc. В расчёте волатильности используются цены открытия, максимума, минимума и закрытия, поэтому они должны содержаться в данных.

2.3.5 forecast - R

Библиотека включает в себя методы и инструметы для отображения и анализа одномерных временных рядов, включая экспоненциальное сглаживание с помощью state space models и automatic ARIMA modelling. Помимо классических функций включает в себя следующие интересные штуки:

- `forecast.easter` учитывает пасхальные праздники в каждом году;
- `forecast.bizdays` выдаёт число торговых дней в каждом году;
- `forecast.woolyrnq` выдаёт квартальный объём производства пряжи в Австралии:).

Приведём примеры кода для прогнозирования временных рядов с помощью ETS и automatic ARIMA из библиотеки forecast:

```
library(forecast)

# ETS forecasts
fit <- ets(USAccDeaths)
plot(forecast(fit))

# Automatic ARIMA forecasts
fit <- auto.arima(WWWusage)
plot(forecast(fit, h=20)
```

3 Не признаками едиными!

Идея создать этот раздел появилась, когда мы столкнулись с рядами, имеющими тренд на протяжении всего периода наблюдений. Проблема заключается в том, что в ситуации восходящего/нисходящего тренда модель не улавливает этой особенности и выдает прогноз, равный максимальному/минимальному значению в обучающей выборке минус/плюс колебания. И любые добавленные признаки в лучшем случае повысят качество прогноза на обучающей выборке.

Перейдём к примеру - месячным данным по индексу потребительских цен в Канаде за период 1953-1970 годов.

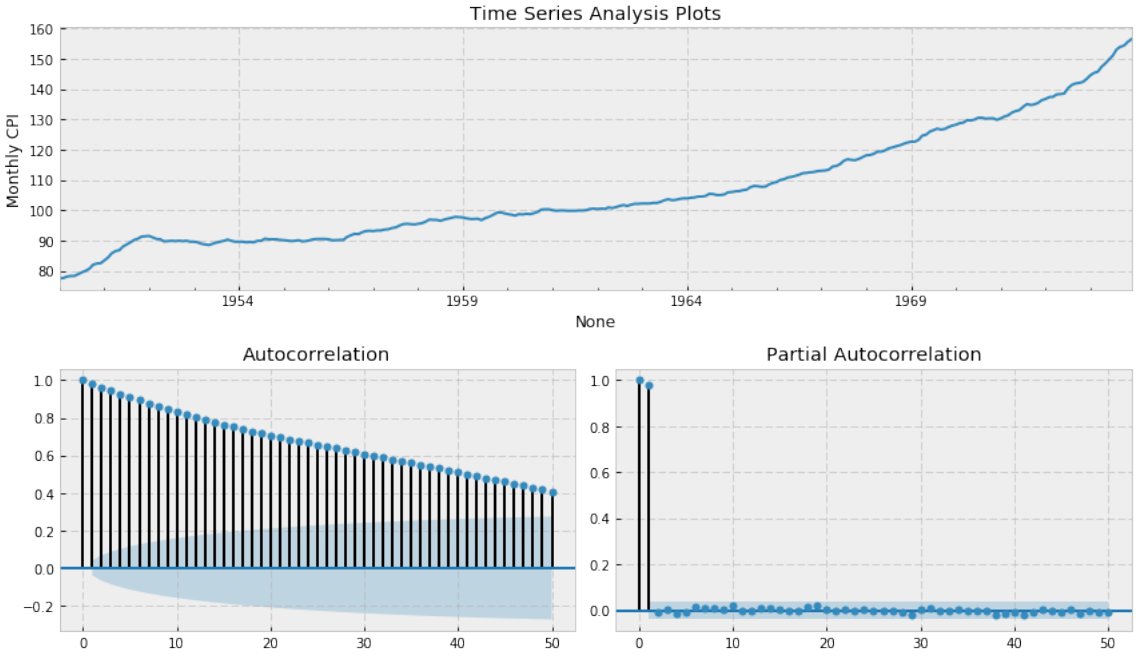


Рис. 14: Индекс потребительских цен

Отталкиваясь от наивного прогноза "завтра-как-вечра"с $MSE = 0.17$, обучаем XGBoost на 12 лагах и получаем такие ошибки: $MSE_{train} = 0.048, MSE_{test} = 139$ (139 - ужас!). Посмотрим на прогноз:

