For office use only	Team Control Number	For office use only
T1 _____	**2018630**	F1 _____
T2 _____		F2 _____
T3 _____	Problem Chosen	F3 _____
T4 _____	**C**	F4 _____

2020
MCM/ICM
Summary Sheet

Amazon Data-Based Measures and Text-Based Reviews Analysis Using Lifelong Sentiment Classification and ARIMA Model

Summary

Our goal is to build a model to analyze and identify key patterns in the reviews made by Amazon customers and to give them advice for their online sales strategy.

First we do data pre-processing, focusing on the cleaning of special characters and stop words. We then use lifelong learning approach with Naive Bayes framework based on stochastic gradient descent to classify the text-based reviews as positive or negative, and use F1-score as the imbalanced setting and use accuracy for balanced distribution. We compare the result with other popular sentiment classifiers including SVMs and NB, with LL giving the highest precision, recall and F1-score. We compute the words frequency to analyze customers' attitude.

We use Multivariate Linear Regression to obtain the weights for star ratings, reviews and helpfulness ratings to build a comprehensive indicator for the company to track. We then analyze the underlying factors and predict the data measures by ACF and PACF, using the time-series model ARIMA.

We provide the company with a comprehensive indicator and advice the company to pay attention to the weight, the power and the size of the hairdryer, which are some important factors for the customers from our analysis.

To: Sunshine Company

After receiving your requests, we are pleasant to tell you that we finally solved your problems and can give you some advices on how to build an online sales strategy basing on our results, by using lifelong learning model, multivariate linear regression model and ARIMA model.

Firstly, we recommend your to pay close attention to low star ratings. Based on our research, we found that low star rating might incite more negative reviews in the future. We advice you pay attention when the number of star rating increase and talk to the customers in order to understand the reason for these low ratings and improve the product accordingly.

Secondly, we recommend you to use the comprehensive indicator provided by us, with the weight of the parameters: star rating, text-based measure, vine and verified purchase being 0.0031, 0.001737, 0.1225 and 0.429659 correspondingly. This should be the most informative comprehensive measure and the best combination of the data sets. It can be very useful for you to track customers' true opinions.

Thirdly, by analyzing time-based measures and patterns within each data set during the period that a product reputation changes, you can improve your products' reputation and attract more customers. We provided the methods in our solution. To take hairdryer as an example, we discovered that three words, "light", "powerful" and "retractable" have the highest frequency in the reviews during these time periods where the product's reputation is increasing, which can in turn indicate that a potentially successful product should have this merits.

Also, for hairdryer, by analyzing word frequency, we discovered that the weight, the power and the size are the three of the most important features for the customers which should also be the most important features for your company when designing and introducing new product. We provided the method in our solution, which can also be used for your other two products.

We hope your company and your new products can make well on the market.

Best regards.
Mar 9, 2020

Contents

1 Introduction **3**

1.1 Restatement of the Problem 3

1.2 Literature Review 3

1.3 Overview 4

1.4 Assumptions 5

2 Text Processing and Quantification **5**

2.1 Datasets 6

2.2 Naïve Bayesian Method 7

2.3 Objective Function Optimization 8

2.4 Stochastic Gradient Descent (SGD) Method 9

2.5 Implementation 9

3 Model for Comprehensive Indicator **10**

3.1 Definition of Comprehensive Indicator 11

3.2 Principle of MLR Model 11

3.3 Model Result's Analysis 12

4 Time Series Analysis with ARIMA Model **13**

4.1 Explanation of the Model 13

4.1.1 Definition of ARMA Model 13

4.1.2 Integrated Process 13

4.2 Result Analysis 14

4.3 ACF and PACF of ARMA Process 15

4.4 Discussions and Results 16

4.4.1 Information Criterion-Akaike Information Criterion (AIC) 16

4.4.2 Data Application 16

5 Conclusions **18**

References **19**

Appendices **22**

Appendix A LSC Code **22**

Appendix B Data **38**

Appendix C WordFreq **40**

Appendix D ARIMA **40**

1 Introduction

1.1 Restatement of the Problem

Sunshine Company is planning to introduce three new products in the online marketplace: a microwave oven, a baby pacifier, and a hair dryer. In order to gain insights into the markets in which Sunshine company will participate and to design a good online sale strategy by identifying some potentially important design features, we first need to identify informative data measures based on star ratings, text-based reviews and helpfulness ratings on Amazon and then realise sentiment classification. We then need to use Autoregressive Integrated Moving Average Model (ARIMA) to find a optimized combination of text-based measure and ratings-based measures that can best indicate a potentially successful or failing product, which can help the company to seek a time-based pattern in order to satisfy customers' demands for their future products. After this, we need the solve the problem of whether specific previous star ratings has influence on later text-based reviews and whether some particular words in the reviews has strong association with rating levels.

1.2 Literature Review

Many well-known classifiers have been proposed in literature and been very successful in application, such as random forest (RF) classifier, support vector machines (SVMs) and Naive Bayes (NB). However, most of them still have some limitations in applications. SVM is a supervised machine learning technique proposed by Vapnik et al. (1995) that can be as a classifier and works well with high dimensional data (Terrence et al., 2000). Meanwhile, Galiano et al. (2012) mentioned that as a powerful machine learning classifier, random forest has some key advantages including its non-parametric nature and capability to determine variable importance. Pal (2005) also stated that random forest classifier requires less user-defined parameters than SVMs and is easier to define. However, Noi et al. (2018) did comparison between three different classifiers including RF, k-Nearest Neighbor (kNN) and SVMs and concluded that even though they all produced high overall accuracy, SVMs provided the highest overall accuracy and the least sensitivity to the training sample sizes among these three. When classifying tree species, Raczko (2017) also compared the result of RF and SVMs and found out that SVMs' median overall classification accuracy is 5 percent higher than the result produced by RF. Similar result was also given by the experiment made by Hasan et al. (2014). Another simple but still efficient classifier, naive Bayes clas-

sifier, can compete with more sophisticated classifiers even thought it is based on the normally false assumption that features are independent given class (Rish, 2001). Pang et al. (2002) analysed the performance of NB, ME, and SVM in movie reviews dataset, with result showing that NB classifier has the worst result and SVM produced the best results, but with not large difference.

When mentioning lifelong machine learning, some traditional learning paradigm are often mentioned, such as multi-task learning and transfer learning. Multi-task learning is an approach that can improve the performance of one task by using the help of some other related tasks (Zhang and Yeung, 2012). It can be applied to different domains and used with different learning algorithms, which makes it useful in real life applications (Caruana, 1997). The main idea of transfer learning is that after gaining experience of one task, the performance of the next task can be improved (Taylor and Stone, 2009). By greatly avoiding the expensive data-labeling efforts, Transfer learning can be used when the data can be easily outdated (Pan and Yang, 2010). However, both of these paradigm face significant challenges. Multi-task learning can not be used for sentiment analysis (Chen, Ma and Liu, 2018) and it also face the problem of excessive storage space required, which can be expensive. Meanwhile, transfer learning does not use either the results of the previous tasks nor the knowledge learned from the past learning process (Chen, Ma and Liu, 2018).

1.3 Overview

Sentiment classification is the process that classify a review or an opinion as a positive or negative sentiment (Chen, Ma and Liu, 2018). In this paper, in order to realise sentiment classification and analysis, we adopted the lifelong learning (LL) approach with naive Bayes framework based on stochastic gradient descent. Lifelong learning is a learning paradigm that has most of the advantages of both transfer learning and multi-task learning. meanwhile, it successfully avoids the main limits of these two. As mentioned before, multi-task learning frequently requires an overwhelmingly amount of storage space, lifelong learning, however, largely reduce the storage space by producing a generator that allow us reproduce previous data instead of remembering all of them. Lifelong learning is also more sophisticated than transfer learning in that it can transfer as well as retention the results and the knowledge learned from the past learning process, which can increase overall accuracy. We compared the overall accuracy produced by LL and some traditional methods such as SVMs,

RF and NB and found out that our method provides the highest overall accuracy.

After transforming the text-based measures into the data-based measures, we then used ARIMA to realise the time-series analysis between all three measures including reviews, star ratings and helpfulness ratings and got our final result, and then gave the company advises for their future product sales basing on the analysis of those results.

1.4 Assumptions

Before establishing model and solving problems, some assumptions should be clearly determined:

- It is assumed that customers' reviews on the same day are independent. There is no link between customers' reviews that are written simultaneously.
- The quality of the same product with identical product-parent is consistent so that we can objectively judge the product.
- The abnormal factors, such as sudden economic policy and natural disaster influencing delivery speed, are excluded and omitted from the model.

2 Text Processing and Quantification

In natural language processing, reservation of English words, word correction, stop word elimination and word pruning, this four steps are involved to enhance the efficiency of later lifelong sentiment classification. These steps scale down word sets by deleting useless words and merging words of similar meanings, and therefore reduce the dimensions of objective functions of lifelong sentiment classification, which easily leads to optima for stochastic gradient descent and saves time.

After exploration, we use a training algorithm incorporating past data and knowledge. For text processing, we focus on binary classification, i.e. positive (+) and negative (-) polarities, to judge the attitude of the text. To make breakthrough for lifelong machine learning, we propose an optimization method according to a combination of the Naïve Bayesian

foundation and stochastic gradient descent. To be specific, the model incorporates the penalty term in order to strengthen the predictive ability and reasonableness of the model. Binary classification review sample in Table 1 about hair-dryer is presented in the table.

