



# Регрессия временных рядов

## Прикладная математика. Отчёт по лабораторной №4

Авторы: Петренко Людмила М33001, Кусайкина Елизавета М33001, Шалимов Иван М33021

Копия отчета в Notion: [Регрессия временных рядов](#)

Код доступен по ссылке:

[https://drive.google.com/file/d/1p3IbDA0KCnz9sNGwig36XowFqJcIPAAx/view?usp=drive\\_link](https://drive.google.com/file/d/1p3IbDA0KCnz9sNGwig36XowFqJcIPAAx/view?usp=drive_link)

### ▼ Постановка задачи.

- Построить авторегрессионную модель временного ряда AR(3):

$$x_t = a_0 + a_1 x_{t-1} + a_2 x_{t-2} + a_3 x_{t-3} + \epsilon_t$$

- Проверить временной ряд коэффициентов  $a_i$  на стационарность.
- Выбрать несколько случайных значений для начальных и сгенерировать 1000 значений временного ряда соответственно авторегрессионной модели стационарного временного ряда с шумом из нормального распределения.
- Построить график временного ряда
- Преобразовать временной ряд к последовательности векторов задержек. Обучить машину опорных векторов для задачи регрессии на 80% выборки и определить авторегрессионные параметры
- Предсказать 20% ряда и сравнить на графике с исходным
- Проанализировать результаты при различных ядрах и других гиперпараметрах модели

### ▼ Модель AR(3)

Будем генерировать случайные корни характеристического многочлена, большие 1, и для них найдем соответствующие значения коэффициентов. Таким образом, мы получим коэффициенты для стационарного ряда.

```
class AR_3:  
    def __init__(self):  
        self.coefs = self.generate_coefs()  
        print(self.coefs)  
  
    def generate(self, n: int):  
        start_series = np.random.rand(1, 3)[0]
```

```

series = [i for i in start_series]

# генерируем следующие n значений
mu = 0
sigma = np.std(series)
self.error = np.random.normal(mu, sigma, n)
for i in range(n):
    series.append(1 + np.dot(np.array(self.coefs), np.array([series[a] for a in
    return series

# генерируем корни характеристического многочлена > 1 и затем коэффициенты по ним
def generate_coefs(self):
    random = np.array(np.random.rand(1, 3)[0])
    random = [abs(rand) + 1 for rand in random]
    return [-i for i in np.poly(random)][1:]

```

## ▼ Проверка на стационарность

Для проверки стационарности можно проанализировать корни характеристического многочлена модели  $a(z) = 1 - \sum_{i=1}^3 a_i z^i$

Если все корни этого полинома лежат вне единичного круга комплексной плоскости (то есть по модулю строго больше единицы), то авторегрессионный процесс является стационарным. В противном случае говорят, что характеристический многочлен имеет единичный корень (корень = 1) и процесс (ряд) является нестационарным.

```

def check_stationarity(self) -> bool:
    roots = np.roots(1 + [-a for a in self.coefs])
    if all(roots > 1):
        return False
    else:
        return True

```

Стационарность ряда также можно проверить с помощью теста Дики-Фуллера. В нем за нулевую гипотезу берется утверждение, что характеристический многочлен ряда имеет единичный корень и ряд нестационарный.

```

from statsmodels.tsa.stattools import adfuller

# проверка на стационарность
def stationary_test_fuller(timeseries):
    p_value = adfuller(timeseries[1:])[1]
    print('ряд не стационарный' if round(p_value, 4) > 0.05 else 'ряд стационарный')

stationary_test_fuller(timeseries)

```

## ▼ Обучение SVR

- Выделяем target переменную - каждая переменная, начиная с четвертой. Признаками для такой target-переменной будут являться три предыдущих значения ряда.
- Дифференцируем ряд
- Делим на тренировочную и тестовую выборки

```

n = 1000
timeseries['1'] = timeseries['series']
timeseries['2'] = timeseries['series']
timeseries['3'] = timeseries['series']
timeseries['target'] = timeseries['series']

# выделим target переменную
for i in range(0, n):

```

```

timeseries['1'].iloc[i] = timeseries['series'].iloc[i]
timeseries['2'].iloc[i] = timeseries['series'].iloc[i + 1]
timeseries['3'].iloc[i] = timeseries['series'].iloc[i + 2]
timeseries['target'].iloc[i] = timeseries['series'].iloc[i + 3]

timeseries = timeseries[:1000]

```

Полученный датасет:

<b>id</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>target</b>
0	0.806517	0.611187	0.960683	0.935874
1	0.611187	0.960683	0.935874	1.034775
2	0.960683	0.935874	1.034775	1.222354
3	0.935874	1.034775	1.222354	0.987985
4	1.034775	1.222354	0.987985	1.215371
...	...	...	...	...
995	1.059139	1.178635	1.503492	1.220625
996	1.178635	1.503492	1.220625	0.962818
997	1.503492	1.220625	0.962818	0.996754
998	1.220625	0.962818	0.996754	0.971437
999	0.962818	0.996754	0.971437	1.435239

```

# разделим данные на обучающую и тренировочную выборки
df.sort_index(inplace=True)
size = len(df)
df_train = df.iloc[:round(size*0.8) + 1]
df_test = df.iloc[round(size*0.8):]

train_x, train_y = df_train.drop(columns=['target']), pd.Series(df_train['target'])
test_x, test_y = df_test.drop(columns=['target']), pd.Series(df_test['target'])

```

- Обучение модели и предсказание

```

from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV

baseline_model = SVR()
parameters = {'C': [1000, 100, 10, 1],
              'epsilon': [1, 0.1, 0.01, 0.005, 0.001], 'gamma': ['scale', 'auto']}
clf = GridSearchCV(baseline_model, param_grid=parameters, cv=5)
clf.fit(train_x, train_y)

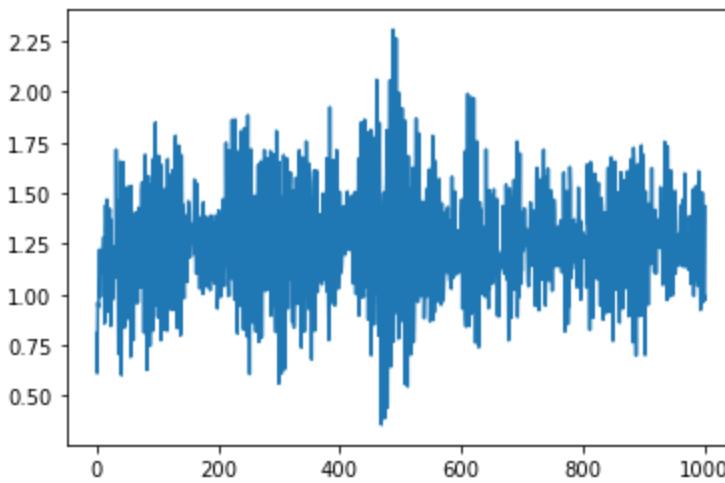
model = SVR(kernel='linear', C=clf.best_params_['C'],
            epsilon=clf.best_params_['epsilon'], gamma=clf.best_params_['gamma'])
model.fit(train_x, train_y)
predictions = model.predict(test_x)

```

## ▼ Предсказание тестовой выборки

В результате генерации ряда из 1000 значений получили:

$$x_t = 1 + 0.7x_{t-1} - 0.3x_{t-2} - 0.2x_{t-3} + \epsilon_t$$

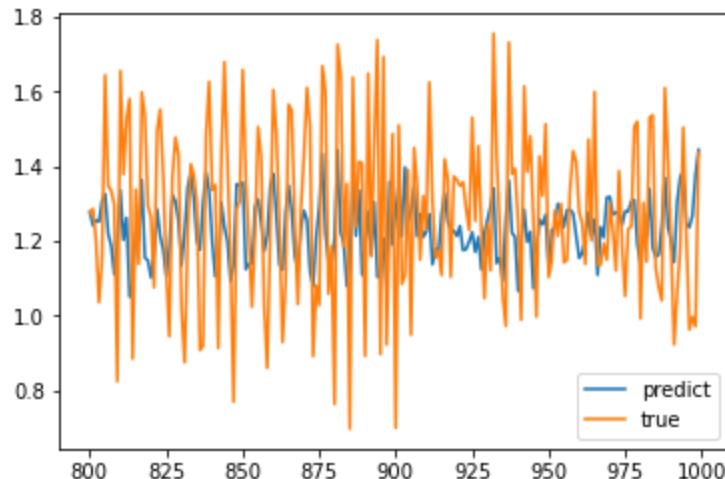
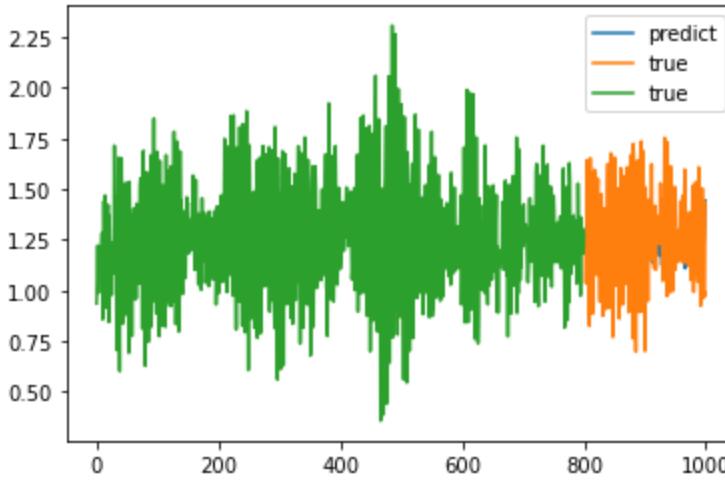