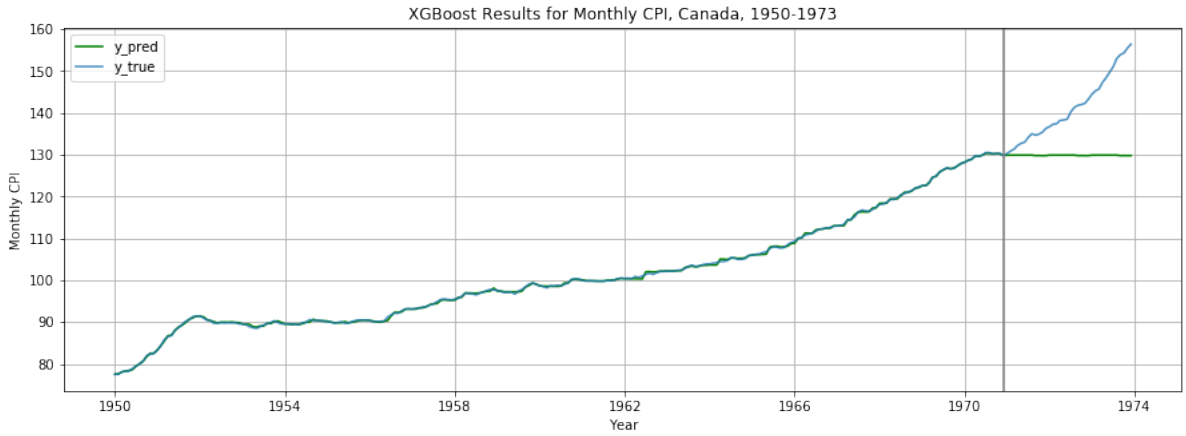


Рис. 15: Индекс потребительских цен - XGBoost с 12 лагами

В поисках решения отправляемся на форум Kaggle, откуда узнаем, что в таком случае истинные джентельмены предсказывают разность текущего и предыдущего значения. Действительно, если так сделать, то тренд исчезнет:

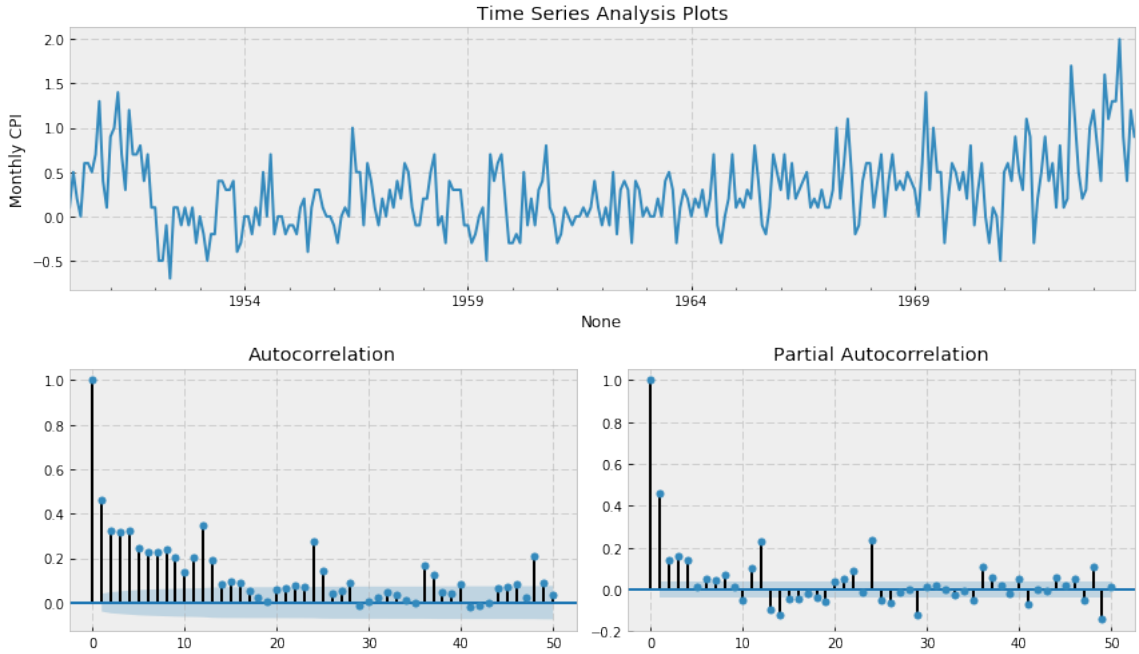


Рис. 16: Индекс потребительских цен - разность соседних значений

Дальше посутпаем, как обынчо, то есть добавляем признаки - здесь мы пользовались тремя лагами, средним значением по месяцам и закодированными месяцами и в результате получили очень маленькую ошибку $MSE_{train} \sim 10^{-7}, MSE_{test} = 0.05$

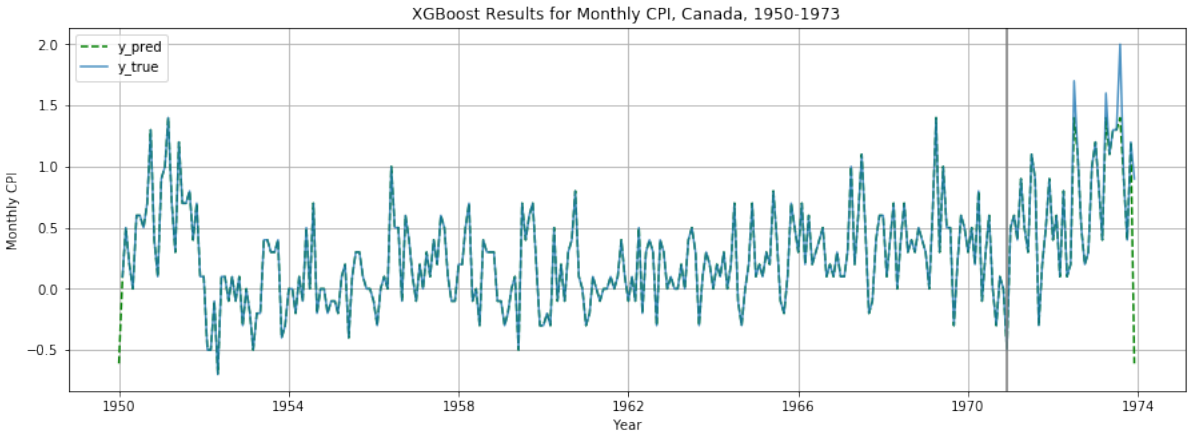


Рис. 17: Индекс потребительских цен - XGBoost, разность соседних значений

И осталось только вернуться к исходным значениям, что даст $MSE_{train} = 0.24, MSE_{test} = 0.77$. Результат на контроле не превосходит результат наивного прогноза, но последний предсказание предыдущего знаечния следующим хорошо только для завтрашнего значения и очень плохо для всех последующих. А XGBoost будет возвращать более-менее адекватные значение на протяжении длительного периода.

