A Contest fORged by Amazon Web Services and cORe

Xingyu Bai, Tiancheng Zhao

October 2019

1 Model Selection

1.1 SARIMA

According to the demand plot, we expect that there exists some seasonality in this time series. After seasonal decomposition, we can see that seasonality information extracted from the series does seem reasonable. The seasonal decomposition result is shown in Figure 1. Where the four sub-figures plot original data, trend, seasonal effect and residuals respectively. Therefore, we consider applying SARIMA model. Initial data exploration steps are documented in attached 'data_exploration.ipynb'.

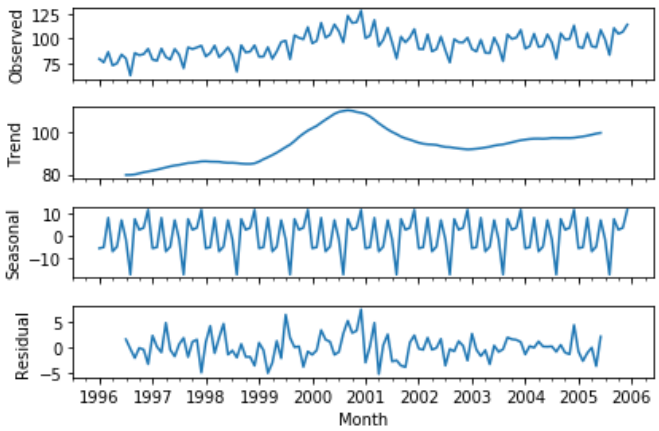

Figure 1: Seasonal Decomposition

The Seasonal Autoregressive Integrated Moving Average (SARIMA) method combines the ARIMA model with the ability to perform the same autoregression, differencing, and moving average modeling at the seasonal level. In SARIMA

model, the next step in the sequence is modeled as a linear function of the differenced observations, errors, differenced seasonal observations, and seasonal errors at prior time steps.

Use ten-year demand as our training data, we get the best model SARIMA $(p,d,q)x(P,D,Q)_m$ = SARIMA $(3,0,2)x(0,1,1)_{12}$. The AR portion (p) implies the output variable at month t depends linearly on past p months. The I portion (d) is the number of non-seasonal differences applied to the series to make it stationary. The MA portion (q) is a moving average of previous error terms. The seasonal portion $(P,D,Q)_m$ has the same structure as the non-seasonal parts and all of these factors operate across the number of period m. The results are shown in Figure 2.

Statespace Model Results

Dep. Variable:	y	No. Observations:	120
Model:	SARIMAX(3, 0, 2)x(0, 1, 1, 12)	Log Likelihood	-278.969
Date:	Sat, 12 Oct 2019	AIC	573.937
Time:	19:54:01	BIC	595.394
Sample:	0-120	HQIC	582.637
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.1789	0.221	0.809	0.418	-0.254	0.612
ar.L1	0.0584	0.106	0.552	0.581	-0.149	0.266
ar.L2	0.0731	0.089	0.823	0.410	-0.101	0.247
ar.L3	0.7798	0.104	7.479	0.000	0.575	0.984
ma.L1	0.4975	0.172	2.900	0.004	0.161	0.834
ma.L2	0.4446	0.155	2.874	0.004	0.141	0.748
ma.S.L12	-0.9991	35.418	-0.028	0.977	-70.417	68.419
sigma2	7.9514	281.298	0.028	0.977	-543.382	559.284

Ljung-Box (Q):	60.01	Jarque-Bera (JB):	2.84
Prob(Q):	0.02	Prob(JB):	0.24
Heteroskedasticity (H):	0.53	Skew:	0.06
Prob(H) (two-sided):	0.06	Kurtosis:	3.79

Figure 2: Arima Result

Next, we split the original dataset into two parts: the first 96 months as the training dataset, the last 24 months as the test dataset. For each month t in test dataset, we use the demand information before it to predict the demand arriving at that month. That is, once we have observed the demand at month t, we incorporate it into the training dataset, and fit our SARIMA $(3,0,2)x(0,1,1)_{12}$ to predict demand next month.

Applying the above method, we get the root-mean-square deviation (rmse) for the test dataset: 2.27.

1.2 XGBoost

We also consider XGBoost model.

we split the original dataset into two parts: the first 96 months as the training dataset, the last 24 months as the test dataset. The features we considered in the model include: year, month, index of month, monthly average demand, demand of previous month and demand of the month before the previous month. Note that we are constructing the features based solely on the training data.

We trained a XGBoost model based on training data, and test its performance on test data. After parameter tuning, the rmse obtained on the test set is 2.92.

1.3 Ensemble

Various other models including OLS regression, LightGBM, Prophet are also considered. These models does not generate better test data performance. So the above are two main models that we rely on predicting demand.

The bagging method in ensemble learning is applied to generate the final demand prediction, this method reduces the variance of error.

2 Policy

Due to the existence of leadtime, we consider the following policy:

- At the beginning of month t, the inventory level is x_t, which equals the ending inventory last month y_{t-1}, and we receive demand $d1_t$.
- Based on the observed demand history, we predict demand $d2_{t+1}$ arriving next month.
- After receiving ordering quantity q, which is ordered last month, the ending inventory level becomes $y_t = x_t + q - d1_t$.
- In order to minimize the total cost, we expect the ending inventory level next month is 0. Thus, we order $o = \max(d2_{t+1} - y_t, 0)$.
- We use the model developed in **section 1.3** to predict demand and compute the optimal policy. In our computation, we find that the unit backorder cost is much greater than the unit holding cost. Therefore, we consider order more each month to avoid backorder cost. Similarly, we use the first 96 months as the training dataset, the last 24 months as the test dataset, and find that in each month, ordering 3 more units will lead to the smallest total cost 112.8 for the last two years.
- Therefore, our final policy is to order $o = \max(d2_{t+1} - y_t, 0) + 3$.

Remark: We assume that there is no arriving order in January 2006, thus the cost this month can be very large.

3 ReadMe

Environment: We coded the program under Python 3.6.5, 64-bit version. Packages needed are listed as the follows: pandas, numpy, statsmodels, pmdarima, xgboost, warnings.

All the packages are the latest version installed through Anaconda's conda install command.

Document input and output: The program needs two file inputs, and they should be in the same directory as the code file 'policy.py'. The first input file is the training dataset as given by the contest website, 'Ten-Year-Demand.csv'. The second input file is the test dataset which needs to have exactly the same data format as the training file, need to be named as 'Two-Year-Test.csv'. Input file names can be changed from line 20, 21 of the code.

The output file of the code is a csv file and a txt file. File 'summary.csv' contains the computed beginning inventory, order quantity, ending inventory, holding cost, and backorder cost for each month. File 'summary.txt' contains total cost, total and average holding costs, and total and average backorder costs in summary.

To run numerical test for year 2006-2007, just run code 'policy.py'.