Positive (+)	Negative (-)
Great price and does its job	The plastic melted and didn't last long
Good weight, feel and ergonomics; using this with little kids and big adults.	Not powerful enough and not a good value for the money
I love this hair dryer; the available speeds and temperatures are perfect and it's the most quiet hair dryer	Not good for my curly hair

Table 1: Review Sample

2.1 Datasets

The source of data is related to records of Amazon in terms of hair-dryer, pacifier and microwave. Each product contains more than 8000 reviews which are helpful to build train sets and test sets. In the Amazon rate and review system, "Star ratings", "Reviews" and "Helpfulness rating" are factors of significance influence. We study text-based review in this section and combine these factors with MLR model in next section. Inspired by lifelong machine learning, we form our own over these 3 domains, in which one can make use of knowledge from past tasks $(1, 2, ..., M-1)$ with binary labels to learn a better classifier for next M-th task. And the significant componpents of lifelong Sentiment Classification are Past Information Store(PIS) in Table 2 and Knowledge Base(KB) in Table 3.

$\mathcal{P}^t(w\|+)$, $\mathcal{P}^t(w\|-)$	The probability of each word w to appear in the positive/negative class for a past learning task t.
$N^t_{+,w}$, $N^t_{-,w}$	The number of times of each word w to appear in a positive/negative document for a past learning task t.

Table 2: Past Information Store(PIS)

$N^{KB}_{+,w}$, $N^{KB}_{-,w}$ and formula $N^{KB}_{+,w} = \sum_t N^t_{+,w}$, $N^{KB}_{-,w} = \sum_t N^t_{-,w}$	Document-level knowledge: Number of occurrences of word w in positive/ negative documents in whole past tasks
$M^{KB}_{+,w}$, $M^{KB}_{-,w}$	Domain-level knowledge for 3 domains: Number of past tasks for a word w such that $\mathcal{P}(w\|+) > \mathcal{P}(w\|-)$ (or $\mathcal{P}(w\|+) < \mathcal{P}(w\|-)$)

Table 3: Knowledge Base(KB)

For Nature Language Processing, data pre-processing is explained by the following graph. Raw data are usually cleaned after segmentation operation, such special tags and stop words. We do feature extraction from Standardized data and further build models.

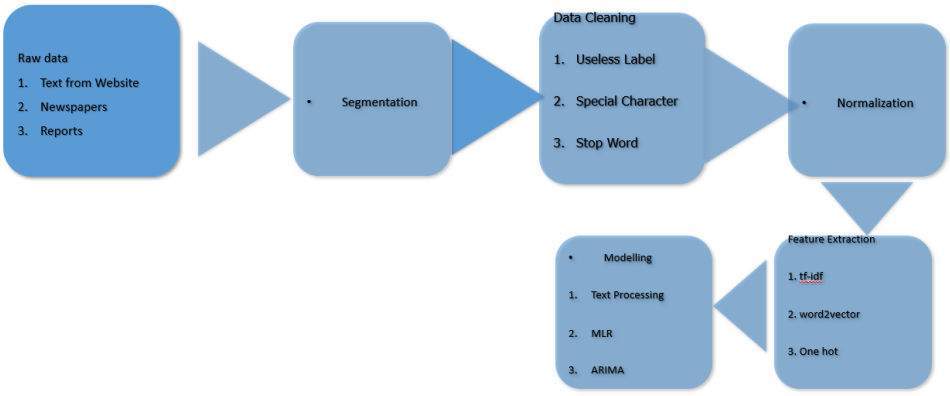

Figure 1: Steps for Data Processing

In reviews of hair-dryer, for instance, customers provided feedback and evaluated products from varied perspectives including price, service time, plastic material, suitability in personal life and appearance etc.

2.2 Naïve Bayesian Method

Naïve Bayesian(NB) text classification fundamentally calculates $\mathcal{P}(w|c_j)$ meaning conditional probability of each word w based on each class c_j and $\mathcal{P}(c_j)$ representing prior probability of each class, utilized to compute posterior probability of each class according to a test document d (i.e. $\mathcal{P}(c_j|d)$) to determine whether c_j is positive or negative.

$\mathcal{P}(w|c_j)$ is calculated as follows:

$$\mathcal{P}(w|c_j) = \frac{\lambda + N_{c_j,w}}{\lambda|V| + \sum_{v=1}^{|V|} N_{c_j,w}} \tag{1}$$

Here, in this equation, $N_{c_j,w}$ is frequency of word w in documents of class c_j and $|V|$ represents size of vocabulary V and λ means parameter for Laplace smoothing.

2.3 Objective Function Optimization

The classification is viewed as an optimization problem in which we will minimize the loss function with respect to the parameters of the score function.

In order to implement stochastic gradient descent (SGD), we introduce optimized variables $X_{+,w}$, $X_{-,w}$, which are called virtual counts to replace empirical counts $N_{+,w}$, $N_{-,w}$. For accurate sentiment classification, we expect the posterior probability $\mathcal{P}(c_j|d_i) = 1$ for labeled class c_j and $\mathcal{P}(c_f|d_i) = 0$ for other class c_f. According to new training data D^t, objective function is

$$\sum_{i=1}^{|D^t|}(\mathcal{P}(c_j|d_i) - \mathcal{P}(c_f|d_i)). \tag{2}$$

Taking positive document as an example, The objective function under stochastic gradient descent for a positive document is

$$F_{+,i} = \mathcal{P}(+|d_i) - \mathcal{P}(-|d_i). \tag{3}$$

To apply the knowledge base, we utilize penalty term into optimization method. For word w satisfying the condition of

$$\frac{\mathcal{P}(w|+)}{\mathcal{P}(w|-)} \geq \sigma \;\; OR \;\; \frac{\mathcal{P}(w|-)}{\mathcal{P}(w|+)} \geq \sigma, \; (\sigma \; is \; a \; parameter), \tag{4}$$

we add following penalty term:

$$\frac{1}{2}\alpha \sum_{w \in V_T} (X_{+,w} - N^t_{+,w})^2 + (X_{-,w} - N^t_{-,w})^2. \tag{5}$$

For word w satisfying another condition of

$$M^{KB}_{+,w} \geq \tau \;\; OR \;\; M^{KB}_{-,w} \geq \tau (\tau \; is \; a \; parameter), \tag{6}$$

we add respective penalty term:

$$\frac{1}{2}\alpha \sum_{w \in V_S} (X_{+,w} - R_w \cdot X^0_{+,w})^2 + \frac{1}{2}\alpha \sum_{w \in V_S} (X_{-,w} - (1 - R_w) \cdot X^0_{-,w})^2. \quad (7)$$

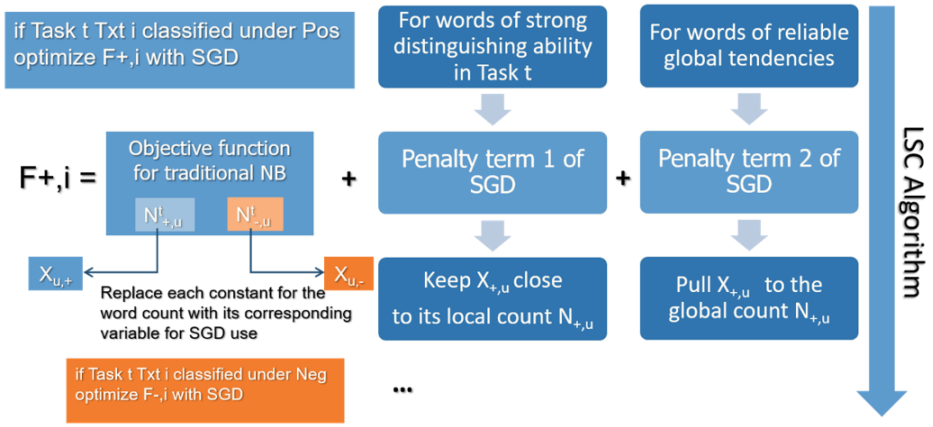

Figure 2: Main Steps

2.4 Stochastic Gradient Descent (SGD) Method

We use $X^0_{+,w} = N^t_{+,w} + N^{KB}_{+,w}$ and $X^0_{-,w} = N^t_{-,w} + N^{KB}_{-,w}$ as the starting point for SGD. This optimization process is conducted according to the following equations where γ is learning rate:

$$X^l_{+,u} = X^{l-1}_{+,u} - \gamma \frac{\partial F_{+,i}}{\partial X_{+,u}}, X^l_{-,u} = X^{l-1}_{-,u} - \gamma \frac{\partial F_{+,i}}{\partial X_{-,u}} \quad (8)$$

It iterates until the difference of $F_{+,i} = \mathcal{P}(+|d_i) - \mathcal{P}(-|d_i)$ for two successive iterations is smaller than 10^{-3}.