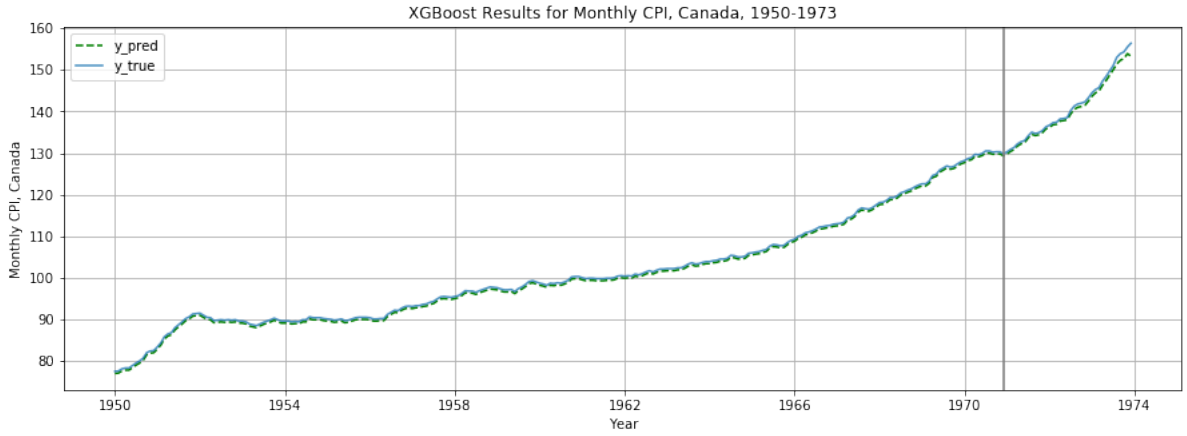
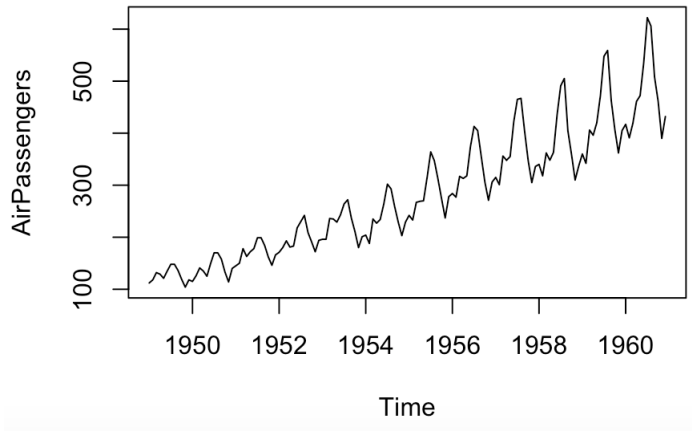


Рис. 18: Индекс потребительских цен - XGBoost, исходные значения

4 Прогнозирование временных рядов на R

Для начала импортируем все библиотеки, и читаем данные для обучения и проверки. В качестве временного ряда берем AirpPassengers, в котором собрано количество клиентов авиакомпаний с января 1949 года по декабрь 1960.

```
> library(forecast)
> library(xgboost)
> data(AirPassengers)
> AP <- AirPassengers
> plot(AP)
```



Сделаем наивный прогноз для данного ряда с помощью скользящего среднего.

```
> mapredict <- ma(AP, 2)
> plot(AP, ylim=range(c(100,600)))
> par(new=TRUE)
> plot(mapredict, ylim=range(c(100,600)), axes = FALSE, xlab = "", ylab = "",col = "red")
> title(main="Moving average prediction", col.main="black", font.main=3)
> legend("topleft",c("Actual values","Lagged values"), cex=0.5,col=c("black","red"))
```