Полученный ряд является стационарным согласно тесту Дики-Фуллера и утверждению об отсутствии у характеристического многочлена единичных корней.

На обучающей выборке с помощью GridSearch были выбраны оптимальные параметры модели:

```
print(clf.best_params_)
{'C': 1, 'epsilon': 0.1, 'gamma': 'auto'}
```

После предсказания (*kernel* = 'linear'):



Ошибка на тестовых данных (посчитанная по формуле MAPE) составила

```
error = np.array(abs(predictions - test_y)/test_y).mean()
print(error)
0.1447316499139354
```

## ▼ Зависимость от гиперпараметров модели

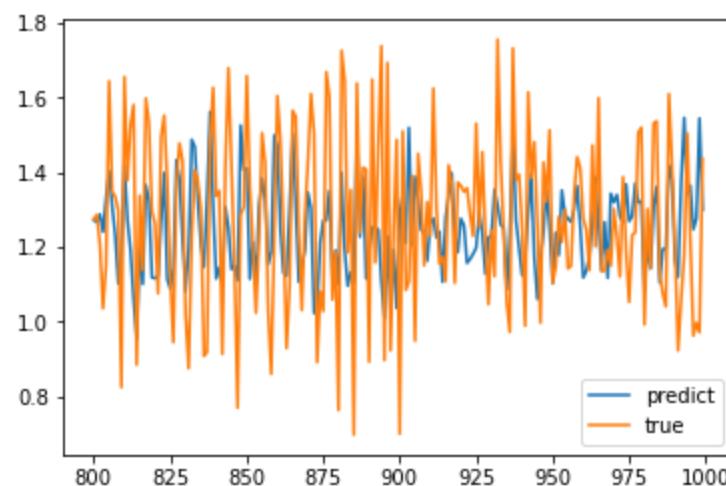
Гиперпараметры SVR:

- *epsilon* - допустимая ошибка
- *kernel* - функция ядра, может быть линейной, полиномом, сигмоидой, *rbf*(стоит по умолчанию, Radial Basis Function). Изменение функции ядра позволяет отобразить нелинейные данные в пространство, в котором они будут линейны.
- С - параметр, отвечающий за регуляризацию. По умолчанию = 1. Данный параметр стоит уменьшать в случае зашумленных данных (уменьшение *C* соответствует большей регуляризации).

- *gamma* - параметр определяет, насколько далеко распространяется влияние одного обучающего примера: низкие значения означают «далеко», а высокие значения означают «близко». Параметры *gamma* можно рассматривать как обратную величину радиуса влияния выборок, выбранных моделью в качестве опорных векторов.

kernel	epsilon	MAPE
linear	0.5	0.1456742336011584
linear	0.1	0.1447316499139354
linear	0.01	0.1454127750897814
linear	0.001	0.1449643240739544
poly, degree = 2	0.5	0.14634229440751953
poly, degree = 2	0.1	0.14562603070697813
poly, degree = 2	0.01	0.14666034133162564
poly, degree = 3	0.5	0.14539829212546465
poly, degree = 3	0.1	0.14572848130706587
poly, degree = 5	0.1	0.14702717783493113
rbf	0.1	0.1442373733599606
rbf	0.01	0.14300655290939898
rbf	0.001	0.14279075855325782
rbf	0.0001	0.1426680120453999
rbf	0.00001	0.14262901115547522
rbf	0.000001	0.14265128339329852

Лучший результат:



## ▼ Общий вывод

Наименьшее значение ошибки при предсказании сгенерированного временного ряда AR(3) было получено при использовании ядра Radial Basis Function (rbf) и допустимой ошибки  $\epsilon = 10^{-5}$