2.5 Implementation

As an important assumption, we consider a review as positive when its star rating is larger than 3 and vice versa. And, with word frequency result, we will proof that as rating grows, review is more positive. Implementing by numpy and pandas toolkits, we represent each text with pd.Series and each task with pd.DataFrame and then summarize counting results and calculation of probabilities into pd.DataFrame. This storage form enables replacement of large scale for-loops with .apply, which

operates on matrices of pandas in its fundamental layer of C to circumvent the defect of interpreting line by line of python. It also realizes quick retrieval to saves the time cost of traversal.

What's more, we use some criteria to test different results from machine learning methods. F1-score is chosen for imbalanced setting and Accuracy is used for balanced distribution. And Precision and recall are two typical indicators to evaluate results of classification. Precision measures the accuracy of classifier (i.e. $Precision = \frac{system.correct}{system.output}$) while recall is understood as sensitivity and regarded as the probability that a relevant document is retrieved by the query (i.e. $Recall = \frac{system.correct}{human.labeled}$). Combining two indicators, F1 Score is the weighted average of Precision and Recall. To seek balance of those two values, $F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$.

In order to show the advantages of the LL model, we compared other popular sentiment classifiers, such as SVM-one_hot, SVM-word2vec, Naïve Bayes (NB), Random Forest (RF). From the results, we clearly found that LSC has a significant F1-score and accuracy in both natural and balanced distributions in Table 4.

Methods	Accuracy	F_1 Score
SVM-one_hot	76.12	70.74
SVM-word2vec	70.86	73.79
NB	62.51	66.33
RF	65.37	77.32
LSC	80.64	79.47

Table 4: Baseline Result Comparison

After implementation, we obtain statistics of word frequency and realise text quantification from initially viewing each text as a vector and conducting sentiment classification. Based on text-based value and other known ratings, it is necessary to construct a comprehensive indicator for later analysis in terms of diversified products.

3 Model for Comprehensive Indicator

In the business analysis field, there exists many models and ways helping researchers to analyze large amount of numerical data. Here, Multivariate Linear Regression(MLR) is applied to obtain the corresponding weights

for each rating-based measures concerning star ratings, quantitative value for reviews and helpfulness ratings.

3.1 Definition of Comprehensive Indicator

According to star ratings, reviews and helpfulness ratings, we first determine the number of parameters based on the balance between R-Square and RMSE and thus suitably adjust parameter estimation by computer implementation, and finally use estimated parameter to attain the comprehensive indicator.

3.2 Principle of MLR Model

According to Montgomery (2012), least squares method is a mathematical regression estimator that discovers the best fit line for a dataset and provides a visual demonstration of relationship among data points. Multiple linear regression (MLR) model is written in matrix form $Y = X\beta + \epsilon$ with $\epsilon \sim N(0, \sigma^2)$. Every value in n-dimensional vectors Y can be obtained from the following equation (Florea et al., 2016):

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \epsilon_i,\ i = 1, 2, ..., n \tag{9}$$

Typically, each $\overrightarrow{x_i} = (x_{i1}, x_{i2}, ..., x_{ip})^T$ is a vector of feature measurements for i-th case and a set of training data $(\overrightarrow{x_1}, Y_1)...(\overrightarrow{x_n}, Y_n)$ exists, from which we can estimate the parameter $\hat{\beta}$. $\hat{\beta}$ with $p + 1$ unknown parameters involving $\hat{\beta}_0, ..., \hat{\beta}_p$ are chosen to minimize the residual sum of squares,

$$S = \sum_{i=1}^{n} (Y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}))^2. \tag{10}$$

The term $\hat{\beta}_0$ is the intercept, also known as the bias in machine learning. If $X^{\mathrm{T}}X$ is invertible, then

$$\hat{\beta} = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}\mathbf{Y} \tag{11}$$

is attained as a closed-form solution which is unique for solving OLS including a system of linear equations (Rice, 2006).

3.3 Model Result's Analysis

Two statistical criterion including R-square and Root mean square error (RMSE) are utilized in MLR to assess the fit of model. The R-square indicates goodness-of-fit of the regression model and RMSE evaluates the distance between the values and the mean.

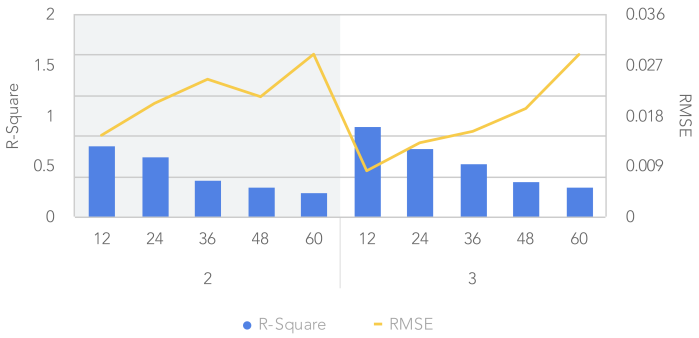

Figure 3: Statistics Balance between R-Square and RMSE

As training period increase or the number of variables decrease, the fitting degree will be worse but the testing accuracy will be better, and vice versa. R-squre and RMSE are calculated for each case and the one chosen eventually has acceptable outcomes. In particular, the abnormal criterion are considered as signal of overfitting. We conclude in Table 5, where "$+$" means uptrend and "$-$" means downtrend.

Table 5: Indicators influence on R-Square and RMSE

Indicator	R^2	RMSE
Training Period increases	-	+
number of parameters increases	+	-

After multivariate linear regression model, parameters $\theta_0, ..., \theta_3$ concerning star rating, text-based measure, vine and verified purchase is obtained to make up comprehensive indicator.

$$[\theta_0, \theta_1, \theta_2, \theta_3] = [0.0031, 0.001737, 0.1225, 0.429659] \tag{12}$$

Continuing with known data, comprehensive indicator $\widehat{Y}$ is constructed.

4 Time Series Analysis with ARIMA Model

Time series data is a sequence of observations recorded in a successive time order. Indeed, there exists correlation between observations and trying to find this relationship can sometimes be challenging.

Autoregressive integrated moving average model is the combination of ARMA model and Integrated model. When judging whether a model fits well to the data, we evaluate the correlogram of sample autocorrelation function (acf) or partial correlation function (pacf), which is the plot of acf and pacf against time lag. Diversified pattern of correlogram indicates different appropriate models for the data.

4.1 Explanation of the Model

4.1.1 Definition of ARMA Model

X_t is an ARMA(p,q) process if X_t is stationary and if for every t,

$$X_t - \phi_1 X_{t-1} - ... - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + ... + \theta_q Z_{t-q},\ where Z_t \sim WN(0, \sigma^2) \tag{13}$$

and the polynomials $(1 - \phi_1 z - ... - \phi_p z^p)$ and $(1 + \theta_1 z + ... + \theta_q z^q)$ have no common factors. (Brockwell and Davis, 1996)

4.1.2 Integrated Process

Some time series data do not satisfy stationarity owing to trends and therefore stationary time series model is unsuitable. By differencing, in fact, many non-stationary time series may be modified to one with stationarity.
The number of times for differentiation of the process to realize stationarity is called the order of integration. If it is necessary to conduct d times differentiation, then the order of the integrated process is d and we use $X_t \sim I(d)$ to denote it. Here, the order is d, therefore we only need to calculate the other two parameters for ARIMA Model. Then this model is converted to ARMA Model. The flow chart of ARIMA process is as follows.

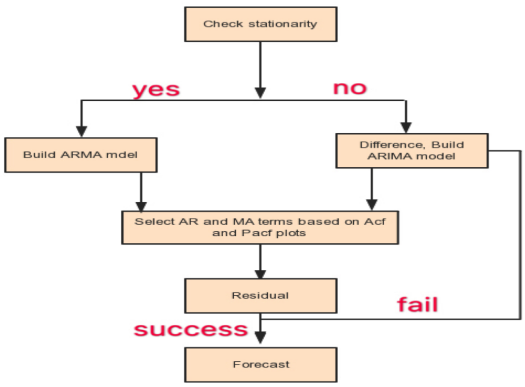

Figure 4: Flow chart for ARIMA Model

4.2 Result Analysis

The outcomes about p-value of Phillips-Perron Test are Table 6,

Domain	original data	first-order differentiation
Hair-dryer	0.8376	< 0.01
Pacifier	0.8145	< 0.01
Microwave	0.7876	< 0.01

Table 6: Stationary Test

H_0: Time series data of ratings is not stationary in the model.
H_1: Time series data of ratings is significantly stationary.

In the table, p-value is large to some extent. Therefore, it has very little evidence against the null hypothesis H_0. However, after first-order differentiation, every p-value is small (< 0.01), indicating that transformed time series do not have a unit root.