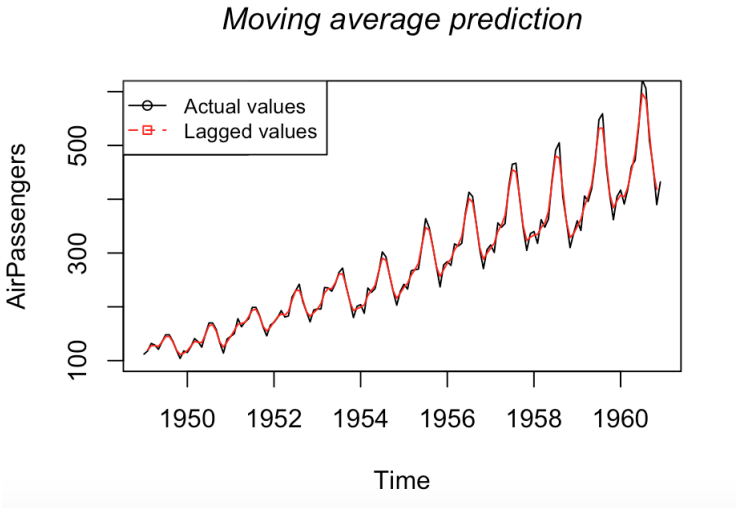


Рис. 19: Прогноз скользящее среднее

Для временных рядов можно с помощью декомпозиции получить сезонность и тренд. Для начала найдем тренд.

```
> trend_air = ma(AP, order = 12, centre = T)
> plot(as.ts(AP))
> lines(trend_air)
> plot(as.ts(trend_air))
```

Получаем такой тренд.

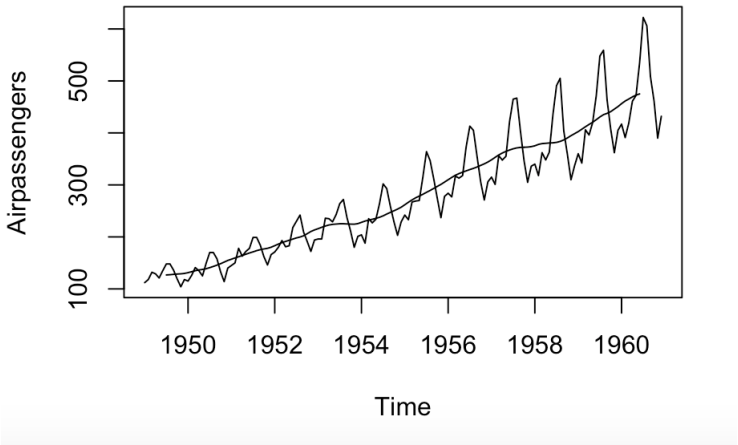


Рис. 20: Временной ряд с трендом

Убираем тренд для того, чтобы получить сезонность.

```
> trend_air = ma(AP, order = 12, centre = T)
> plot(as.ts(AP))
> lines(trend_air)
> plot(as.ts(trend_air))
```

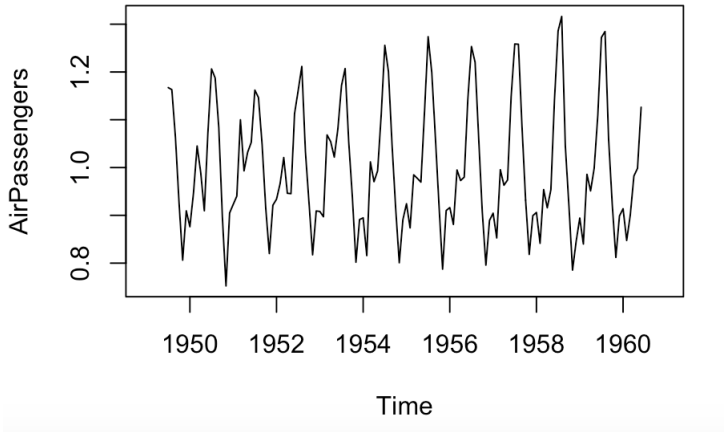


Рис. 21: Временной ряд с сезонностью

Временной ряд со средней сезонностью.

```
> trend_air = ma(AP, order = 12, centre = T)
> plot(as.ts(AP))
> lines(trend_air)
> plot(as.ts(trend_air))
```