4.3 ACF and PACF of ARMA Process

Two essential functions called ACF and PACF can help to determine the order of ARMA model. To compute the ACF, it is necessary to first calculate the auto-covariance (ACVF). From Equation

$$X_t - \phi_1 X_{t-1} - ... - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + ... + \theta_q Z_{t-q}, \tag{14}$$

by multiplying both side with X_{t-1}, $k = 0, 1, 2, ...$ and taking expectations on both side, we obtain that

$$\gamma(k) - \phi_1 \gamma(k-1) - ... - \phi_p \gamma(k-p) = 0, k \geq m, \tag{15}$$

and for $0 \leq k \leq m$,

$$\begin{aligned} &\gamma(k) - \phi_1 \gamma(k-1) - ... - \phi_p \gamma(k-p) \\ &= \sigma^2 [\sum_{j=0}^{\infty} \psi_j (\psi_{j+k} - \phi_1 \psi_{j+k-1} - ... - \phi_p \psi_{j+k-p})] \\ &= \sigma^2 \sum_{j=0}^{\infty} \theta_{k+j} \psi_j. \end{aligned} \tag{16}$$

Here, $m = max(p, q+1)$, $\psi := 0$ for $j \leq 0$, $\theta_0 := 1$ and $\theta_j = 0$ for $j \notin \{0, 1, ..., q\}$. The solution is $\gamma(h) = \alpha_1 \xi_1^{-h} + \alpha_2 \xi_2^{-h} + ... + \alpha_p \xi_p^{-h}, h \geq m - p$, where $\alpha_1, ..., \alpha_p$ are constants and $\xi_1, ..., \xi_p$ are the roots of $\phi(z) = 0$. According to known ACVF function, the ACF of an ARMA model $\rho(.)$ is attained by $\rho(h) = \frac{\gamma(h)}{\gamma(0)}$.

The PACF at lag h is the ordinary least square estimation of ϕ_h for AR(h) model, i.e. ϕ_h minimizes

$$S(\phi_1, ..., \phi_h) = \sum_{t=k+1}^{T} (y_t - \phi_1 y_{t-1} - ... - \phi h y_{t-h})^2$$

.

4.4 Discussions and Results

4.4.1 Information Criterion-Akaike Information Criterion (AIC)

The criterion for selecting the orders p and q for ARMA model is written as general form:

$$ln\hat{\sigma}^2_{p,q} + (\#free\ parameters)\frac{C(T)}{T} = ln\hat{\sigma}_{p,q} + (p+q)\frac{C(T)}{T} \to \min_{p,q}, \quad (17)$$

where $ln\hat{\sigma}^2_{p,q}$ is relevant to log-likelihood and $C(T)$ is the penalty term for using more parameters. The formula for AIC is $AIC(p,q) = ln\hat{\sigma}_{p,q} + (p+q)\frac{2}{T}$. Fitting in varied models and comparing AIC to choose the model with lowest AIC value is the aim.

4.4.2 Data Application

In the previous step, we know comprehensive ratings for three categories are not stationary but are stationary for first-order differentiation. The correlogram and partial correlogram are plotted using acf and pacf.

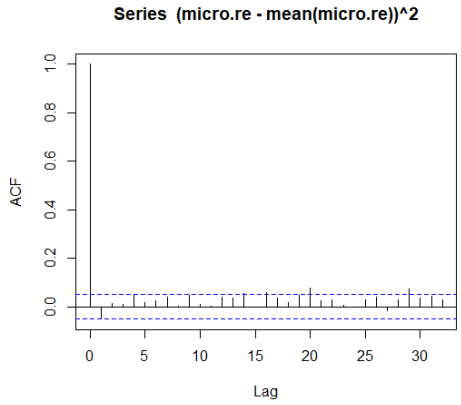

Figure 5: Microwave's ACF Plot

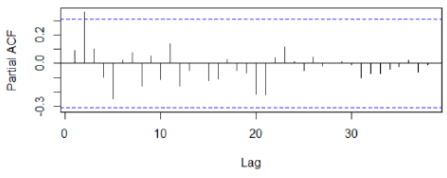

Figure 6: PACF of Microwave

In Figure 4 and 5, it is obvious that both acf and pacf exceed the blue line. If arbitrary γ_k falls outside the line, indicating not in the range of confidence interval, it has evidence against null hypothesis that $\rho_k = 0$ at the 5% level.

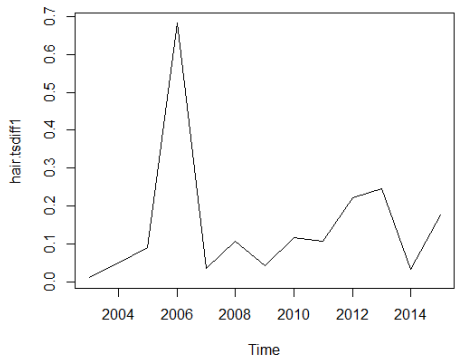

Figure 7: Hair-dryer Plot

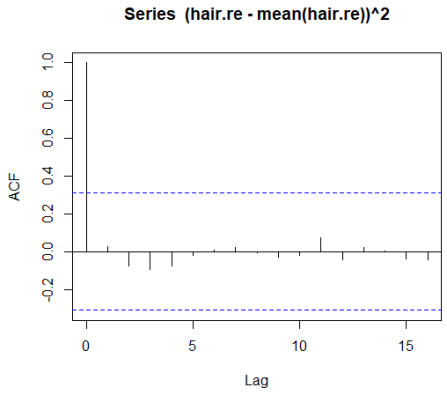

Figure 8: ACF of Hair-dryer

Considering the fact that comprehensive indicator varies with time, comprehensive indicator representing the combination of different ratings is in the form of time series data. To be highlighted, the comprehensive indicator ranging from 0 to 1 in terms of Hair-dryer involves the interval between 2002 and 2015 and this becomes time period that is considered. In Figure 7, there exists significant performances of ascending and descending trend to motivate us to help Sunshine Company find underlying factors and main features or changes of product.
If $\frac{\partial y}{\partial t} > 0$ and $\frac{\partial^2 y}{\partial t^2} = 0$ shows increase with quickest speed. To find characteristic of product which will achieve potential success, we consider many aspects popular degree in market, cost and sustainability etc. based on customers' feedback. By word frequency analysis on hair-dryer, from year 2005 to around year 2006, top 5 words with positive sentiment are as follows. Similarly, we also analyze the information between 2006 and 2007 in Table 7.

2005-06	Frequency	2006-07	Frequency
five star	73	small	52
recommend	70	thick	49
light	66	low	48
powerful	59	heavy	40
retractable	42	disappoint	35

Table 7: Strategies of Word Frequency

Based on time-series data considering reviews, discounting and verified purchase etc., we intend to analyze process and to predict customers' true thoughts. The part of Prediction is presented in the chart below.

product_pa	status	status_pred	prob_0	prob_1
16483457	1	1	0.3428	0.6572
815035474	0	1	0.3475	0.6525
423421857	1	0	0.5215	0.4785
			1-positive	
			0-negative	

Figure 9: Prediction

Through exploration in extremely positive or negative orientation and forecasting further situation, the transition probabilities are considered. After some low star ratings, for instance, it is likely to incite reviews with pessimistic sentiment, but simultaneously how to tackle the phenomenon or issues might be laid emphasis on and thus effective adjustments in product sales could result in enhancing reputation.

5 Conclusions

In conclusion, we solved the questions and requests from the Sunshine Company by sentiment classification and ARIMA model. We first used lifelong sentiment classification with Naïve Bayesian framework based on stochastic gradient descent and quantified the text-based reviews. We compared the overall accuracy of our result with baseline methods including NB and SVMs and found out that our method performs better, even though the difference is not extremely large. After we realized the word frequency analysis, we found out that when the star rating is higher than three, the frequency of specific words including "love", "recommend" and "great" is much higher than those whose star ratings are lower than three. We then achieved a comprehensive indicator by using MLR model and therefore determined the data measures based on ratings and reviews that are most informative for Sunshine Company to use. After that, we used ARIMA as time-series model to analyze the underlying factors for the company to use and to predict the data measures in the future. We found the time period where the product's reputation is increasing or decreasing and analyzed the word frequency of those period and found out the patterns and text-based measures. For example, We discovered that three words, "light", "powerful", and "retractable" have the highest frequency in the reviews during these time periods that the product's reputation is increasing, which can in turn indicate that a potentially successful product might have these merits. We also found out that specific star ratings indeed can incite more reviews by combining the result of lifelong

learning and the prediction of the ratings made by using ARIMA model.

Because we used lifelong learning to realize the sentiment classification, our model has the strength of high accuracy, high efficiency and low storage space required. However, because the data set provided is not large enough, and lifelong learning works better with large data sets, the accuracy of our model could have been better if the data sets of more products are provided. We also gave the Sunshine company our advice for them to build an online sales strategy basing on our results, which are included in the letter.

References

[1] Chen, Z., Ma, N. and Liu, B., 2018. Lifelong learning for sentiment classification. arXiv preprint arXiv:1801.02808.

[2] V. Vapnik, The Nature of Statistical Learning Theory, New York:Springer-Verlag, 1995.

[3] Ahmed, B.H. and Ghabayen, A.S., 2020. Review rating prediction framework using deep learning. Journal of Ambient Intelligence and Humanized Computing, pp.1-10.

[4] Stratigi, M., Li, X., Stefanidis, K. and Zhang, Z., 2019, September. Ratings vs. Reviews in Recommender Systems: A Case Study on the Amazon Movies Dataset. In European Conference on Advances in Databases and Information Systems (pp. 68-76). Springer, Cham.

[5] Rish, I., 2001, August. An empirical study of the naive Bayes classifier. In IJCAI 2001 workshop on empirical methods in artificial intelligence (Vol. 3, No. 22, pp. 41-46).

[6] Lewis, D.D., 1998, April. Naive (Bayes) at forty: The independence assumption in information retrieval. In European conference on machine learning (pp. 4-15). Springer, Berlin, Heidelberg.