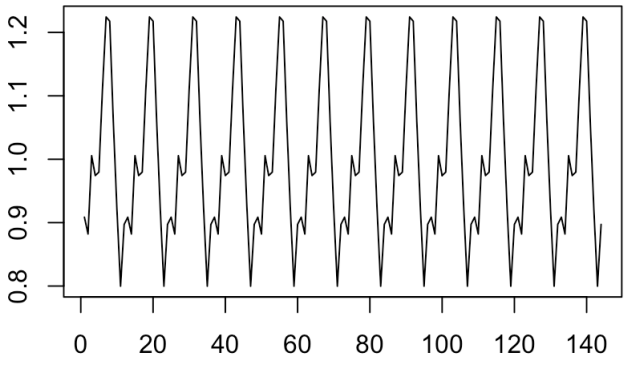


Рис. 22: Временной ряд со средней сезонностью

Ну вот и получили сезонность для нашего ряда!

Теперь перейдем к градиентному бустингу и сделаем все на R.Возьмем пятилетний временной ряд фондового рынка и попробуем прогнозировать в каком направлении будут менятся цены акций.

Читаем ряд.

```
> symbol = "ACC"
> fileName = paste(getwd(),"/",symbol,".csv",sep="") ;
> df = as.data.frame(read.csv("/Users/garik/Desktop/ACC.csv"))
> colnames(df) = c("Date","Time","Close","High", "Low", "Open","Volume")
```

Определим все характеристики. Индикатор adx(Average Directional Movement Index) определяет среднее направление движения. Данный индикатор позволяет анализировать тенденции рынка и принимать торговые решения, в том числе и на рынке forex. Индикатор SAR по своему смыслу аналогичен скользящей средней, с той лишь разницей, что движется с большим ускорением и может менять положение относительно цены. RSI это индикатор технического анализа, определяющий силу тренда и вероятность его смены.

$RSI = 100 - \frac{100}{1+RS}$

$RS = \frac{CU}{CD}$

CU - среднее значение положительных изменений цены закрытия;

CD - среднее значение отрицательных изменений цены закрытия;

Показатель RSI будем прогнозировать для нашего ряда.

```
> rsi = RSI(df$Close, n=14, maType="WMA")
> adx = data.frame(ADX(df[,c("High","Low","Close")]))
> sar = SAR(df[,c("High","Low")], accel = c(0.02, 0.2))
> trend = df$Close - sar
```

Создаем лаг в технических характеристиках, чтобы избежать сильной зависимости от предыдущих данных.

```
> rsi = c(NA,head(rsi,-1))
> adx$ADX = c(NA,head(adx$ADX,-1))
> trend = c(NA,head(trend,-1))
```

Создаем матрицу данных для обучения XGBoost.

```
> model_df = data.frame(class,adx$ADX,trend)
> model = matrix(c(class,adx$ADX,trend), nrow=length(class))
> model = na.omit(model)
> colnames(model) = c("class","adx","trend")
```

Разделяем датасет на тестовую и обучающую выборку.

```
> train_size = 2/3
> breakpoint = nrow(model) * train_size
> training_data = model[1:breakpoint,]
> test_data = model[(breakpoint+1):nrow(model),]
> X_train = training_data[,1:3] ; Y_train = training_data[,2]
> X_test = test_data[,1:3] ; Y_test = test_data[,2]
```

В конце не забываем проверять тип обучающей выборки и выборки со значениями, так как Xgboost принимает определенные типы данных при обучении.

```
> class(X_train)[1]; class(Y_train)
[1] "matrix"
[1] "numeric"
>
> class(X_test)[1]; class(Y_test)
[1] "matrix"
[1] "numeric"
```

Осталось обучить нашу модель.

```
> dtrain = xgb.DMatrix(data = X_train, label = Y_train)
> xgModel = xgboost(data = dtrain, nround = 5, objective = "reg:linear")
[1] train-rmse:18.898916
[2] train-rmse:13.327759
[3] train-rmse:9.405130
[4] train-rmse:6.646983
[5] train-rmse:4.703330
```

Ну и конечно сделаем предсказания на тестовой выборке и найдем ошибку на тесте.

```
> preds = predict(xgModel, X_test)

> error_value = mean((preds - Y_test)^2)
> print(paste("MSE =", error_value))
[1] "MSE = 19.9519728472893"
```

5 Как настроить параметры у XGboost?