[7] Pal, M., 2005. Random forest classifier for remote sensing classification. International Journal of Remote Sensing, 26(1), pp.217-222.

[8] Rodriguez-Galiano, V.F., Ghimire, B., Rogan, J., Chica-Olmo, M. and Rigol-Sanchez, J.P., 2012. An assessment of the effectiveness of a random forest classifier for land-cover classification. ISPRS Journal of Photogrammetry and Remote Sensing, 67, pp.93-104.

[9] Thanh Noi, P. and Kappas, M., 2018. Comparison of random forest, k-nearest neighbor, and support vector machine classifiers for land cover classification using Sentinel-2 imagery. Sensors, 18(1), p.18.

[10] Raczko, E. and Zagajewski, B., 2017. Comparison of support vector machine, random forest and neural network classifiers for tree species classification on airborne hyperspectral APEX images. European Journal of Remote Sensing, 50(1), pp.144-154.

[11] Hasan, M.A.M., Nasser, M., Pal, B. and Ahmad, S., 2014. Support vector machine and random forest modeling for intrusion detection system (IDS). Journal of Intelligent Learning Systems and Applications, 2014.

[12] Furey, T.S., Cristianini, N., Duffy, N., Bednarski, D.W., Schummer, M. and Haussler, D., 2000. Support vector machine classification and validation of cancer tissue samples using microarray expression data. Bioinformatics, 16(10), pp.906-914.

[13] Pan, S.J. and Yang, Q., 2009. A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10), pp.1345-1359

[14] Taylor, M.E. and Stone, P., 2009. Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research, 10(Jul), pp.1633-1685.

[15] Zhang, Y. and Yeung, D.Y., 2012. A convex formulation for learning task relationships in multi-task learning. arXiv preprint arXiv:1203.3536.

[16] Caruana, R., 1997. Multitask learning. Machine learning, 28(1), pp.41-75. Bichindaritz, I. and Montani, S., 2012. Report on the eighteenth international conference on case-based reasoning. AI Magazine, 33(1), pp.79-82.

[17] Pang B, Lee L, Vaithyanathan S (2002) Thumbs up?: sentiment classification using machine learning techniques. In: Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10, Association for computational linguistics, pp 79–86

[18] Brockwell, P.J. and Davis, R.A., 2016. Introduction to time series and forecasting. springer.

[19] Montgomery, D.C., Peck, E.A. and Vining, G.G. Introduction to Linear Regression Analysis. John Wiley & Sons, 2012.

[20] Rice, J.A. Mathematical Statistics and Data Analysis. Cengage Learning, 2006.

Appendices

Appendix A LSC Code

```
# coding: utf-8

# In[1]:


import os
import pyprind
import numpy as np
import pandas as pd
from decimal import Decimal
from fractions import Fraction
from time import time
from sklearn.model_selection import StratifiedKFold
import math


natural_dataframes_path = 'D:/Program Files/LSC-project/data/
                                    natural_dataframes/'
balanced_dataframes_path = 'D:/Program Files/LSC-project/data/
                                    balanced_dataframes/'

# numpy ignore overflow
# np.seterr(over=
#
np.seterr(all='raise')

def get_task_names(dataframes_path):
    task_names = []
    for file_name in os.listdir(dataframes_path):
        split_name = os.path.splitext(file_name)[0].split('_')
        if split_name[0][0:1].isupper() == True:
            useful_part = []
            reformed_name = ''
            for part in split_name:
                if part[0:1].isupper() == True:
                    useful_part.append(part)
            for part in useful_part:
                reformed_name += (part + '_')
            reformed_name = reformed_name[:-1]
            if reformed_name not in task_names:
                task_names.append(reformed_name)
    return task_names


#                     k
def split_task(task_no, docs, docs_full_mat, labels, fold):
```

```
    train_docs_lists = []
    train_docs_full_mat_lists = []
    test_docs_lists = []
    test_docs_full_mat_lists = []
    train_labels_lists = []
    test_labels_lists = []
    skf = StratifiedKFold(n_splits=5)
    for train_index, test_index in skf.split(docs, labels):
        train_docs_list, test_docs_list = docs[train_index],
                                          docs[test_index]
        train_docs_full_mat_list, test_docs_full_mat_list =
                                          docs_full_mat[
                                          train_index],
                                          docs_full_mat[test_index
                                          ]
        train_labels_list, test_labels_list = labels[train_index
                                          ], labels[test_index]
        train_docs_lists.append(train_docs_list)
        test_docs_lists.append(test_docs_list)
        train_docs_full_mat_lists.append(
                                          train_docs_full_mat_list
                                          )
        test_docs_full_mat_lists.append(test_docs_full_mat_list)
        train_labels_lists.append(train_labels_list)
        test_labels_lists.append(test_labels_list)
    return np.array(train_docs_lists), np.array(
                                      train_docs_full_mat_lists),
                                      np.array(train_labels_lists)
                                      , np.array(
        test_docs_lists), np.array(test_docs_full_mat_lists), np
                                          .array(test_labels_lists
                                          )


def LSC(dataframes_path):
    #     pbar = pyprind.ProgBar(20)
    fold = 5
    task_names = get_task_names(dataframes_path)
    total_labels = pd.read_csv(dataframes_path + 'total_labels.
                                      csv', index_col=0, header=0)
    total_pos_counts = np.array(pd.read_csv(dataframes_path + '
                                      total_pos_counts.csv',
                                      index_col=0, header=0))
    total_neg_counts = np.array(pd.read_csv(dataframes_path + '
                                      total_neg_counts.csv',
                                      index_col=0, header=0))
    accuracy_sum = 0
    F1_neg_sum = 0
    F1_pos_sum = 0
    for task_no, task_name in enumerate(task_names):
    # task_no = 5
    # task_name = task_names[task_no]
```

```
        print('task number: ', task_no)

        labels_array = total_labels.values[task_no]
        labels_array = labels_array[labels_array != -1]
        docs = pd.read_csv(dataframes_path + task_names[task_no]
                                        + '_docs.csv',
                                        index_col=0, header=0)
        docs_array = docs.values
        print(docs_array.shape)
        train_docs_full_mat = pd.read_csv(dataframes_path +
                                        task_name + '
                                        _docs_full_mat.csv',
                                        index_col=0, header=0,
                                          engine='python').
                                                                values

        train_docs_arrays, train_docs_full_mat_arrays,
                                        train_labels_arrays,
                                        test_docs_arrays,
                                        test_docs_full_mat_arrays
                                        , test_labels_arrays =
                                        split_task(
            task_no, docs_array, train_docs_full_mat,
                                            labels_array, fold)
        for fold_no in range(fold):
            current_task = task(task_no, task_name,
                                            train_docs_arrays[
                                            fold_no],
                                            train_docs_full_mat_arrays
                                            [fold_no],
                                train_labels_arrays[fold_no],
                                                        total_pos_counts
                                                        ,
                                                        total_neg_counts
                                                        )

            # current_task.optimize()
            true_pos, false_pos, true_neg, false_neg =
                                            current_task.test(
                                            test_docs_arrays[
                                            fold_no],
                                                                test_docs_

                                                                test_labe
```

```
            if true_pos == 0:
                true_pos += 1
            if false_pos == 0:
                false_pos += 1
            if true_neg == 0:
                true_neg += 1
            if false_neg == 0:
                false_neg += 1
            print('TP:', true_pos, 'TN:', true_neg, 'FP:',
                                                false_pos, 'FN:',
                                                false_neg)
            #
            p_pos = float(true_pos) / float(true_pos + false_pos
                                                )
            #
            r_pos = float(true_pos) / float(true_pos + false_neg
                                                )
            #
            p_neg = float(true_neg) / float(true_neg + false_neg
                                                )
            #
            r_neg = float(true_neg) / float(true_neg + false_pos
                                                )
            #
            accuracy = float(true_neg + true_pos) / float(
                                                true_pos + true_neg
                                                + false_pos +
                                                false_neg)
            #    F1    -score
            F1_pos = float(2 * p_pos * r_pos) / float(p_pos +
                                                r_pos)
            #    F1    -score
            F1_neg = float(2 * p_neg * r_neg) / float(p_neg +
                                                r_neg)
            print('Negative F1-score:', float(F1_neg))
            print('Accuracy:', float(accuracy))

            accuracy_sum += accuracy
            F1_neg_sum += F1_neg
            F1_pos_sum += F1_pos


    print()
average_accuracy = float(accuracy_sum) / 100
average_F1_neg = float(F1_neg_sum) / 100
average_F1_pos = float(F1_pos_sum) / 100

print('Average F1-score-POS:', float(average_F1_pos))
print('Average F1-score-NEG:', float(average_F1_neg))
print('Average accuracy:', float(average_accuracy))
```

```
#             break
#         break
#         pbar.update()


class task:
    def __init__(self, task_no, task_name, train_docs_array,
                                    train_docs_full_mat_array,
                                    train_labels_array,
                 total_pos_counts, total_neg_counts):

        self.task_no = task_no
        self.task_name = task_name
        self.train_docs_array = train_docs_array
        self.train_docs_full_mat_array =
                                        train_docs_full_mat_array

        self.train_labels_array = train_labels_array

        indices = list(range(20))
        indices.remove(self.task_no)
        self.past_pos_counts = np.take(total_pos_counts, indices
                                        , axis=0)
        self.past_neg_counts = np.take(total_neg_counts, indices
                                        , axis=0)
        self.total_past_pos_counts = self.past_pos_counts.sum(
                                        axis=0)
        self.total_past_neg_counts = self.past_neg_counts.sum(
                                        axis=0)
        cond = train_labels_array == 0
        self.pos_counts = np.compress(cond,
                                        train_docs_full_mat_array
                                        , axis=0).sum(axis=0)
        self.neg_counts = np.compress(~cond,
                                        train_docs_full_mat_array
                                        , axis=0).sum(axis=0)
        task_counts = np.add(self.pos_counts, self.neg_counts)
        self.vocab_size = task_counts[task_counts != 0].shape[0]
        self.smoothing = 1
        self.learning_rate = 10
        self.vs_threshold = 6
        self.vt_threshold = 6
        self.reg_co = 0.1

        # ST
        # self.pos_virtual_counts = np.add(self.
                                        total_past_pos_counts,
                                        self.pos_counts)
        # self.neg_virtual_counts = np.add(self.
                                        total_past_neg_counts,
                                        self.neg_counts)
        # T
```

```
        self.pos_virtual_counts = self.pos_counts
        self.neg_virtual_counts = self.neg_counts
        # S
        # self.pos_virtual_counts = self.total_past_pos_counts
        # self.neg_virtual_counts = self.total_past_neg_counts

        self.pos_virtual_counts = self.pos_virtual_counts.astype
                                        (np.float64)
        self.neg_virtual_counts = self.neg_virtual_counts.astype
                                        (np.float64)
        self.calculate_probs()
        self.calculate_label_prob()
        self.vs = self.calculate_vs()
        self.vt = self.calculate_vt()
        self.vs_full_mat = np.zeros(self.total_vocab_size)
        self.vt_full_mat = np.zeros(self.total_vocab_size)
        np.put(self.vs_full_mat, self.vs, 1)
        np.put(self.vt_full_mat, self.vt, 1)

    def optimize(self):
        #         print('Target domain training set label
                                        probabilities: ', self.
                                        pos_label_prob, self.
                                        neg_label_prob)
        train_docs_df = pd.DataFrame(self.train_docs_array)
        train_docs_df.apply(lambda doc_series: self.optimize_doc
                                        (doc_series), axis=1)

    def calculate_vs(self):
        total_past_counts = np.add(self.past_pos_counts, self.
                                        past_neg_counts)
        self.past_pos_domain_knowledge = np.where(self.
                                        past_pos_counts > 0, 1,
                                        0).sum(axis=0)
        self.past_neg_domain_knowledge = np.where(self.
                                        past_neg_counts > 0, 1,
                                        0).sum(axis=0)
        vs = np.where(np.where(total_past_counts > 0, 1, 0).sum(
                                        axis=0) >= self.
                                        vs_threshold)[0]
        total_counts = np.add(self.pos_counts, self.neg_counts)
        exist_words = np.where(total_counts[total_counts != 0])[
                                        0]
        vs = np.intersect1d(exist_words, vs)
        return vs

    def calculate_vt(self):
        a = self.pos_virtual_probs / self.neg_virtual_probs
        b = self.neg_virtual_probs / self.pos_virtual_probs
        #         print(self.pos_virtual_probs.shape)
        #         print(self.pos_virtual_probs)
        #         print(self.neg_virtual_probs)
```

```
        #         print(a)
        #         print(b)
        vt_pos = np.where(a >= self.vt_threshold)[0]
        vt_neg = np.where(b >= self.vt_threshold)[0]
        vt = np.unique(np.append(vt_pos, vt_neg))
        total_counts = np.add(self.pos_counts, self.neg_counts)
        exist_words = np.where(total_counts[total_counts != 0])[
                                        0]
        vt = np.intersect1d(exist_words, vt)
        return vt

    def optimize_doc(self, doc_series):
        doc_full_mat = self.train_docs_full_mat_array[doc_series
                                        .name]
        doc_label = self.train_labels_array[doc_series.name]
        doc_len = int(np.sum(doc_full_mat[doc_full_mat != 0]))
        initial_objective_function = self.objective_function(
                                        doc_series, doc_full_mat
                                        , doc_label)
        final_objective_function = -2
        #         print("optimize doc: ", 'No.', doc_series.name
                                        , ' label:', doc_label)
        #         print('initial objective function: ',
                                        initial_objective_function
                                        )
        self.objective_function(doc_series, doc_full_mat,
                                        doc_label)
        pos_mul_virtual_counts = self.mul_virtual_counts(
                                        doc_full_mat, 0)
        neg_mul_virtual_counts = self.mul_virtual_counts(
                                        doc_full_mat, 1)

        SGD_round = 0
        while (abs(initial_objective_function -
                                        final_objective_function
                                        ) >= 10 ** -3):
            initial_objective_function = self.objective_function
                                            (doc_series,
                                            doc_full_mat,
                                            doc_label)
            self.pos_starting_points = self.pos_virtual_counts.
                                            copy()
            self.neg_starting_points = self.neg_virtual_counts.
                                            copy()
            doc_series.apply(
                lambda word: self.optimize_word(word, doc_label,
                                                 doc_len,
                                                doc_full_mat,
                                                pos_mul_virtual_counts
                                                ,
                                                neg_mul_virtual_counts
                                                )
```

```
        final_objective_function = self.objective_function(
                                            doc_series,
                                            doc_full_mat,
                                            doc_label)
        SGD_round += 1

#         print('final objective function: ', self.
                                objective_function(
                                doc_series, doc_full_mat,
                                doc_label))
#         print('round: ', SGD_round)

def objective_function(self, doc_series, doc_full_mat,
                                doc_label):
    self.calculate_probs()

    pos_numerator = Decimal(self.pos_label_prob)
    neg_numerator = Decimal(self.neg_label_prob)
    doc_word = np.where(doc_full_mat != 0)[0]
    for word in doc_word:
        pos_numerator *= Decimal(self.pos_virtual_probs[word
                                            ]) ** Decimal(
                                            doc_full_mat[word].
                                            astype(np.float64))
        neg_numerator *= Decimal(self.neg_virtual_probs[word
                                            ]) ** Decimal(
                                            doc_full_mat[word].
                                            astype(np.float64))
    # ratio = 10000
    # pos_numerator = np.prod(np.power((self.
                                    pos_virtual_probs *
                                    ratio), doc_full_mat)) *
                                     self.pos_label_prob
    # neg_numerator = np.prod(np.power((self.
                                    neg_virtual_probs *
                                    ratio), doc_full_mat)) *
                                     self.neg_label_prob

    if pos_numerator == 0:
        print("Warning: ", str(pos_numerator))

    pos = pos_numerator / (pos_numerator + neg_numerator)
    neg = neg_numerator / (pos_numerator + neg_numerator)
    #         print(pos_numerator)
    #         print(neg_numerator)
    if doc_label == 0:
        #             print(pos - neg)
        return pos - neg
    else:
        #             print(neg - pos)
```

```
            return neg - pos

    def optimize_word(self, word, doc_label, doc_len,
                                    doc_full_mat,
                                    pos_mul_virtual_counts,
                                    neg_mul_virtual_counts):
        if word != -1:
            pos_virtual_count = self.pos_virtual_counts[word]
            neg_virtual_count = self.neg_virtual_counts[word]
            #             print('initial virtual counts: ',
                                        pos_virtual_count,
                                        neg_virtual_count)
            #             print(type(self.pos_virtual_counts[
                                        word]), type(self.
                                        neg_virtual_counts[
                                        word]))
            word_freq = doc_full_mat[word]

            a1 = word_freq / (self.smoothing + self.
                                        pos_virtual_counts[
                                        word])
            # print('a1 ', a1)
            a2 = word_freq / (self.smoothing + self.
                                        neg_virtual_counts[
                                        word])
            # print('a2 ', a2)
            #             print('self.pos_label_prob ', self.
                                        pos_label_prob)
            #             print('self.neg_label_prob ', self.
                                        neg_label_prob)
            #             print('pos_mul_virtual_counts ',
                                        pos_mul_virtual_counts
                                        )
            #             print('neg_mul_virtual_counts ',
                                        pos_mul_virtual_counts
                                        )
            # math.isinf(b1)
            b1 = float(self.neg_label_prob) / float(self.
                                        pos_label_prob) *
                                        float(
                                        pos_mul_virtual_counts
                                        )
            if math.isinf(b1) or b1 == float(0):
                b1 = Decimal(self.neg_label_prob) / Decimal(self
                                            .pos_label_prob)
                                             * Decimal(
                                            pos_mul_virtual_counts
                                            )
            # print('b1 ', b1)
            b2 = float(self.pos_label_prob) / float(self.
                                        neg_label_prob) *
                                        float(
```

```
                                        neg_mul_virtual_counts
                                        )
    if math.isinf(b2) or b2 == float(0):
        b2 = Decimal(self.pos_label_prob) / Decimal(self
                                            .neg_label_prob)
                                             * Decimal(
                                            neg_mul_virtual_counts
                                            )
    # print('b2 ', b2)
    pos_sum = self.sum_virtual_counts(0)
    # print('pos_sum ', pos_sum)
    neg_sum = self.sum_virtual_counts(1)
    # print('neg_sum ', neg_sum)
    pos = float(self.smoothing * self.vocab_size) +
                                        float(pos_sum)
    # print('pos ', pos)
    neg = float(self.smoothing * self.vocab_size) +
                                        float(neg_sum)
    # print('neg ', neg)
    c1 = self.g_derivatives(0, doc_len, pos, neg)
    # print('c1 ', c1)
    c2 = self.g_derivatives(1, doc_len, pos, neg)
    # print('c2 ', c2)
    g = self.g(doc_len, pos_sum, neg_sum)
    # print('g ', g)

    if self.vs_full_mat[word] == 1:
        vs_pos, vs_neg = self.calculate_vs_derivatives()
    else:
        vs_pos, vs_neg = 0, 0

    # print('vs_pos ', vs_pos)
    # print('vs_neg ', vs_neg)

    if self.vt_full_mat[word] == 1:
        vt_pos, vt_neg = self.calculate_vt_derivatives()
    else:
        vt_pos, vt_neg = 0, 0

    # print('vs_pos ', vt_pos)
    # print('vs_neg ', vt_neg)

    # derivatives_pos = 0
    # derivatives_neg = 0
    try:
        if doc_label == 0:
            derivatives_pos = (a1 + b1 * c1) / (1 + b1 *
                                                g) - a1 +
                                               vt_pos +
                                               vs_pos
            derivatives_neg = (a2 * g + c2) / (b2 + g) +
                                                vt_neg +
```

```
                                        vs_neg
            elif doc_label == 1:
                derivatives_pos = (a1 + b1 * c1) / (1 + b1 *
                                                    g) - (c1 /
                                                   g) + vt_pos
                                                   + vs_pos
                derivatives_neg = (a2 * g + c2) / (b2 + g) -
                                                    (a2 + c2 /
                                                   g) + vt_neg
                                                   + vs_neg
        except TypeError:
            a1 = Decimal(a1)
            a2 = Decimal(a2)
            b1 = Decimal(b1)
            b2 = Decimal(b2)
            c1 = Decimal(c1)
            c2 = Decimal(c2)
            vs_pos = Decimal(vs_pos)
            vs_neg = Decimal(vs_neg)
            vt_pos = Decimal(vt_pos)
            vt_neg = Decimal(vt_neg)
            g = Decimal(g)
            if doc_label == 0:
                derivatives_pos = (a1 + b1 * c1) / (1 + b1 *
                                                    g) - a1 +
                                                   vt_pos +
                                                   vs_pos
                derivatives_neg = (a2 * g + c2) / (b2 + g) +
                                                    vt_neg +
                                                   vs_neg
            elif doc_label == 1:
                derivatives_pos = (a1 + b1 * c1) / (1 + b1 *
                                                    g) - (c1 /
                                                   g) + vt_pos
                                                   + vs_pos
                derivatives_neg = (a2 * g + c2) / (b2 + g) -
                                                    (a2 + c2 /
                                                   g) + vt_neg
                                                   + vs_neg
            derivatives_pos = float(derivatives_pos)
            derivatives_neg = float(derivatives_neg)

        if pos_virtual_count - self.learning_rate *
                                        derivatives_pos > 0:
            #                   print('pos_change ', self.
                                            learning_rate *
                                            derivatives_pos)
            self.pos_virtual_counts[word] -= float(self.
                                            learning_rate *
                                            derivatives_pos)
        if neg_virtual_count - self.learning_rate *
                                        derivatives_neg > 0:
```

```
                #                   print('neg_change ', self.
                                                learning_rate *
                                                derivatives_neg)
                self.neg_virtual_counts[word] -= float(self.
                                                learning_rate *
                                                derivatives_neg)

#               print('final virtual counts: ', self.
                                pos_virtual_counts[word],
                                self.neg_virtual_counts[word
                                ])
#               print(type(self.pos_virtual_counts[word]),
                                type(self.neg_virtual_counts
                                [word]))

    def g(self, doc_len, pos, neg):
        try:
            a = pos / neg
            result = a ** doc_len
        except OverflowError:
            result = Decimal(a) ** Decimal(doc_len)
        if math.isinf(result):
            result = Decimal(a) ** Decimal(doc_len)
        return result

    def g_derivatives(self, label_num, doc_len, pos, neg):
        try:
            if label_num == 0:
                result = float(doc_len * ((pos / neg) ** (
                                                doc_len - 1)) *
                                                (1 / neg))
            else:
                result = float(doc_len * ((pos / neg) ** (
                                                doc_len - 1)) *
                                                (- (pos / (neg
                                                ** 2))))
        except OverflowError:
            if label_num == 0:
                result = Decimal(doc_len) * (Decimal(pos / neg)
                                                ** Decimal(
                                                doc_len - 1)) *
                                                (Decimal(1) /
                                                Decimal(neg))
            else:
                result = Decimal(doc_len) * (Decimal(pos / neg)
                                                ** Decimal(
                                                doc_len - 1)) *
                                                (Decimal(-1) * (
                                                Decimal(pos) / (
                                                Decimal(neg) **
                                                Decimal(2))))
        if math.isinf(result):
```

```
            if label_num == 0:
                result = Decimal(doc_len) * (Decimal(pos / neg)
                                             ** Decimal(
                                             doc_len - 1)) *
                                             (Decimal(1) /
                                             Decimal(neg))
            else:
                result = Decimal(doc_len) * (Decimal(pos / neg)
                                             ** Decimal(
                                             doc_len - 1)) *
                                             (Decimal(-1) * (
                                             Decimal(pos) / (
                                             Decimal(neg) **
                                             Decimal(2))))
        return result

    def calculate_vs_derivatives(self):
        past_pos_domain_knowledges = np.take(self.
                                             past_pos_domain_knowledge
                                             , self.vs)
        past_neg_domain_knowledges = np.take(self.
                                             past_neg_domain_knowledge
                                             , self.vs)
        needed_pos_virtual_counts = np.take(self.
                                             pos_virtual_counts, self
                                             .vs)
        needed_neg_virtual_counts = np.take(self.
                                             neg_virtual_counts, self
                                             .vs)
        needed_pos_starting_points = np.take(self.
                                             pos_starting_points,
                                             self.vs)
        needed_neg_starting_points = np.take(self.
                                             neg_starting_points,
                                             self.vs)

        past_pos_domain_knowledges += 1
        past_neg_domain_knowledges += 1
        c = past_pos_domain_knowledges.astype(np.float64)
        c *= 1. / np.add(past_pos_domain_knowledges,
                                             past_neg_domain_knowledges
                                             )
        vs_pos = (needed_pos_virtual_counts - c *
                                             needed_pos_starting_points
                                             ).sum()
        #         c      c      - 1
        c -= 1
        vs_neg = (needed_neg_virtual_counts + c *
                                             needed_neg_starting_points
                                             ).sum()
        vs_pos *= self.reg_co
        vs_neg *= self.reg_co
```

```
        #         self.past_neg_counts = np.take(
                                        total_neg_counts,
                                        indices, axis=0)
        #         for word in self.vs:
        #             pos_dom = self.past_pos_domain_knowledge[
                                        word] + 1
        #             neg_dom = self.past_neg_domain_knowledge[
                                        word] + 1
        #             c = float(pos_dom) / float(pos_dom +
                                        neg_dom)
        #             vs_pos += (float(self.pos_virtual_counts[
                                        word]) - c * float(self.
                                        pos_starting_points[word
                                        ]))
        #             vs_neg += (float(self.neg_virtual_counts[
                                        word]) - (1 - c) * float
                                        (self.
                                        neg_starting_points[word
                                        ]))
        #         vs_pos *= self.reg_co
        #         vs_neg *= self.reg_co
        return vs_pos, vs_neg

    def calculate_vt_derivatives(self):
        needed_past_pos_counts = np.take(self.
                                        total_past_pos_counts,
                                        self.vt)
        needed_past_neg_counts = np.take(self.
                                        total_past_neg_counts,
                                        self.vt)
        vt_pos = needed_past_pos_counts.sum()
        vt_neg = needed_past_neg_counts.sum()
        vt_pos *= self.reg_co
        vt_neg *= self.reg_co

        #         for word in self.vt:
        #             vt_pos += (self.total_past_pos_counts[word
                                        ])
        #             vt_neg += (self.total_past_neg_counts[word
                                        ])
        #         vt_pos *= self.reg_co
        #         vt_neg *= self.reg_co
        return vt_pos, vt_neg

    def sum_virtual_counts(self, label_num):
        sum_virtual_counts = 0
        if label_num == 0:
            sum_virtual_counts = np.sum(self.pos_virtual_counts)
        elif label_num == 1:
            sum_virtual_counts = np.sum(self.neg_virtual_counts)
        return float(sum_virtual_counts)
```

```
    def mul_virtual_counts(self, doc_full_mat, label_num):
        result = Decimal(1)
        if label_num == 0:
            a = (self.neg_virtual_counts + self.smoothing) / (
                                                self.
                                                pos_virtual_counts +
                                                 self.smoothing)
        elif label_num == 1:
            a = (self.pos_virtual_counts + self.smoothing) / (
                                                self.
                                                neg_virtual_counts +
                                                 self.smoothing)

        # print(a.shape)
        doc_word = np.where(doc_full_mat != 0)[0]
        for word in doc_word:
            result *= Decimal(a[word]) ** Decimal(doc_full_mat[
                                                word].astype(np.
                                                float64))

        # mul_virtual_counts = np.prod(np.power(a, doc_full_mat)
                                            )
        return result

    def calculate_label_prob(self):
        total_labels = self.train_labels_array.shape[0]
        neg_labels = int(self.train_labels_array.sum())
        self.neg_label_prob = float(neg_labels / total_labels)
        self.pos_label_prob = float((total_labels - neg_labels)
                                            / total_labels)

    def calculate_probs(self):
        self.pos_virtual_probs = self.pos_virtual_counts.copy()
        self.neg_virtual_probs = self.neg_virtual_counts.copy()
        total_pos_counts = int(self.pos_virtual_counts.sum())
        total_neg_counts = int(self.neg_virtual_counts.sum())
        self.total_vocab_size = self.pos_virtual_counts.shape[0]
        self.pos_virtual_probs += 1
        self.neg_virtual_probs += 1
        pos_denominator = np.float64(self.smoothing * self.
                                            total_vocab_size +
                                            total_pos_counts)
        neg_denominator = np.float64(self.smoothing * self.
                                            total_vocab_size +
                                            total_neg_counts)
        self.pos_virtual_probs /= pos_denominator
        self.neg_virtual_probs /= neg_denominator

    def test(self, test_docs_arrays, test_docs_full_mat_arrays,
                                        test_labels_arrays):
```

```
        test_labels_arrays = test_labels_arrays[
                                        test_labels_arrays != -1
                                        ]
        self.test_results = np.zeros(test_labels_arrays.shape[0]
                                        )
        test_docs_df = pd.DataFrame(test_docs_arrays)
        test_docs_df.apply(
            lambda doc_series: self.calculate_label(doc_series,
                                            test_docs_full_mat_arrays
                                            [doc_series.name]),
                                            axis=1)
        test_booleans = np.equal(test_labels_arrays, self.
                                        test_results)
        true_indices = np.where(test_booleans == True)[0]
        false_indices = np.where(test_booleans == False)[0]
        true_neg = np.take(test_labels_arrays, true_indices).sum
                                        ()
        true_pos = true_indices.shape[0] - true_neg
        false_neg = np.take(test_labels_arrays, false_indices).
                                        sum()
        false_pos = false_indices.shape[0] - false_neg
        return true_pos, false_pos, true_neg, false_neg

    def calculate_label(self, doc_series, doc_full_mat):
        # doc_len = int(np.sum(doc_full_mat[doc_full_mat != 0]))
        pos_numerator = Decimal(self.pos_label_prob)
        neg_numerator = Decimal(self.neg_label_prob)
        doc_word = np.where(doc_full_mat != 0)[0]
        for word in doc_word:
            pos_numerator *= Decimal(self.pos_virtual_probs[word
                                            ]) ** Decimal(
                                            doc_full_mat[word].
                                            astype(np.float64))
            neg_numerator *= Decimal(self.neg_virtual_probs[word
                                            ]) ** Decimal(
                                            doc_full_mat[word].
                                            astype(np.float64))
        # ratio = 10000
        # pos_numerator = np.prod(np.power((self.
                                        pos_virtual_probs *
                                        ratio), doc_full_mat)) *
                                         self.pos_label_prob
        # neg_numerator = np.prod(np.power((self.
                                        neg_virtual_probs *
                                        ratio), doc_full_mat)) *
                                         self.neg_label_prob
        if pos_numerator < neg_numerator:
            self.test_results[doc_series.name] = 1

start = time()
LSC(natural_dataframes_path)
stop = time()
```

```
print('time: ', str(stop-start))
#\n# %debug')
```