Параметров у **XGBoost** очень много, но делятся они в основном на две ключевые группы: те, которые отвечают за предотвращение переобучения, и те, которые отвечают за случайность алгоритма. К первым относятся, например, `max_depth`, `reg_alpha`, `gamma`, а ко вторым: `subsample`, `colsample_bytree`, `colsample_bylevel`. Кроме того, есть параметр `n_estimators`, регулирующий количество деревьев в алгоритме. В большинстве случаев, чем больше этот парметр, тем лучше предсказание. Также есть возможность настраивает параметр `objective` в зависимости от того, какая нам необходима функция потерь (от линейной до гамма регрессии).

Рассмотрим настройку параметров на примере датасета с курсом доллара. Целевая перменная - сумма разностей курсов двух предыдущих дней. Будем проводить кросс-валидацию на некоторых признаках и сравним получившиеся результаты. Важно понимать, что далеко не всегда

оптимально сработает "жадный"принцип, когда мы отдельно максимизируем каждый параметр на кросс-валидации, а потом получаем итоговый алгоритм. Дело в том, что все параметры взаимосвязаны, поэтому часто разумно проверять на кросс-валидации связку параметров. В идеале, конечно, проверить связку из всех параметров, но на это не хватит ни времени, ни мощности.

Попробуем для начала отрегулировать `max_depth` на кросс-валидации из пяти фолдов. При этом максимальную глубину каждого дерева варьируем от 1 до 10 с шагом 1. Известно, что нет необходимости использовать очень глубокие деревья в данном алгоритме, т.к. здесь скорее важно их количество.

Получаем следующую зависимость от глубины дерева, которая как раз отражает необходимость ограничивать глубину дерева во избежание переобучения. Чем больше глубина дерева, тем лучше результаты на тренировочной выборке, но тем хуже - на тестовой. Видим, что оптимум достигается в точке `max_depth`=3.

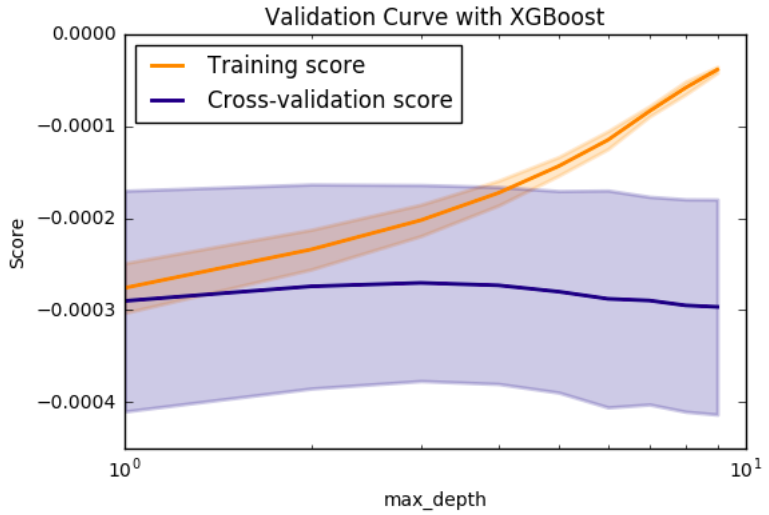


Рис. 23: Зависимость результатов прогноза XGBoost от глубины деревьев.

Построим аналогичный график для числа деревьев в итоговом алгоритме и заметим, что - MSE в данном случае имеет верхнюю грань, поэтому разумно выбирать некоторый баланс между скоростью алгоритма и качеством. Кроме того, любопытно, что здесь результаты обладают почти нулевой дисперсией, а тестовая ошибка отличается от тренировочной лишь в седьмом знаке после запятой.