Appendix B Data

```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,
                                confusion_matrix, accuracy_score


#          tsv
# hair_dryer = pd.read_csv('hair_dryer.tsv', sep='\t')
# microwave = pd.read_csv('microwave.tsv', sep='\t')
pacifier = pd.read_csv('pacifier.tsv', sep='\t')
# data1 = pacifier.to_csv("pacifier.csv")
# data2 = pacifier.to_csv("pacifier.csv")
# data3 = pacifier.to_csv("pacifier.csv")
# print(pacifier.columns.values.tolist())

#
#           ['marketplace', 'customer_id', 'review_id', '
                                product_id', 'product_parent',
# 'product_title', 'product_category', 'star_rating', '
                                helpful_votes',
# 'total_votes', 'vine', 'verified_purchase', 'review_headline',
                                 'review_body', 'review_date']
pacifier.drop('marketplace',axis = 1,inplace = True)
pacifier.drop('review_id',axis = 1,inplace = True)
pacifier.drop('product_id',axis = 1,inplace = True)
pacifier.drop('product_title',axis = 1,inplace = True)
pacifier.drop('product_category',axis = 1,inplace = True)
pacifier.drop('star_rating',axis = 1,inplace = True)
pacifier.drop('helpful_votes',axis = 1,inplace = True)
pacifier.drop('total_votes',axis = 1,inplace = True)
pacifier.drop('vine',axis = 1,inplace = True)
pacifier.drop('verified_purchase',axis = 1,inplace = True)
pacifier.drop('review_date',axis = 1,inplace = True)
#           ['customer_id', 'product_parent', 'review_headline',
                                 'review_body']
# print(pacifier.columns.values.tolist())
```

```
# print(pacifier)

#    review
pacifier['blanket'] = pd.Series(' ', index=pacifier.index)
pacifier['reviewgo'] = pacifier['review_headline'] + pacifier['
                                blanket'] + pacifier['
                                review_body']
pacifier['review'] = pacifier['reviewgo'] + pacifier['blanket']
                                + pacifier['review_headline']
# print(pacifier.columns.values.tolist())
pacifier.drop('reviewgo', axis=1, inplace=True)
pacifier.drop('blanket', axis=1, inplace=True)
pacifier.drop('review_headline', axis=1, inplace=True)
pacifier.drop('review_body', axis=1, inplace= True)
# print(pacifier.columns.values.tolist())
# print(pacifier)


#
labels = []
# labels = pacifier.iloc[:,2].values
features = pacifier.iloc[:, 2].values #       d      3
processed_features = []


for sentence in range(0, len(features)):
    # Remove all the special characters
    processed_feature = re.sub(r'\W', ' ', str(features[sentence
                                ]))
    # remove all single characters
    processed_feature= re.sub(r'\s+[a-zA-Z]\s+', ' ',
                                processed_feature)
    # Remove single characters from the start
    processed_feature = re.sub(r'\^[a-zA-Z]\s+', ' ',
                                processed_feature)
    # Substituting multiple spaces with single space
    processed_feature = re.sub(r'\s+', ' ', processed_feature,
                                flags=re.I)
    # Removing prefixed 'b'
    processed_feature = re.sub(r'^b\s+', '', processed_feature)
    # Converting to Lowercase
    processed_feature = processed_feature.lower()
    #
    r4 = u'[a-zA-Z0- 9  !"#$%&\'()*+,-./:;<=>?  @    ?
                                ...()+_=  `"  %^&*! ?[\\]^_`
                                {|}~]+'
    processed_feature = re.sub(r4, '', processed_feature)
    #
    processed_feature = [w for w in processed_feature.split(' ')
                                 if w not in stopwords.words
                                ('english')]
    #             (stemming)
```

```
    stemmer = SnowballStemmer("english")
    stemmer.stem('sentence')


    processed_features.append(processed_feature)

# ssssss
# vectorizer = TfidfVectorizer(max_features=2500, min_df=7,
                                max_df=0.8,stop_words=stopwords.
                                words('english'),input='list')
# processed_features = vectorizer.fit_transform(
                                processed_features).toarray()

datanew = pacifier.to_csv("pacifier_initial.csv")
```