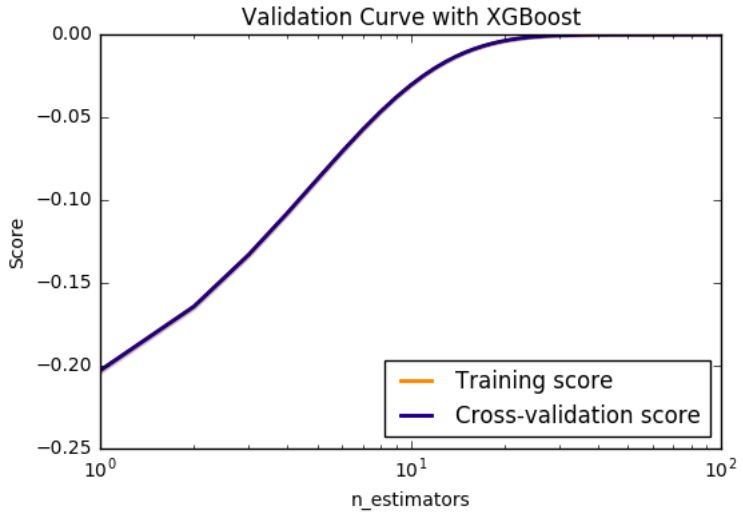


Рис. 24: Зависимость результатов прогноза XGBoost от количества деревьев.

Что касается параметров, отвечающих за случайность алгоритма, проведем эксперимент на `subsample`, показывающем, какая часть объектов используется для построения каждого дерева. Видим, что оптимум находится в точке 0.6, однако график представляет из себя практически прямую, поэтому данный параметр на ошибку почти не влияет.

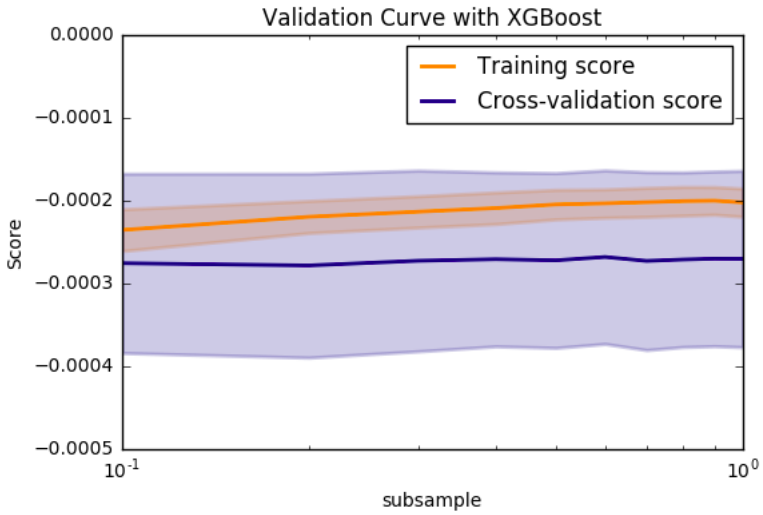


Рис. 25: Зависимость результатов прогноза XGBoost от доли объектов для построения дерева.

Если же посмотреть, как изменялась ошибка на тренировочной и тестовой выборках с настройкой каждого из вышеописанных парметров, то увидим, что negMSE на тренировочной выборке после регулирования параметров уменьшилась на 9%, а на тестовой - на 0.9%.

Однако, всегда стоит помнить, что приведенные выше графики - иллюстрация настройки параметров для конкретного датасета. Велика вероятность, что на других данных эти параметры будут вести себя иначе, поэтому необходимо каждый раз выполнять кросс-валидацию и настраивать все параметры (отдельно или в связке).

6 Источники

1. Jason Brownlee. Basic Feature Engineering With Time Series Data in Python // URL: http://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/
2. Jason Brownlee. Feature Selection for Time Series Forecasting with Python // URL: http://machinelearningmastery.com/feature-selection-time-series-forecasting-python/
3. Python API Reference // URL: http://xgboost.readthedocs.io/
4. R Documentation and manuals // URL: https://www.rdocumentation.org/