Appendix C WordFreq

```
r1='!@$%^&*()_+=-|\}]{["':;?/>.<,]}'
speech_etxt = re.sub(r1,'',speech_etxt)
stemmer = SnowballStemmer("english")
stemmer.stem(speech_etxt)
speech = speech_etxt.lower()
speech = [w for w in speech.split(' ') if w not in stopwords.
                                words('English')]
dic = {}
for word in speech:
    if word not in dic:
        dic[word] = 1
    else:
        dic[word] = dic[word] + 1

swd = sorted(dic.items(), key=operator.itemgetter(1), reverse=
                                True)
print(swd)
```

Appendix D ARIMA

```
R for ARIMA
micro<-read.table('micro.csv')
micro.ts<-ts(micro,start=2002,end=2015)
plot.ts(micro.ts)
library("forecast")
auto.arima(micro.ts)
micro.arima<-arima(micro,order=c(2,1,0))
micro.arima
micro.tsdiff1<-diff(micro.ts,difference=1)
```

```
plot.ts(micro.tsdiff1)
acf(micro.tsdiff1,lag.max=40)
pacf(micro.tsdiff1,lag.max=20)
micro.re<-micro.arima$residuals
Box.test(micro.re,  lag=20, type="Ljung-Box")
library("tseries")
acf((micro.re-mean(micro.re))^2)
micro.regarch<-garch(micro.re,grad = "numerical",trace = FALSE)
confint(micro.regarch)
```